Universidad de las Ciencias Informáticas

Facultad 7

Título: Infraestructura de firma y validación digital de los documentos clínicos electrónicos generados por el sistema alas HIS

Trabajo de Diploma para optar por el título de Ingeniero Informático

Autor: David Ricardo Ledo Báster

Tutor: Ing. Alejandro Mario Velázquez Carralero

Co-tutor: Ing. Rodney Ledo Ramírez

Ciudad de la Habana, junio de 2011

“Año del 53 aniversario del triunfo de la Revolución”

DECLARACIÓN DE AUTORÍA

Declaro que soy el único autor de este trabajo y autorizo al _____ de la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio. Para que así conste firmo la presente a los ____ días del mes de _____ del año _____.

David Ricardo Ledo Báster

_____

Firma de Autor

Ing. Alejandro Mario Velázquez Carralero

_____

Firma del Tutor

Ing. Rodney Ledo Ramírez

_____

Firma del Co-tutor

DATOS DE CONTACTO

Datos del tutor

Nombre: Ing. Alejandro Mario Velázquez Carralero

Correo electrónico: amvelazquez@uci.cu

Datos del co-tutor

Nombre: Ing. Rodney Ledo Báster

Correo electrónico: rledo@uci.cu

Datos del autor

Nombre: David Ricardo Ledo Báster

Correo electrónico: drledo@estudiantes.uci.cu

AGRADECIMIENTOS

A mis tutores, Alejandro y Rodney, por haber confiado ciegamente en mi trabajo. A los profesores Ferrás, Reynel, Soto y David por permitirme interrumpirlos por una que otra duda. A todos los miembros del tribunal por sus señalamientos y consejos.

DEDICATORIA

A mi mamá, mi papá y mis abuelos por ser las personas que más admiro y respeto. A Leandro, Norge, Mansolo y Carlos David, a todos mis amigos en general. A mi novia, y futura esposa, por su paciencia y tolerancia.

RESUMEN

En el presente trabajo se hace un estudio de la de firma digital como mecanismo de seguridad y se desarrolla una infraestructura para la firma y validación digital de los documentos clínicos electrónicos generados en las instituciones hospitalarias por el sistema alas HIS, que garantiza la autenticidad, integridad y no repudio de la información almacenada en dichos documentos. Además se hace uso del servicio de sellado de tiempo para garantizar la validez de la firma digital a largo plazo.

El desarrollo de la solución está guiado por el Proceso Unificado de Desarrollo y se basa en tecnologías libres y multiplataforma . Se utiliza Java como lenguaje de programación sobre una arquitectura basada en componentes. Como Sistema de Gestión de Bases de Datos se hace uso de PostgreSQL y como servidor de aplicaciones el JBoss Server.

La infraestructura posibilita la creación y actualización de las identidades digitales de los usuarios y gestiona, de manera centralizada, los niveles de confianza en los certificados digitales de las entidades hospitalarias para una futura validación de la firma.

PALABRAS CLAVES

firma digital, validación digital, documentos clínicos electrónicos, certificados digitales, identidades digitales, niveles de confianza.

Índice de contenido

INTRODUCCIÓN

El desarrollo de la sociedad compromete habilidades y capacidades de todas las estructuras sociales en el manejo de información y la comunicación de diversas formas y maneras dirigidas a lograr mejores niveles de eficacia y eficiencia.

El avance de las tecnologías de la información y la comunicaciones (TIC) irrumpen, cada vez con más fuerza, en el marco de la salud y de la medicina. La informatización del sector involucra procesos de sistematización y automatización en el tratamiento de la información y las comunicaciones de las actividades de financiamiento, gestión y en las prácticas médicas y de salud.

Las necesidades crecientes de intercambio de información en los sistemas sanitarios, demandan estándares para el desarrollo de sistemas integrados, interactivos, fiables, económicos y seguros. En sanidad como en otros campos de aplicaciones telemáticas existe una corriente general de demanda de los usuarios hacia sistemas abiertos, distribuidos e interconectados, con un alto grado de fiabilidad y requisitos de seguridad cada vez más exigentes. La gestión integrada de los servicios sanitarios y la continuidad de las atenciones médicas requieren la adopción de mensajes, formatos, codificación y estructura de registros médicos aceptados de forma generalizada que sean interpretados correctamente por dichos sistemas.

Así surgen los llamados Sistemas de Información Hospitalario (HIS por sus siglas en inglés), que satisfacen problemas como el procesamiento de datos médicos y el almacenamiento de la información resultante de los procesos hospitalarios médico-administrativos de las distintas áreas que abarca. Estas situaciones han convertido a los Sistemas de Información Hospitalaria en uno de los frentes fundamentales en la búsqueda de la informatización de procesos y servicios, que redunden, a su vez, en una mejor atención a las personas.

En la actualidad las instituciones hospitalarias manipulan gran cantidad de información de vital importancia para el paciente. Hoy se habla de una Historia Clínica Electrónica como el conjunto de documentos que contienen los datos, valoraciones, atenciones, observaciones, tratamientos e informaciones de cualquier índole sobre la situación clínica de un paciente a lo largo del proceso asistencial durante toda su vida aportando grandes ventajas, tanto en costos como en acceso a la información, sobre la forma tradicional de la Historia Clínica [1]. La historia clínica en papel puede perderse con más facilidad que si está informatizada, con una forma de registro que impida que pueda borrarse o modificarse. También, la historia clínica electrónica evita tener que repetir

pruebas como las radiografías o escáneres, que se podrían ver desde un ordenador o puesto de trabajo.

Los documentos clínicos, que conforman la Historia Clínica Electrónica, poseen características que deben preservarse:

- Inviolabilidad : Que la información no pueda ser adulterada.
- Autoría: Identificación del responsable que la generó.
- Confidencialidad: No puede difundirse libremente, tiene que tener control de quienes tienen acceso.
- Secuencialidad: Debe seguir el orden en que fue escrita.
- Disponibilidad: Debe garantizar la posibilidad de consulta cuando el paciente y los profesionales lo necesiten en tiempo y forma.
- Integridad: Los que están justificadamente habilitados deben poder acceder a toda la información que se requiera para el acto médico, así como para la auditoría, estadísticas, epidemiología, planes de prevención y peritajes legales.
- Temporalidad precisa: Fecha y hora en que se generó.
- Durabilidad: Debe permanecer inalterable en el tiempo para que su información pueda ser consultada.

Con la forma tradicional de la historia clínica, el cumplimiento de las características antes mencionadas es muy rústico y arriesgado. El doctor cuando genera un nuevo documento clínico le estampa su firma hológrafa, que lo identifica, y lo envía, junto a los demás documentos clínicos del paciente, a un archivo físico donde puede ser fácilmente modificado sin su consentimiento y su firma puede ser falsificada.

Teniendo en cuenta lo antes expuesto es necesario el empleo de un instrumento que cumpla con rigor las características señaladas.

La firma hológrafa le otorga autoría a una determinada persona sobre un documento, y es reconocida legalmente; pero su uso es inútil en los documentos con soporte de almacenamiento

digital, como son los documentos clínicos que conforman la Historia Clínica Electrónica. Para solucionar este problema se utiliza la firma digital.

La firma digital básicamente garantiza:

- La autoría (no repudio).
- Inviolabilidad e integridad de la información.
- Temporalidad.

La firma digital es el medio para garantizar la seguridad de la HCE, para que sus datos no puedan ser modificados por cualquier persona no autorizada. Permite recibir y emitir información segura. La autoría se logra mediante atributos de la firma digital por medio de las técnicas criptográficas asimétricas, que encriptan el texto con la clave privada del autor de la información. Las técnicas de fechado o time stamping, son las que permiten saber cuando fue firmado el documento.

En general, la firma digital provee al usuario la seguridad de que la información no ha sido modificada luego de su firma (integridad), la seguridad de que el documento y su firma se corresponden con la persona que ha firmado (autenticidad), la seguridad de que la persona que ha firmado no puede decir que no lo ha hecho (no repudio), la seguridad de que la información contendida ha sido cifrada y la voluntad del emisor, solo permite que el receptor que él determine pueda descifrarla (confidencialidad).

Los sistemas informáticos de gestión de la información deben contar ante todo con una adecuada seguridad, que no solo esté basada en el uso de usuarios, contraseñas y gestión de roles, sino que permita la garantía de que la información que se genere sea representada unívocamente por su autor además de detectar cambios no autorizados. En los Sistemas de Información Hospitalaria esta característica permite depurar responsabilidades así como detectar la modificación o alteración de la información lo que es muy importante para el correcto tratamiento del paciente.

Actualmente el sistema alas HIS, Sistema de Información Hospitalario desarrollado por la Universidad de Ciencias Informáticas, no posee una infraestructura que permita la firma y validación digital para garantizar la integridad, autenticidad, autoría y temporalidad de la información clínica que se genere en las entidades hospitalarias. Este trabajo surge para darle solución a la situación antes expuesta, por lo que el problema a resolver queda formulado de la siguiente forma:

¿Cómo desarrollar una infraestructura de firma y validación digital que garantice la integridad, temporalidad, autenticidad y autoría de los documentos clínicos electrónicos que se generen en las entidades hospitalarias pos el sistema alas HIS?

Se define como **objeto de estudio** la infraestructura de firma y validación digital como elemento de seguridad y **el campo de acción** queda determinado por los procesos de firma y validación digital de los documentos clínicos electrónicos generados en las instituciones hospitalarias por el sistema alas HIS.

Objetivo general:

Desarrollar los procesos de firma y validación digital de los documentos clínicos electrónicos, basado en estándares y protocolos de comunicación reconocidos internacionalmente, generados por el sistema alas HIS.

Para dar cumplimiento a los objetivos anteriormente planteados se definen las siguientes **tareas de la investigación:**

- Identificar los estándares más utilizados para la firma digital de documentos.
- Valorar la arquitectura y metodología definida por el Departamento de Gestión Hospitalaria para el desarrollo de sus aplicaciones.
- Diseñar la infraestructura de firma y validación digital.
- Implementar la infraestructura de firma y validación digital.
- Garantizar la integración de la infraestructura con el Visor de Historias Clínicas del sistema alas HIS.

Para un mayor entendimiento el documento estará estructurado de la manera siguiente:

Capítulo 1: Fundamentación Teórica: Estudio preliminar de los principales conceptos, estándares y algoritmos. Tecnologías y herramientas de desarrollo a utilizar.

Capítulo 2: Características del sistema: Descripción del negocio a automatizar e información que se maneja. Descripción de la arquitectura.

Capítulo 3: Descripción y análisis de la solución: Descripción y análisis de la solución que se

propone para dar respuesta a la problemática planteada.

Capítulo 4: Modelo de pruebas: Se describe el método de prueba a utilizar y se describen los casos de prueba .

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

Introducción

En este capítulo se hace referencia a los principales elementos a tener en cuenta para desarrollar una infraestructura de software que garantice la integridad y autenticidad de documentos digitales, se ofrecen algunos conceptos de importancia para darle cumplimiento al objetivo del trabajo. También se realiza un estudio del arte de las principales metodologías, plataformas de desarrollo y lenguajes de programación existentes, mostrando las características de cada uno que lo diferencia del resto.

1.1. Criptografía

La palabra “Criptografía” proviene de las palabras griegas “kryptos” relativo a oculto y “gráphein” relativo a escritura y la definición que el diccionario de la Real Academia de la Lengua Española da sobre ella es: “Arte de escribir con clave secreta o de un modo enigmático” [2]. Visto desde otro punto de vista la criptografía es la ciencia o técnica que permite proteger la información por medio de la aplicación de un método de cifrado.

La historia de la criptografía comienza cuando las primeras civilizaciones desarrollaron técnicas para enviar mensajes durante las campañas militares, de forma que si el mensajero era interceptado la información que portaba no corriera el peligro de caer en manos del enemigo. El primer método de criptografía fue en el siglo V antes de Cristo, era conocido como "Escítala". El segundo criptosistema que se conoce fue documentado por el historiador griego Polibio: un sistema de sustitución basado en la posición de las letras en una tabla.

También los romanos utilizaron sistemas de sustitución, siendo el método actualmente conocido como César, porque supuestamente Julio César lo empleó en sus campañas, uno de los más conocidos en la literatura (según algunos autores, en realidad Julio César no usaba este sistema de sustitución, pero la atribución tiene tanto arraigo que el nombre de este método de sustitución ha quedado para los anales de la historia). Otro de los métodos criptográficos utilizados por los griegos fue la escítala espartana, un método de transposición basado en un cilindro que servía como clave en el que se enrollaba el mensaje para poder cifrar y descifrar. Estos métodos conforman la llamada Criptografía clásica [3].

La Criptografía moderna nace al mismo tiempo que las computadoras. Durante la Segunda Guerra Mundial, en un lugar llamado Bletchley Park, un grupo de científicos entre los que se encontraba Alan Turing, trabajaba en el proyecto ULTRA tratando de descifrar los mensajes enviados por el

ejército alemán con el más sofisticado ingenio de codificación ideado hasta entonces: la máquina ENIGMA. Este grupo de científicos empleaba lo que hoy se considera el primer computador aunque esta información permaneció en secreto hasta mediados de 1970. Su uso y la llegada del polaco Marian Rejewski tras la invasión de su país natal cambiarían para siempre el curso de la Historia [3].

1.1.1. Criptografía de clave privada o simétrica

Los criptosistemas simétricos, también conocidos como de llave única, basan su fortaleza en el secreto de una llave “k”. De modo que, si esta llave se llega a descubrir, hallar la información inicial a partir de la cifrada sería una tarea teóricamente fácil.

Figura 1.1: Proceso de encriptación con clave simétrica

La llave “k” es secreta y está compartida por los dos usuarios, el transmisor y el receptor, y es así como se consigue la confidencialidad e integridad en este tipo de sistemas: Solamente el receptor autorizado conocerá la llave con la que ha sido cifrado el mensaje; otro cualquiera al no conocerla, no podría interpretarlo.

1.1.1.1. Algoritmos de clave privada

DES

DES (Data Encryption Standard) es un esquema de cifrado simétrico desarrollado en 1977 por el Departamento de Comercio y la Oficina Nacional de Estándares de EEUU en colaboración con la empresa IBM, que se creó con objeto de proporcionar al público en general un algoritmo de cifrado normalizado para redes de ordenadores. Estaba basado en la aplicación de todas las

teorías criptográficas existentes hasta el momento de su surgimiento, y fue sometido a las leyes de Estados Unidos de América. Posteriormente se creó una versión de DES implementada por hardware, que entró a formar parte de los estándares de la ISO con el nombre de DEA. DES fue el algoritmo más utilizado en el mundo hasta la década del 90 del siglo pasado.

Triple DES

El sistema DES se considera en la actualidad poco práctico, debido a la corta longitud de su llave. Para solventar este problema y continuar utilizando DES se creó el sistema Triple DES (TDES), basado en tres iteraciones sucesivas del algoritmo DES, con lo que se consigue una longitud de llave de 128 bits, y que es compatible con DES simple. Este hecho se basa en que DES tiene la característica matemática de no ser un grupo, lo que implica que si se cifra el mismo bloque dos veces con dos llaves diferentes se aumenta el tamaño efectivo de la llave.

AES

Advanced Encryption Standard (AES), también conocido como Rijndael, es un esquema de cifrado por bloques adoptado como un estándar de cifrado por el gobierno de los Estados Unidos. Se espera que sea usado en el mundo entero y analizado exhaustivamente, como fue el caso de su predecesor, el Data Encryption Standard (DES). NIST (National Institute of Standards and Technology), publicó las especificaciones para el AES a finales de 1997. Es un algoritmo simétrico de cifrado por bloques, donde las longitudes del bloque y de la llave son variables. Los bloques adoptados por el estándar son de 128 bits (no de 64 bits como el DES) y las llaves de 128, 192 o 256 bits (no 56 bits como el DES o 112 bits como el Triple DES).

IDEA

Sistema criptográfico simétrico, creado en 1990 por Lai y Massey, que trabaja con bloques de texto de 64 bits, operando siempre con números de 16 bits usando operaciones como OR Exclusivo y suma y multiplicación de enteros. El algoritmo de descifrado es muy parecido al de cifrado, por lo que resulta muy fácil y rápido de programar. Este algoritmo es de libre difusión y no está sometido a ningún tipo de restricciones o permisos nacionales, por lo que se ha difundido ampliamente, utilizándose en sistemas como UNIX y en programas de cifrado de correo como PGP.

1.1.2. Criptografía de clave pública o asimétrica

1.1.2.1. Orígenes de las infraestructuras de clave pública o asimétrica.

Hasta hace poco menos de cuarenta años los sistemas de cifrado estaban basados en criptografía simétrica. En estos sistemas, las dos partes que desean intercambiar un mensaje deben compartir, y mantener en absoluto secreto, una clave acordada de antemano, mediante un método no criptográfico; por ejemplo comunicándosela cara a cara o usando un método de entrega convencional certificado. Este tipo de sistemas presentan un gran número de dificultades prácticas a la hora de realizar la distribución de claves. La invención de la criptografía de clave pública solucionó estos problemas; con la criptografía de clave pública los usuarios pueden comunicarse de manera segura a través de un canal inseguro sin necesidad de acordar una clave de antemano.

En 1976 Whitfield Diffie y Martin Hellman, influenciados por los trabajos de Ralph Merkle en el campo de la distribución de claves publicaron un método para la negociación de claves. Este método, conocido como Diffie-Hellman key exchange fue el primer método de aplicación práctica para compartir una clave secreta sobre un canal inseguro de comunicaciones sin la necesidad de disponer de un secreto compartido de antemano. Dos años después, en 1977, Rivest, Shamir y Adleman trabajando en el MIT publicaron el algoritmo RSA que fue el primer algoritmo conocido que permite realizar tanto cifrado como firmas digitales, y que supuso uno de los mayores avances en la criptografía de clave pública.

Desde los años setenta se han publicado una gran variedad de algoritmos de cifrado, firma digital y negociación de claves dentro del campo de la criptografía de clave pública: en 1984 Taher Elgamal publicó el sistema de cifrado ElGamal, basado en el algoritmo de negociación de claves de Diffie-Hellman, que permite realizar cifrado de datos con clave pública. en 1985 Neal Koblitz y Victor S. Miller publicaron, de manera independiente, varios artículos en los que sugerían el uso de curvas elípticas en criptografía. La criptografía de curva elíptica, generalmente conocida como ECC, es una propuesta para utilizar la estructura algebraica de las curvas elípticas sobre campos finitos como algoritmos de clave pública. en 1991 el NIST publicó el algoritmo DSA para su uso en su estándar de firma digital.

Los algoritmos de clave pública, como RSA o DSA, se basan en que cada usuario utiliza un par de claves relacionadas matemáticamente, en la que una de ellas descifra el cifrado que se realiza con la otra. Estos algoritmos tienen la propiedad adicional de que conociendo una de las claves del par es computacionalmente imposible deducir la otra. Una de estas claves, que debe

permanecer siempre en secreto, se conoce como privada y la otra como pública. Estos algoritmos permiten realizar dos operaciones:

Cifrado: Un mensaje cifrado con la clave pública de un destinatario no puede ser descifrado por nadie excepto por el destinatario que está en posesión de la correspondiente clave privada. Este mecanismo proporciona confidencialidad.

Firma digital: Un mensaje cifrado con la clave privada del emisor puede ser descifrado por cualquiera que tenga la clave pública de dicho emisor, probando de esa manera que sólo ese emisor pudo cifrar el mensaje y que no ha sido modificado. La firma digital proporciona autenticación.

Figura 1.2: Proceso de encriptación con clave privada y pública

1.1.2.2. Generación de claves

Para utilizar los algoritmos de clave pública, cada una de las dos entidades que desean intercambiar información deben disponer de un par de claves relacionadas matemáticamente: una privada, que sólo conoce su propietario y otra pública que debe ser conocida por cualquiera que desee comunicarse con él.

1.1.2.3. Principales algoritmos de clave pública

RSA

El algoritmo propuesto por Ron Rivest, Adi Shamir y Len Adleman en 1978, conocido como RSA,

es uno de los primeros y más versátil de los algoritmos de clave pública. Es conveniente para el cifrado/descifrado, la firma/verificación (y, por tanto, de la integridad de datos), y para el establecimiento de claves (específicamente la transferencia de llave). Puede ser utilizado como base para un generador de números pseudoaleatorios seguras, así como para la seguridad en algunos juegos electrónicos. Su seguridad se basa en la dificultad de factorizar enteros muy grandes. El estado actual de la investigación sugiere que la longitud de las claves RSA debe ser de al menos 1024 bits de longitud para proporcionar una seguridad adecuada para los próximos años o más.

DSA

DSA ha sido diseñado exclusivamente para la firma/verificación y por lo tanto también para la integridad de los datos. La seguridad de este algoritmo se basa en la dificultad de calcular logaritmos en un campo finito. El estado actual de la investigación con respecto a los logaritmos discretos sugiere que las claves DSA debe ser de al menos 1024 bits de longitud para proporcionar una seguridad adecuada para los próximos años o más.

DH

El algoritmo que Diffie y Hellman propusieron, conocido como DH, es un maravilloso ejemplo de elegancia y simplicidad. El algoritmo se deriva su seguridad de la dificultad de calcular logaritmos en un campo finito y permite el intercambio secreto de claves entre dos partes que no han tenido contacto previo, utilizando un canal inseguro, y de manera anónima (no autenticada).. Al igual que con los anteriores, el estado actual de la investigación con respecto a los logaritmos discretos sugiere que las claves DH debe ser de al menos 1024 bits de longitud para proporcionar una seguridad adecuada para los próximos años o más.

ELGamal

El procedimiento de cifrado/descifrado ElGamal se refiere a un esquema de cifrado basado en problemas matemáticos de algoritmos discretos. Es un algoritmo de criptografía asimétrica basado en la idea de Diffie-Hellman y que funciona de una forma parecida a este algoritmo discreto. El algoritmo de ElGamal puede ser utilizado tanto para generar firmas digitales como para cifrar o descifrar. Fue descrito por Taher Elgamal en 1984 y se usa en software GNU Privacy Guard, versiones recientes de PGP, y otros sistemas criptográficos. Este algoritmo no esta bajo ninguna patente lo que lo hace de uso libre. La seguridad del algoritmo se basa en la suposición que la

función utilizada es de un sólo sentido y la dificultad de calcular un logaritmo discreto [3].

Firma digital

La firma digital es un procedimiento que permite simular la seguridad que proporciona la firma manuscrita convencional. Una firma digital consiste en lo siguiente: Pedro desea firmar digitalmente un contrato con Juan; Pedro cogerá el contrato, lo cifrará con su clave privada y se lo enviará a Juan; Juan, que tiene la clave pública de Pedro será capaz de descifrar el contenido del mensaje; como sólo Pedro está en posesión de la clave privada, sólo él pudo realizar el cifrado. Con este mecanismo se obtiene autenticación del emisor.

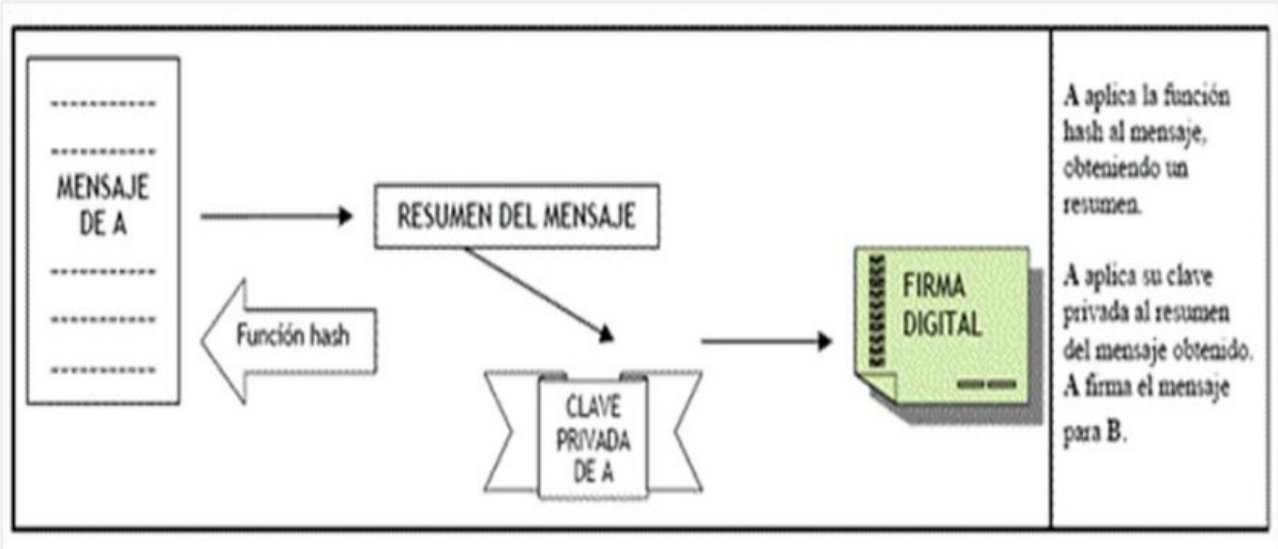

Figura 1.3: Proceso de generación de la firma digital

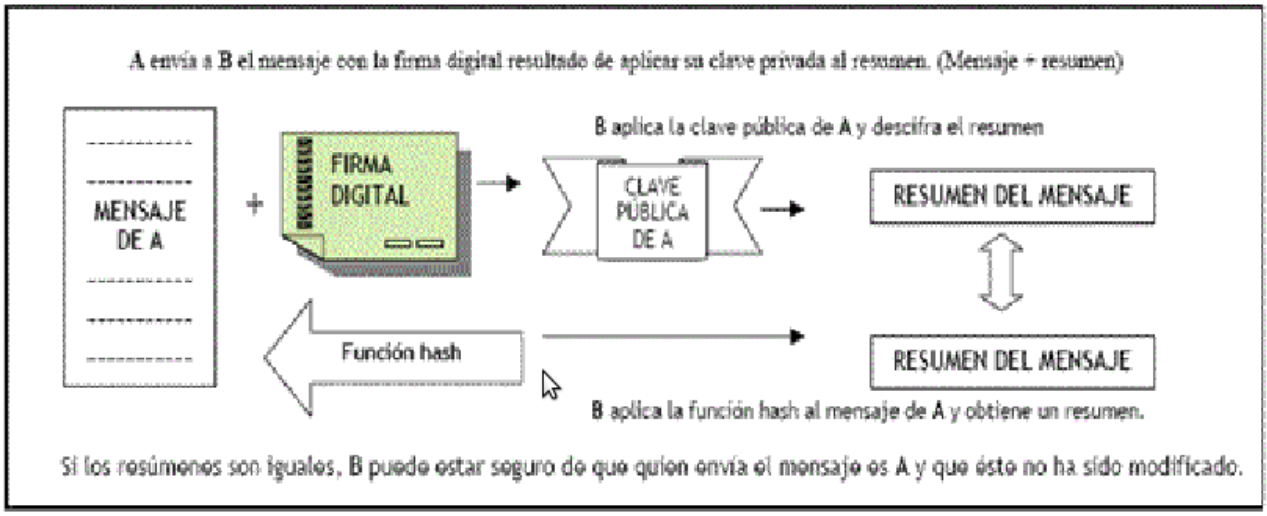

Figura 1.4: Proceso de verificación de la firma digital

Los algoritmos asimétricos son más lentos que los simétricos por lo que si el mensaje que se desea firmar es muy grande, el proceso de firma y verificación puede llevar bastante tiempo. Debido a esto, lo que se hace en lugar de cifrar todo el mensaje es aplicar una función resumen (hash) al mensaje original y cifrar el resultado. Una función “hash” es una transformación que toma como entrada una secuencia de longitud arbitraria y devuelve una secuencia de longitud fija que se denomina valor “hash” o resumen. El “hash” es un tipo de "huella digital" del documento original. Una función “hash” debe tener las siguientes propiedades: La función debe ser no invertible: dado un valor “hash” h debe ser computacionalmente imposible encontrar un valor m tal que h = hash(m). Dado un valor m1 debe ser difícil encontrar un valor distinto m2 tal que hash(m1) = hash(m2). Debe ser difícil encontrar dos valores m1 y m2 tal que hash(m1) = hash(m2).

Básicamente la firma digital constituye una herramienta para conservar la legitimidad de las obras digitalizadas, su importancia radica en lograr preservar la autenticidad e integridad de las mismas. Con el uso de la firma digital se han derivado disímiles funciones que patentizan la necesidad de su utilización, dentro de las cuales se encuentran las siguientes:

- Prueba la manifestación de la voluntad informáticamente expresada.
- Verifica que el contenido del documento no ha sido alterado.
- Atribuye el documento a su autor de manera fehaciente.

- Garantiza que el autor no puede negar haber firmado el documento o mensaje.

1.2. Certificados digitales

Son documentos legales con un grupo de datos de una persona o entidad que desea publicar su llave pública y una firma digital de quien lo acredita. La entidad que acredita los certificados es llamada agencia certificadora (AC). Uno de los modelos de certificados más usados es el X.509 que incluye datos como nombre del propietario de la llave pública, llave pública, tiempo de validez, nombre de la AC, firma de la AC, etc.) La exportación de las claves públicas por medio de certificados garantiza la existencia de una entidad que responda por los fallos de seguridad. El chequeo de veracidad de la clave viene dado por la verificación de la firma de la agencia certificadora [4]. Existen varios tipos de certificados digitales, los mas usados son:

- Certificados X.509
- Certificados SPKI (Simple Public Key Infrastructure)
- Certificados PGP (Pretty Good Privacy)

1.3. Sellado de tiempo

El sellado de tiempo o timestamping es un mecanismo on-line que permite demostrar que una serie de datos han existido y no han sido alterados desde un instante específico en el tiempo. Una autoridad de sellado de tiempo actúa como tercera parte de confianza testificando la existencia de dichos datos electrónicos en una fecha y hora concretos [4].

Pasos para llevar a cabo el sellado de tiempo:

- Un usuario quiere obtener un sello de tiempo para un documento electrónico que él posee.
- Un resumen digital (técnicamente un hash) se genera para el documento en el ordenador del usuario.
- Este resumen forma la solicitud que se envía a la autoridad de sellado de tiempo (TSA).
- La TSA genera un sello de tiempo con esta huella, la fecha y hora obtenida de una fuente fiable y la firma electrónica de la TSA.

- El sello de tiempo se envía de vuelta al usuario.
- La TSA mantiene un registro de los sellos emitidos para su futura verificación.

1.4. Estándares, protocolos y recomendaciones

1.4.1. Almacenamiento de identidades

Toda PKI debe proporcionar un método para almacenar de manera segura, tanto las claves generadas por los usuarios como las utilizadas por las autoridades de certificación para firmar los certificados que emiten. Así mismo debe ser posible almacenar los certificados digitales asociados a dichas claves. Existen diferentes soluciones para el almacenamiento de claves, tanto software como hardware, los formatos más utilizados han sido elaborados y publicados, dentro del grupo de estándares PKCS, por la empresa RSA Security.

1.4.1.1. Almacenes de claves y certificados

Un keystore(almacén de claves) es una base de datos (representada por un fichero) que contiene información acerca de los certificados y las claves privadas de los usuarios y es accedida a través de un API genérica que proporciona la máquina virtual de java [5]. Existen tres tipos fundamentales de keystores:

- "JKS": Es el tipo por defecto. Este algoritmo es capaz de leer y almacenar certificados tanto de usuarios como de autoridades de certificación. No permite almacenar claves simétricas. Además las claves privadas almacenadas no se protegen con un algoritmo PBE, sino que la clave privada se almacena en texto plano, y lo único que se guarda es un “hash” de la contraseña que usa el algoritmo a la hora de recuperar la contraseña. Esto se hizo así por las restricciones de exportación de los EEUU. Los algoritmos que se utilizan son propietarios y el proveedor es Sun.
- "JCEKS". Este tipo sí permite almacenar claves simétricas encriptadas con PBE, y las claves privadas también las encripta con PBE. Este tipo de keystore también es privado y su proveedor es SunJCE.
- "PKCS12". No es un keystore totalmente funcional, sino que sólo permite leer los documentos PKCS #12, aunque no permite escribirlos. El proveedor de servicios criptográficos Bouncy Castle que implementa JCA y JCE proporciona las funcionalidades necesarias para operar completamente con este tipo de keystore.

1.4.1.2. PKCS #12: Estándar de sintaxis de intercambio de información personal

El estándar PKCS #12 describe una sintaxis para intercambiar información personal, incluyendo

claves privadas, certificados y secretos de diversos tipos. Los objetos PKCS #12 permiten mover información personal de manera segura y con garantías de integridad entre diferentes sistemas. El estándar define diferentes modelos de uso a la hora de mantener la seguridad y la integridad. Para mantener la privacidad existen dos modelos de uso:

- Modelo de privacidad basado en clave pública: En este modelo la información se cifra en la plataforma origen utilizando una clave pública confiable. El envoltorio resultado se podrá abrir en la plataforma destino con la clave privada correspondiente.
- Modelo de privacidad basado en una contraseña: La información se cifra con una clave simétrica derivada de un nombre de usuario y la contraseña proporcionada.
- Para mantener la integridad de los datos también se dispone de dos modelos:
- Modelo de integridad basado en clave pública: En este modo de funcionamiento la integridad se garantiza mediante una firma digital de los datos, que se genera con la clave privada de la plataforma de origen. En la plataforma de destino se utiliza la correspondiente clave pública para verificar la firma.
- Modelo de integridad basado en contraseñas: La integridad se garantiza mediante un código de autenticación de mensajes (MAC 5) derivado de la contraseña secreta de integridad. Cuando se usa en conjunto con el modelo de privacidad basado en contraseñas, ambas contraseñas no tienen porque ser iguales [6].

1.4.2. Estándares en entornos hospitalarios

1.4.2.1. HL7

HL7 (Health Level Seven) es un conjunto de estándares para el intercambio electrónico de información médica.

Desde su origen en 1987, el nombre de HL7 se asociaba a las versiones del estándar de mensajería para el intercambio electrónico de datos de salud.

Este estándar está enfocado al intercambio de datos entre aplicaciones (facilitando el desarrollo de interfaces). Sin embargo, la creciente necesidad de generar sistemas de información integrados regionalmente (ciudades, regiones, países) hizo necesario el desarrollo de un espectro más amplio de estándares que faciliten la interoperatividad.

Por esta razón, en la actualidad (a partir del año 2000), la organización HL7 cuenta con un proceso para definir una serie de herramientas de interoperatividad (mensajes, documentos electrónicos, reglas, modelos de referencia), esto ha dado origen a varios estándares que facilitan los procesos de intercambio de información de salud.

Debido a ello, hoy en día, se habla de Estándares HL7. Algunos de estos estándares son:

- Mensajería HL7 Versión 2**:** Estándar de mensajería para el intercambio electrónico de datos de salud.
- Mensajería HL7 Versión 3**:** Estándar de mensajería para el intercambio electrónico de datos de salud basada en el RIM (Reference Information Model).
- CDA HL7**:** (Clinical Document Architecture) Estándar de arquitectura de documentos clínicos electrónicos.
- SPL HL7**:** (Structured Product Labeling) Estándar electrónico de etiquetado de medicamentos.
- HL7 Medical Records**:** Estándar de administración de Registros Médicos.
- GELLO**:** Estándar para la expresión de reglas de soporte de decisiones clínicas.
- Arden Sintax**:** Es estándar sintáctico para compartir reglas de conocimiento clínico.
- CCOW**:** Es un estándar framework para compartir contexto entre aplicaciones[7].

1.4.2.2. CDA

CDA (arquitectura de documento clínico) es una especificación para el intercambio de documentos clínicos. En la misma se define la sintaxis y semántica de los documentos clínicos, basándose en el RIM y utilizando XML para la codificación. Un documento CDA contiene información sobre quien lo crea, a que fecha y hora, con que propósito, a quien se lo crea y donde, además de su potencial para autenticación; en la composición del cuerpo o asunto del documento existen estructuras para el uso ordenado de nomencladores o vocabularios terminológicos. Debido a ello ha sido tomado como estructura básica para implementaciones de la HCE [7].

1.4.3. Recomendaciones del W3C para la firma de XML

1.4.3.1. Sintaxis para la firma digital XML (XML-DSig)

El estándar XMLDSig (XML Signature Syntax and Processing) recoge las reglas básicas de creación y procesamiento de firmas electrónicas de documentos XML.

Las firmas digitales XML (XMLDSIG) son firmas digitales diseñadas para su uso en transacciones XML. El estándar define un esquema para capturar el resultado de la firma digital aplicada a datos arbitrarios (a menudo en XML).

Elementos del XML:

- Elemento Signature: Encapsula la firma digital.
- Elemento SignedInfo: Contiene la información necesaria para la creación y validación de la firma.
- Elemento CanonicalizationMethod: Especifica el algoritmo de transformación canónica aplicado al código XML de SignedInfo antes de realizar el cálculo de su firma digital.

- Elemento SignatureMethod: Referencia al algoritmo utilizado para el cálculo del valor de la firma digital.
- Elemento Reference: Referencia a la información o documento que se encuentra firmado. Contiene Transforms, DigestMethod y DigestValue.
- Elemento SignatureValue: Contiene el valor de la Firma Digital.
- Elemento KeyInfo: Elemento opcional que permite a los receptores del mensaje obtener datos del certificado, como clave pública…
- Elemento Object: Elemento opcional que permite añadir información a la firma, por ejemplo, los datos que se van a firmar.

En relación con la situación de los datos que se van a firmar con respecto a su firma se tiene la siguiente clasificación:

- Enveloping Signature: La firma XML envuelve al contenido que se firma.
- Detached Signature: El objeto que es firmado está separado de la firma XML.
- Enveloped Signature: El contenido que se desea firmar engloba a la firma [8].

Las siguientes imágenes muestran de manera mas intuitiva el significado de cada clasificación:

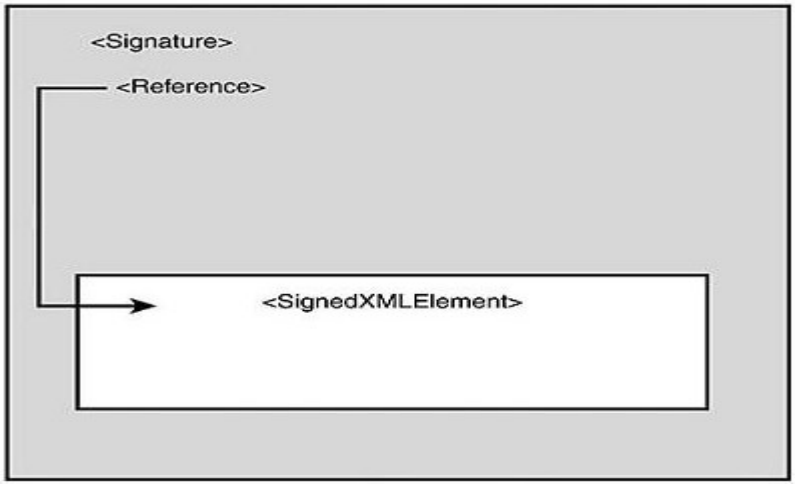

Figura 1.5: Enveloping Signature

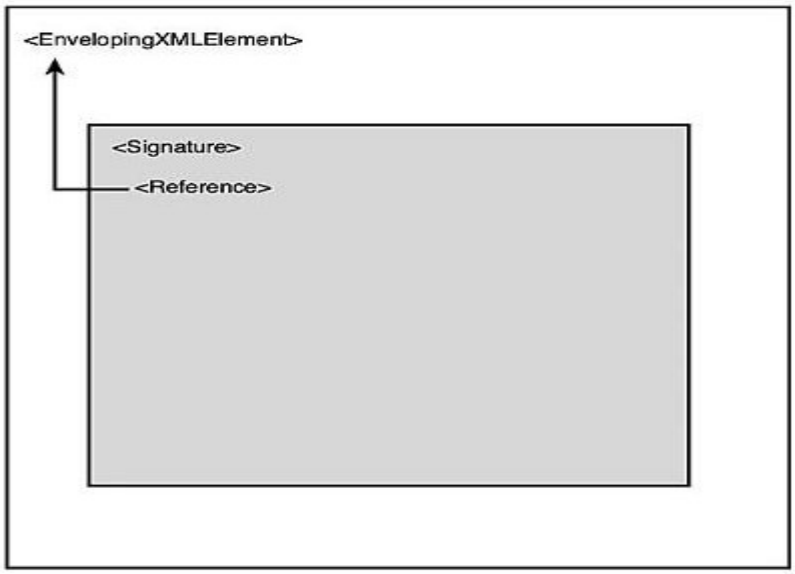

Figura 1.6: Enveloped Signature

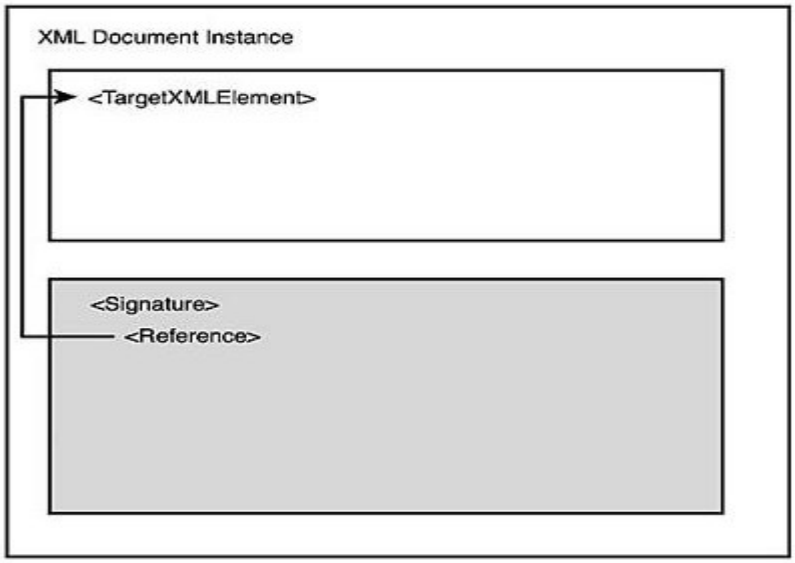

Figura 1.7: Detached Signature en el mismo documento XML

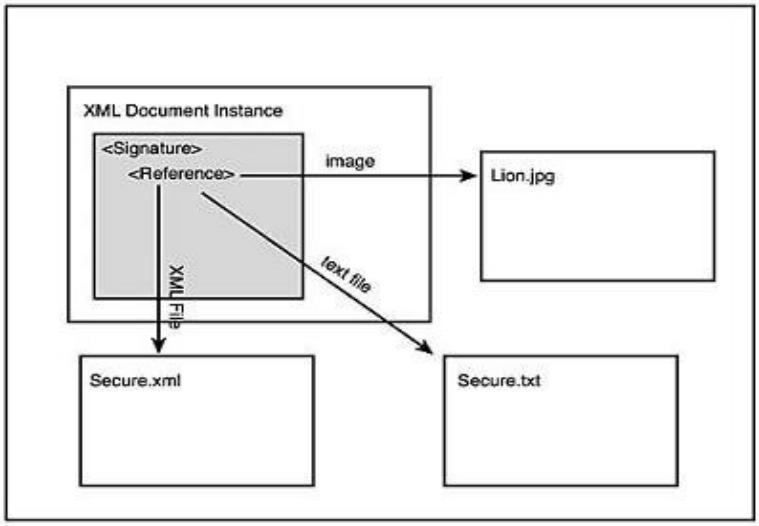

Figura 1.8: Detached Signature referenciando múltiples recursos

1.4.3.2. Firma electrónica avanzada de XML (XAdES)

XAdES sigla en inglés de **X**ML **Ad**vanced **E**lectronic **S**ignatures (Firma electrónica avanzada XML) es un conjunto de extensiones a las recomendaciones XML-DSig haciéndolas adecuadas para la firma electrónica avanzada.

Mientras que XML-DSig es un entorno general para firmar digitalmente documentos XML, XAdES especifica perfiles precisos de XML-DSig para ser usados con firma electrónica reconocida con el sentido de la directiva 1999/93/EC de la Unión Europea. Un beneficio importante de XAdES es que los documentos firmados electrónicamente pueden seguir siendo válidos durante largos períodos, incluso en el caso de que los algoritmos criptográficos subyacentes hayan sido rotos.

XAdES define seis perfiles (formas) según el nivel de protección ofrecido. Cada perfil incluye y extiende al previo:

- XAdES-BES, forma básica que simplemente cumple los requisitos legales de la Directiva para firma electrónica avanzada.
- XAdES-EPES, forma básica a la que se la ha añadido información sobre la política de firma.
- XAdES-T (timestamp), añade un campo de sellado de tiempo para proteger contra el repudio.
- XAdES-C (complete), añade referencias a datos de verificación (certificados y listas de revocación) a los documentos firmados para permitir verificación y validación off-line en el futuro (pero no almacena los datos en sí mismos).
- XAdES-X (extended), añade sellos de tiempo a las referencias introducidas por XAdES-C para evitar que pueda verse comprometida en el futuro una cadena de certificados.
- XAdES-X-L (extended long-term), añade los propios certificados y listas de revocación a los documentos firmados para permitir la verificación en el futuro incluso si las fuentes originales (de consulta de certificados o de las listas de revocación) no estuvieran ya disponibles.
- XAdES-A (archivado), añade la posibilidad de timestamping periódico (por ej. cada año) de documentos archivados para prevenir que puedan ser comprometidos debido a la debilidad de la firma durante un periodo largo de almacenamiento [9].

1.5. Metodologías de desarrollo de software

Para obtener un producto que cumpla con todas las especificaciones y esté en el tiempo estimado, se necesita una metodología que guíe el proceso de desarrollo de software. Una metodología es la que define Quién debe hacer Qué, Cuándo y Cómo. Es un proceso donde se realizan tareas para obtener un producto a través de un conjunto de actividades definidas

previamente. En la actualidad no existe una metodología universal que se le pueda aplicar a todos los proyectos y su selección depende de las características de cada proyecto.

1.5.1. RUP

El Proceso Unificado de Desarrollos (RUP por sus siglas en inglés) como lo dice su nombre es un proceso de desarrollo de software que ha unificado técnicas de desarrollo a través del Lenguaje de Unificado de Modelado (Unified Modeling Language, UML (Booch, y otros, 2005)) .

En RUP las actividades se realizan en a través de 9 flujos de trabajos y 4 fases.

Las características que hace a RUP único son tres:

- Dirigido por casos de uso: Un caso de uso es un fragmento de funcionalidad del sistema que proporciona al usuario un resultado importante. Es el reflejo de lo que los clientes necesitan y desean. Además de constituir una herramienta para especificar los requisitos de un sistema, estos guían el diseño, implementación y prueba, es decir, guían el proceso de desarrollo.
- Centrado en la arquitectura: La arquitectura de software engloba aspectos estáticos y dinámicos más significativos del sistema, esta muestra una visión común del sistema con la que el equipo de proyecto y los clientes deben estar de acuerdo. Para darle inicio al desarrollo del sistema se comienza por los casos de uso más relevantes desde el punto de vista de la arquitectura.
- Iterativo e incremental: RUP propone que el trabajo se divida en partes más pequeñas denominadas miniproyectos. Cada miniproyecto es una iteración que resulta en un incremento. Las iteraciones hacen referencia a pasos en el flujo de trabajo y los incrementos al crecimiento del producto.

RUP utiliza UML para modelar, construir y documentar los elementos que forman un sistema software orientado a objetos. Se ha convertido en el estándar de facto de la industria, debido a que ha sido impulsado por los autores de los tres métodos más usados de orientación a objetos: Grady Booch, Ivar Jacobson y Jim Rumbaugh. Esta notación ha sido ampliamente aceptada debido al prestigio de sus creadores y debido a que incorpora las principales ventajas de cada uno de los métodos particulares en los que se basa. UML ha puesto fin a las llamadas “guerras de métodos” que se han mantenido a lo largo de los 90, en las que los principales métodos sacaban nuevas versiones que incorporaban las técnicas de los demás. Con UML se fusiona la notación de estas técnicas para formar una herramienta compartida entre todos los ingenieros software que trabajan en el desarrollo orientado a objetos.

Entre sus características principales se encuentran:

- Modela estructuras complejas.

- Las estructuras más importantes que soportan tienen su fundamento en las tecnologías orientadas a objetos, tales como objetos, clase, componentes y nodos.
- Comportamiento del sistema: casos de uso, diagramas de secuencia y de colaboraciones, que sirven para evaluar el estado de las máquinas.
- Emplea operaciones abstractas como guía para variaciones futuras, y añade variables si es necesario.
- Reemplaza a decenas de notaciones empleadas con otros lenguajes.

1.6. Estado del arte

1.6.1. Software de firma digital

Sinadura: firma digital de PDF.

Las empresas especialistas en Software Libre, Irontec y Zylk han publicado y liberado el código fuente de Sinadura, una aplicación Java que permite firmar digitalmente ficheros PDF, usando por ejemplo, certificados digitales de Izenpe, FNMT o el DNIE. Es la primera aplicación de escritorio para firma de PDF bajo licencia libre que se conoce y la interfaz de la aplicación está preparada para el soporte multi-idioma.

Viafirma

La plataforma está basada en los patrones de arquitectura Service Oriented Architecture (SOA) se caracteriza por ser un producto ágil e innovador integrable con cualquier aplicación empresarial.

De forma resumida, las características técnicas de **Viafirma Platform** son:

- Desarrollo 100% Java .
- Estable y escalable. Soporta clustering y disposición en balanceo de carga.
- Multiplataforma en servidor: puede desplegarse sobre Windows, Linux, Unix.
- Multiplataforma en cliente: los usuarios pueden autenticarse y firmar en Windows, Linux, Mac Os, etc.
- Generación de recibos de firma con códigos de barra o códigos bidimensionales (QR Code) para poder leer datos de la firma con dispositivos lectores o incluso teléfonos móviles.
- Cacheo de CRL's, asegurando velocidad de transacciones y seguridad.
- VIAFIRMA Credentials Secure Store (VCSS): Módulo adicional a la plataforma que permite que los certificados digitales de los usuarios residan en un módulo seguro central existiendo un sistema que gestiona una lógica de asociación/visibilidad entre esos certificados y las personas autenticadas. Este módulo evita la dependencia a tarjetas criptográficas y lectores, problemática de pérdida de tarjetas, olvido de PIN, problemas con

los drivers, etc.

Servicios Básicos de Criptografía:

Características Criptográficas

- API Criptográfica para Java y C#
- Soporte para los algoritmos RSA, DSA, AES, Triple DES, SHA, PKCS#5, RC2, y RC4
- PKCS#11 Cryptographic Token Support

Formatos

- Tratamiento de objetos con estándar ASN.1
- Soporte para estándares X.509 V3

Public Key Infraestructure (PKI)

- Acceso a listas de certificados revocados (CRLs): X.509
- Soporte para protocolo OCSP (RFC 2560)
- Soporte para repositorios de certificados revocados sobre LDAP
- Soporte PKIX (RFC 3280)

Almacén de Certificados

- PKCS#11 (smartCards)
- PKCS#12
- Microsoft Windows Keystore (CAPI)
- Java Key Store (JKS)
- KeyChain Mac OSX

Formatos de Firma

- XMLSignature
- XadES-BES
- XadES-T
- XadES-C
- XadES-X
- XadES-XL
- PDF-Signature

Servidor de Aplicaciones

Viafirma puede desplegarse en los siguientes servidores de aplicaciones:

- Apache Tomcat
- Websphere

- WebLogic

Otras Funcionalidades

- TimeStamping (RFC 3161)

Cliente para Java

Viafirma habilita a las aplicaciones terceras un sencillo API que posibilita la inyección en las mismas de las funcionalidades de autenticación y firma digital con certificados. Para ello, ofrece todos sus servicios mediante métodos estándar (Servicios Web y OpenID), además de disponer de API y kits de desarrollo multiplataforma. Para las aplicaciones desarrolladas en java dispone de:

- VIAFIRMA Java client (.jar)
- Dependencias Maven
- Aplicación Cliente de Ejemplo
- Código Fuente de Ejemplo
- Kit de desarrollo
- Manual de Integrador

Viafirma es una plataforma muy completa pero presenta dificultades que no premiten utilizarlo en el desarrollo:

- No se puede integrar con el servidor de aplicaciones Jboss Application Server.
- La arquitectura del cliente no permite su utilización dentro del sistema alas HIS.
- Es un software con licencia comercial [10].

EParapher

Para firma digital de archivos se puede utilizar EParapher, una herramienta que soporta casi todo tipo de archivos. Esta herramienta firma y convierte el archivo a tres posible estándares: PDF, PDF/A y XML.

Entre sus características destaca:

- Conversión y firma rápida de cualquier archivo de texto, imagen y archivos de Office u Open Office.
- Permite crear, suprimir, importar y exportar certificados y llaves privadas.
- Soporta los archivos de almacenamiento de llaves: PKCS#12, JKS, JCEKS y BKS.
- Permite utilizar certificados de Windows y Smart Cards.
- EParapher está disponible para las plataformas: Windows, MacOSX y Linux.
- Implementación de estándares PKI: X.509v3, CRL, OCSP, TSP, LDAP.

No es posible integragrarlo con el sistema alas HIS porque posee una interfaz de usuario basada en Eclipse RCP [11].

@firma

La Plataforma @firma es la solución tecnológica en la que se basa la implementación de la plataforma de validación y firma electrónica del Ministerio de la Presidencia de España. Es un producto robusto e integral, compuesto por diferentes productos y servicios. Es una solución basada en software libre y estándares abiertos.

El componente Cliente @firma, desarrollado por la Dirección General para el Impulso de la Administración Electrónica, perteneciente al Ministerio de la Presidencia, es un módulo que se distribuye de forma independiente a la Plataforma y que permite a los usuarios realizar el proceso de firma digital de documentos en sus máquinas locales.

El Cliente @Firma es una herramienta de firma electrónica que se ejecuta en cliente (PC del usuario) basada en java. Esto es así para evitar que la clave privada asociada a un certificado digital tenga que “salir” del contenedor del usuario (tarjeta, dispositivo USB o navegador) ubicado en su PC. Funciona bajo dos modalidades distintas:

- como applet de Java integrado en páginas web mediante comandos JavaScript
- como aplicación de escritorio

Este cliente de firma electrónica contiene las interfaces y componentes web necesarios para la realización de los siguientes procesos:

- Firma de formularios web.
- Firma de datos y ficheros.
- Multifirma masiva de datos y ficheros.
- Cofirma (CoSignature): Multifirma al mismo nivel.
- Contrafirma (CounterSignature): Multifirma en cascada.

Como complemento al Cliente @Firma, se encuentra un cliente de cifrado que permite realizar funciones de cifrado y descifrado de datos atendiendo a diferentes algoritmos y configuraciones.

El cliente es distribuido bajo licencia GPL pero la plataforma es de uso privado(solo en España).

1.6.2. Trabajos relacionados con la firma digital

Firma Digital de Documentos utilizando Smart Cards

Aplicación multiplataforma, desarrollada en la Universidad de las Ciencias Informáticas, para la firma digital de documentos en formato PDF utilizando tarjetas inteligentes, que garantiza la autenticidad, integridad y no repudio de la información almacenada en dichos documentos. Permite el acceso a tarjetas inteligentes para obtener los certificados y las claves privadas.

Incorpora a la firma el sello de tiempo. Está concebido para su uso como una aplicación de escritorio [12].

Componente de software para la firma digital de documentos jurídicos tratados en formato electrónico en el proyecto Tribunales

Componente web para la firma digital, desarrollado en la Universidad de las Ciencias Informáticas, de documentos en formato PDF, que garantiza la autenticidad, integridad y no repudio de la información. Utiliza PHP como lenguaje de desarrollo [13].

Componente de firma digital para el proyecto Sistema de Gestión Fiscal

Componente de firma digital, desarrollado en la Universidad de las Ciencias Informáticas, que sirve para concederle integridad y autenticidad a los documentos que se generan en la Fiscalía General de la República de Cuba. Utiliza PHP como lenguaje de desarrollo [14].

1.7. Lenguajes, librerías de clases y tecnologías de desarrollo

1.7.1. Java

Java es un lenguaje de programación orientado a objetos desarrollado por Sun Microsystems a principios de los años 90. El lenguaje en sí mismo toma mucha de su sintaxis de C y C++, pero tiene un modelo de objetos más simple y elimina herramientas de bajo nivel, que suelen inducir a muchos errores, como la manipulación directa de punteros o memoria.

Las aplicaciones Java están típicamente compiladas en un bytecode, aunque la compilación en código máquina nativo también es posible. En el tiempo de ejecución, el bytecode es normalmente interpretado o compilado a código nativo para la ejecución, aunque la ejecución directa por hardware del bytecode por un procesador Java también es posible.

La implementación original y de referencia del compilador, la máquina virtual y las bibliotecas de clases de Java fueron desarrollados por Sun Microsystems en 1995. Desde entonces, Sun ha controlado las especificaciones, el desarrollo y evolución del lenguaje a través del Java Community Process, si bien otros han desarrollado también implementaciones alternativas de estas tecnologías de Sun, algunas incluso bajo licencias de software libre.

Entre noviembre de 2006 y mayo de 2007, Sun Microsystems liberó la mayor parte de sus tecnologías Java bajo la licencia GNU GPL, de acuerdo con las especificaciones del Java Community Process, de tal forma que prácticamente todo el Java de Sun es ahora software libre (aunque la biblioteca de clases de Sun que se requiere para ejecutar los programas Java aún no lo es).

1.7.2. Java SE y Java EE

Java Platform Standard Edition o Java SE (conocido anteriormente hasta la versión 5.0 como Plataforma Java 2, Standard Edition o J2SE), es una colección de API del lenguaje de programación java útiles para muchos programas de la Plataforma Java. La Plataforma Java Enterprise Edition incluye todas las clases en el Java SE, además de algunas de las cuales son útiles para programas que se ejecutan en servidores sobre “workstations”.

Comenzando con la versión J2SE 1.4 (Merlin), la plataforma Java SE ha sido desarrollada bajo la supervisión del Java Community Processes. JSR 59 la especificación para J2SE 1.4 y JSR 176 especificó J2SE 5.0 (Tiger). En 2006, Java SE 6 (Mustang) está siendo desarrollada bajo el JSR 270. Dentro de las numerosas ventajas que proporciona el uso de esta tecnología se pueden citar algunas:

- Independiente de la plataforma : Con la información de una empresa desplegada en formatos dispares, en distintas plataformas y aplicaciones, es importante adoptar un lenguaje de programación que pueda funcionar perfectamente en la empresa sin tener que recurrir a mecanismos ineficientes de traducción. Un modelo de programación unificado también reduce las dificultades surgidas de la integración de muchas de las diversas tecnologías que se convierten en específicas para ciertas plataformas y aplicaciones.
- Objetos gestionados : Java EE proporciona un entorno gestionado para componentes y las aplicaciones de Java EE son céntricas respecto del contenedor. Ambas nociones son críticas para construir aplicaciones de cliente. Al ser gestionados, los componentes de Java EE utilizan la infraestructura proporcionada por los servidores de Java EE sin que el programador sea consciente de ello. Las aplicaciones de Java EE son también declarativas, un mecanismo con el que puede modificar y controlar el funcionamiento de las aplicaciones sin cambiar de código. A primera vista, estas características pueden parecer incómodas de ejecutar. Sin embargo, cuando considera las aplicaciones a gran escala con varios cientos de componentes interactuando para ejecutar complejos procesos empresariales, estas características hacen que crear y mantener tales aplicaciones resulte menos complejo. Junto con su independencia respecto de la plataforma, este es uno de los aspectos mis importantes de J2EE.
- Reusabilidad : Separar los requisitos económicos de una aplicación en sus partes integrantes es un modo de conseguir la reutilización; utilizar la orientación al objeto para encapsular la funcionalidad compartida es otro. Sin embargo, a diferencia de los objetos, los componentes distribuidos requieren una infraestructura más compleja para su construcción y gestión. Los conceptos básicos orientados a objetos no proporcionan este

tipo de marco, pero la Enterprise Edition de Java ofrece una arquitectura notablemente rigurosa para la reutilización de componentes. Los componentes de empresa (llamados Enterprise JavaBeans) desarrollados en Java EE son de básica granulometría y reproducibles. Manteniendo estos componentes básicos, es posible crear funcionalidades de empresa complejas de un modo libremente coordinado. Además, ya que los componentes son reproducibles, lo cual quiere decir que es posible identificar ciertos metadatos sobre los componentes, las aplicaciones pueden ser creadas componiendo tales componentes. Ambas características fomentan la reutilización del código a alta granulometría.

- Modularidad : A la hora de desarrollar una aplicación de servidor completa, los programas pueden ampliarse y complicarse rápidamente. Siempre es mejor romper una aplicación en módulos discretos, cada uno de ellos responsable de una tarea específica. Esto hace que nuestras aplicaciones sean mucho más fáciles de mantener y comprender. Por ejemplo, los servlets de Java, las Páginas de JavaServer y Enterprise JavaBeans proporcionan un modo de modularizar nuestra aplicación, rompiendo nuestras aplicaciones en diferentes niveles y tareas individuales.

1.7.3. Bouncy Castle

Bouncy Castle es un proveedor de servicios criptográficos que implementa las API JCA y JCE de JAVA. Es uno de los más utilizados por la comunidad de criptografía para el desarrollo de aplicaciones con elementos de seguridad digital. Se puede obtener de manera gratuita, su código es abierto y sus API criptográficas para Java consisten en:

- Un proveedor criptográfico para JCA y JCE.
- Una implementación clara de JCE 1.2.1.
- Librerías para el manejo de objetos con codificación ASN.1.
- Generadores para certificados X.509 versiones 1 y 3, CRL versión 2 y archivos PKCS12.
- Generadores de atributos de certificados X.509 versión 2.
- Generadores /Procesadores para OCSP .
- Generadores /Procesadores para TSP [15].

1.7.4. Apache Commons

Apache Commons, comúnmente referido como Commons, es un conjunto de librerías desarrolladas en java que van desde lo mas simple hasta lo mas complejo. Es un proyecto de la Apache Software Fundation y se distribuye bajo la licencia Apache Software License, Version 2.0.

Las librerías que conforman dicho proyecto, y que se pueden incluir en cualquier proyecto java, son las siguientes [16]:

- Apache Commons BeanUtils
- Apache Commons Betwixt
- Apache Commons CLI
- Apache Commons Codec
- Apache Commons Collections
- Apache Commons Configuration
- Apache Commons Digester
- Apache Commons HttpClient
- Apache Commons ID
- Apache Commons IO
- Apache Commons JEXL
- Apache Commons JXPath
- Apache Commons Lang
- Apache Commons Logging
- Apache Commons Math
- Apache Commons Net
- Apache Log4J
- Apache Velocity
- FreeMarker
- Apache Lucene
- Apache Slide
- Apache XML-Security

Para el desarrollo del componente se usaron las siguientes:

- commons-codec-1.2
- commons-httpclient-3.0.1
- commons-lang-2.4
- commons-logging-1.1
- log4j
- xalan-2.7.1
- xml-apis

- xmlsec-1.4.3

1.7.5. Seam Framework 2.1

Es un framework desarrollado por JBoss, una división de Red Hat. Combina a dos frameworks: Enterprise JavaBeans(EJB) y JavaServerFaces(JSF). Gracias a él se puede acceder a cualquier componente EJB desde la capa de presentación refiriéndote a él mediante su nombre de componente seam.

Seam introduce el concepto de contextos. Cada componente de Seam existe dentro de un contexto. El contexto conversacional por ejemplo captura todas las acciones del usuario hasta que éste sale del sistema o cierra el navegador, inclusive puede llevar un control de múltiples pestañas y mantiene un comportamiento consistente cuando se usa el botón de regresar del navegador. Puede automáticamente generarse una aplicación web de altas, bajas, cambio y modificaciones a partir de una base de datos existente utilizando una herramienta de línea de comandos llamada seam-gen incluida con el framework. Seam puede ser integrado con las bibliotecas de componentes JBoss RichFaces [17].

1.7.6. EJB 3

EJB 3 es una especificación que simplifica el desarrollo sobre Java 5. Mediante EJB 3 se especifica la lógica del lado del servidor en las aplicaciones empresariales sobre Java con una notable simplificación del modelo de programación pues se reemplazan los descriptores de despliegue XML por anotaciones, se elimina el uso innecesario de las interfaces y se incluye la inyección de dependencias, mecanismo mediante el cual el contenedor web provee de las componentes a los controladores que lo necesiten [18].

1.7.7. JavaServer Faces

Es un framework para aplicaciones Java basadas en web que simplifica el desarrollo interfaces de usuario en aplicaciones Java EE . JSF incluye:

- Un conjunto de API para representar componentes de una interfaz de usuario y administrar su estado, manejar eventos, validar entrada, definir un esquema de navegación de las páginas y dar soporte para internacionalización y accesibilidad.
- Dos librerías de etiquetas personalizadas para Java Server Pages que permiten expresar una interfaz Java Server Faces dentro de una página JSP.
- Un modelo de eventos en el lado del servidor.
- Beans administrados.

1.7.8. Controles Seam JSF

La plataforma Seam contiene varios controles JSF útiles que complementan las componentes nativas de JSF y que son compatibles con otras implementaciones de componentes JSF como

RichFaces, ICEfaces y Apache MyFaces Trinidad. Incluye controles para la navegación, conversión de valores, propagación de contextos, validación, formateo de textos, entrada de valores y subida de archivos [19].

1.7.9. JPA

JPA es la especificación para la persistencia en Java en el cual se pauta cómo realizar las conversiones o mapeos entre el modelo orientado a objetos y el modelo relacional. JPA implementa las especificaciones EJB 3 y muchas implementaciones toman ventaja de ello para aportar un valor agregado como la generación de código del modelo de objetos hacia la base de datos o desde la base de datos hacia el modelo de objetos.

1.8. Herramientas de Desarrollo

1.8.1. Eclipse 3.5 Galileo

Eclipse es un entorno de desarrollo integrado de código abierto multiplataforma para desarrollar lo que el proyecto llama "Aplicaciones de Cliente Enriquecido", opuesto a las aplicaciones "Cliente-liviano" basadas en navegadores. Esta plataforma, típicamente ha sido usada para desarrollar entornos de desarrollo integrados (del inglés IDE), como el IDE de Java llamado Java Development Toolkit (JDT) y el compilador (ECJ) que se entrega como parte de Eclipse (y que son usados también para desarrollar el mismo Eclipse). Sin embargo, también se puede usar para otros tipos de aplicaciones cliente, como BitTorrent Azureus.

Eclipse es también una comunidad de usuarios, extendiendo constantemente las áreas de aplicación cubiertas. Un ejemplo es el recientemente creado Eclipse Modeling Project, cubriendo casi todas las áreas de Model Driven Engineering.

Eclipse fue desarrollado originalmente por IBM como el sucesor de su familia de herramientas para VisualAge. Eclipse es ahora 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.

1.8.2. Servidor de aplicaciones Jboss 4.2

Es el servidor de aplicaciones de código abierto más ampliamente desarrollado del mercado. Por ser una plataforma certificada Java EE, soporta todas las funcionalidades de J2EE 1.4, incluyendo servicios adicionales como clustering, caching y persistencia. JBoss es ideal para aplicaciones Java y aplicaciones basadas en la web. También soporta Enterprise Java Beans (EJB) 3.0, esto hace que el desarrollo de las aplicaciones empresariales sean mucho más simples.

1.8.3. PostgreSQL Server 8.4

PostgreSQL es un sistema de gestión de bases de datos objeto relacional. Considerado estable y robusto PostgreSQL encabeza a los sistemas de gestión de bases de datos objeto relacional desarrollados bajo licencia de código abierto. El mismo implementa a una gran parte del lenguaje de consultas SQL y permite realizar consultas complejas, soporta el uso de llaves foráneas, eventos disparadores, vistas e integridad transaccional. También puede ser extendido añadiendo nuevos tipos de datos, funciones, operadores y lenguajes procedimentales.

1.8.4. Ant 1.8.2

Es una herramienta usada en programación para la realización de tareas mecánicas y repetitivas, normalmente durante la fase de compilación y construcción (build). Es, por tanto, un software para procesos de automatización de compilación desarrollado en lenguaje Java y requiere la plataforma Java, así que es más apropiado para la construcción de proyectos Java.

Esta herramienta, hecha en el lenguaje de programación Java, tiene la ventaja de no depender de las órdenes del shell de cada sistema operativo, sino que se basa en archivos de configuración XML y clases Java para la realización de las distintas tareas, siendo idónea como solución multi-plataforma. Actualmente es un proyecto de la Apache Software Foundation. Es software open source, y se lanza bajo la licencia Apache Software.

Conclusiones

En este capítulo se ha realizado un estudio de los principales sistemas existentes en el mundo de la firma digital. Además fue realizada una valoración de las principales tecnologías, metodologías y herramientas que son empleadas en el desarrollo de la aplicación. Se explicaron también los principales conceptos a tratar durante el desarrollo.

Los sistemas estudiados no satisfacen las necesidades, por lo que se decide diseñar e implementar una infraestructura que resuelva los problemas presentes en estos.

CAPÍTULO 2: CARACTERÍSTICAS DEL SISTEMA

Introducción

En este capítulo se esboza la propuesta de solución para el desarrollo de una infraestructura de software para firma y validación digital. Para ello se comienza con la realización del modelo de dominio correspondiente, en el cual se definen los principales conceptos que se emplearán. Luego se puntualizan los requisitos no funcionales que debe cumplir el sistema, además de describir la arquitectura a utilizar y presentar el diagrama de despliegue del sistema. Se analizan también aspectos como la seguridad y los estándares de codificación.

2.1. Modelo de Dominio

El Modelo de Dominio o Modelo Conceptual es una representación visual de los conceptos u objetos que se manejan en el dominio del sistema. Los objetos o conceptos incluidos en el modelo de dominio no describen clases u objetos del software; sino entidades o conceptos del mundo real que están asociadas al problema en cuestión. Dicho modelo podrá ser utilizado como una base de las abstracciones relevantes en el proceso de construcción del sistema.

En el Modelo de Dominio no se muestra comportamiento. Las clases conceptuales pueden tener atributos pero no métodos. Para cualquier solución de casos de uso que se haya elegido, los conceptos e ideas propias del dominio del problema son las mismas; un mismo modelo de dominio contempla cualquiera de las soluciones analizadas. El modelo de dominio es global, es decir se realiza para todos los casos de uso y no para uno en particular.

2.2. Conceptos fundamentales del dominio

Con la finalidad de una mejor comprensión del Diagrama del Modelo de Dominio a continuación se dará una breve descripción de los conceptos encontrados en el ámbito del problema. Estos son:

- **CDA**: Documento clínico electrónico del paciente.
- **Firmar CDA**: Es la acción de firmar un documento clínico electrónico.
- **CDA firmado**: Fichero que contiene el CDA con su firma digital.
- **Validar CDA**: Proceso mediante el cual se verifica y se confirma la identidad del firmante y la veracidad del documento clínico electrónico.
- **Usuario**: Persona que realiza la firma del documento clínico electrónico.

- **Administrador**: Persona encargada de la configuración de los parámetros de firma y validación digital de los documentos clínicos electrónicos.
- **Identidad digital**: Representa al firmante del documento de manera unívoca, está formada por pares de claves públicas y privadas.
- **Certificado de confianza**: Son aquellos certificados que la entidad hospitalaria y sus usuarios consideran válidos.
- **Servidor**: Representa a los servidores que prestan servicios en una PKI.
- **Entidad hospitalaria**: Hospital donde se encuentra instalado el sistema.

2.3. Diagrama del modelo de dominio

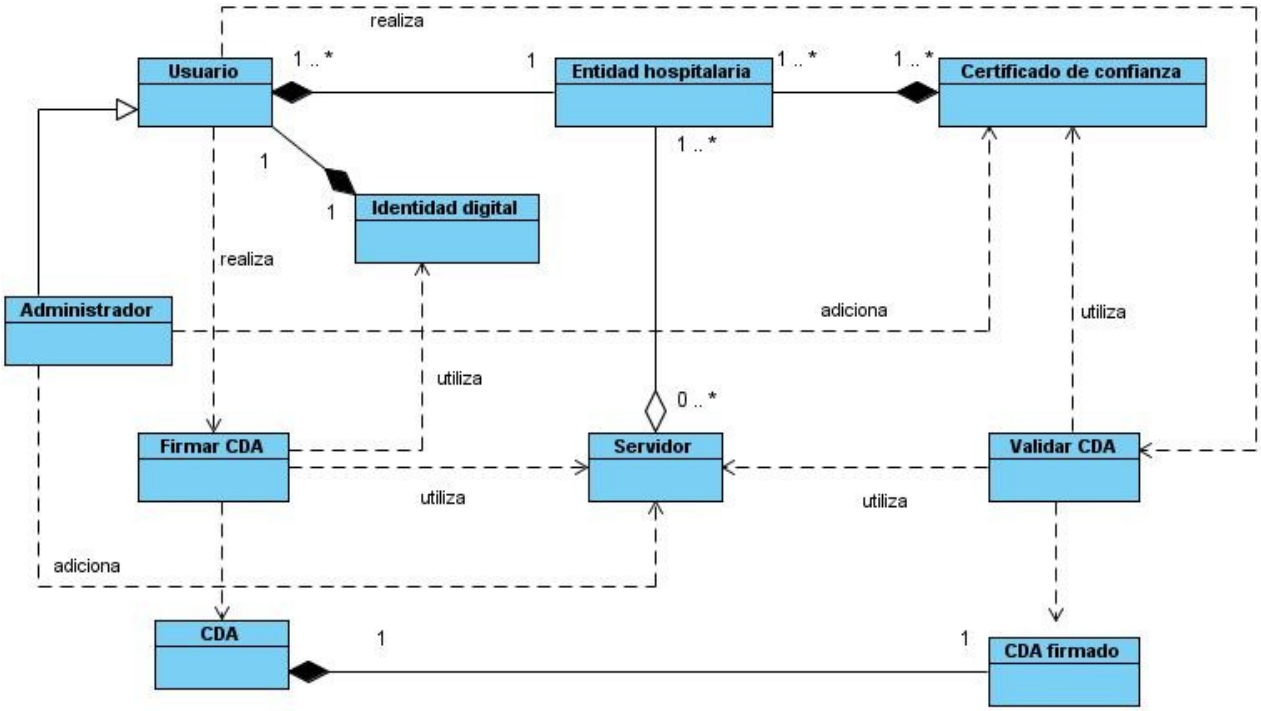

Figura 2.1: Diagrama del modelo de dominio

2.4. Especificación de los requisitos del software

Un requisito de software podría definirse como una condición o capacidad que debe encontrarse o estar en un sistema o componente para satisfacer un contrato, norma, especificación u otro documento impuesto formalmente. El conjunto de todas las necesidades es el fundamento para el consiguiente desarrollo del sistema o componente.

2.4.1. Requisitos no funcionales del sistema

Los requisitos no funcionales son cualidades o propiedades con las que debe de cumplir la solución. Son las características que logran que la solución sea más atractiva, usable, rápida y confiable.

Requisitos de rendimiento

RNF1- Permitir la realización de varias acciones por parte del sistema, garantizando que la rapidez de realización de dichas acciones sea máxima.

Requisitos de soporte

RNF2- El sistema debe contar con un mantenimiento y una revisión periódica, garantizando la disponibilidad de los servicios que brinda el sistema.

Requisitos de portabilidad

RNF3- El sistema podrá ser utilizado bajo cualquier arquitectura o sistema operativo donde exista instalada la máquina virtual de java (JVM).

Requerimientos de Software

RNF4- Debe contar con Sistema Operativo Windows, Linux o UNIX. Además debe contar con la máquina virtual de java (JVM) 1.6 y como gestor de base de datos PostgreSQL 8.3 o superior.

Requerimientos de Hardware

RNF5- Procesador Intel Quad Core o superior, 2 GB de memoria RAM, disco duro con capacidad de 40 GB.

2.5. Descripción de la arquitectura

La Arquitectura de Software es un diseño de alto nivel de la estructura de un sistema. En la misma se describen los patrones y diagramas más relevantes estableciendo un modelo o línea base de

desarrollo. De esta forma se crea un sistema de abstracciones coherentes que proporciona el marco de referencia necesario y erige las pautas a seguir en la construcción del sistema.

Para el desarrollo de la infraestructura de firma y validación digital se define el uso de una arquitectura basada en componentes. Las arquitecturas basadas en componentes descomponen el software en componentes funcionales que pueden ser utilizados como una caja negra en donde se tiene de manera externa una especificación general, la cual es independiente de la especificación interna. Es el proceso de definir, implementar e integrar en sistemas componentes independientes débilmente acoplados [20]. A continuación se citan algunas ventajas que ofrece este tipo de arquitectura:

- Facilidad de instalación: Se puede reemplazar sin perjudicar la funcionalidad de otro componente.
- Costos reducidos: Se puede disminuir costos de desarrollo y mantenimiento.
- Facilidad de desarrollo: Ya que se realiza por separado no interfiere en el desarrollo de otros componentes.
- Reusable: Se puede usar para otras aplicaciones.

Enfocado a las características arquitectónicas de dicho patrón se define la siguiente estructura de componentes:

El componente **Comunicación con servidores** contiene la lógica necesaria para que la aplicación interactúe con servidores que implementen los estándares de una PKI. Los mismos pueden ser:

- Servidores Web
- Servidores de bases de datos
- Servidores de Sellado de tiempo

En el componente **Administración de las identidades y certificados digitales** se encuentran los mecanismos para acceder a los diferentes almacenes de certificados y claves privadas que puedan tener los usuarios, por lo que este componente resulta ser de gran importancia pues de el depende la protección de la identidad digital y la correcta realización de la firma.

En **Administración de los certificados de confianza** se gestionan los certificados de las entidades y usuarios que serán reconocidos como válidos para el sistema, este componente es el punto de apoyo para poder realizar la validación de una firma digital.

Por último, el componente **Firma y validación digital** encapsula toda la lógica que le permitirá a la aplicación realizar la firma del documento y la validación posterior de la misma.

2.6. Diagrama de despliegue

El diagrama de despliegue muestra las relaciones físicas entre los componentes de hardware y software en el sistema final. Es un grafo de nodos unidos por asociaciones de comunicación. Estas asociaciones indican:

- Algún tipo de ruta de comunicación entre los nodos.
- Los nodos intercambian objetos o envían mensajes a través de esta ruta

Además, el tipo de comunicación se identifica con un estereotipo que indica el protocolo de comunicación o la red. Teniendo en cuenta las características del sistema el Diagrama de Despliegue quedó como se muestra a continuación:

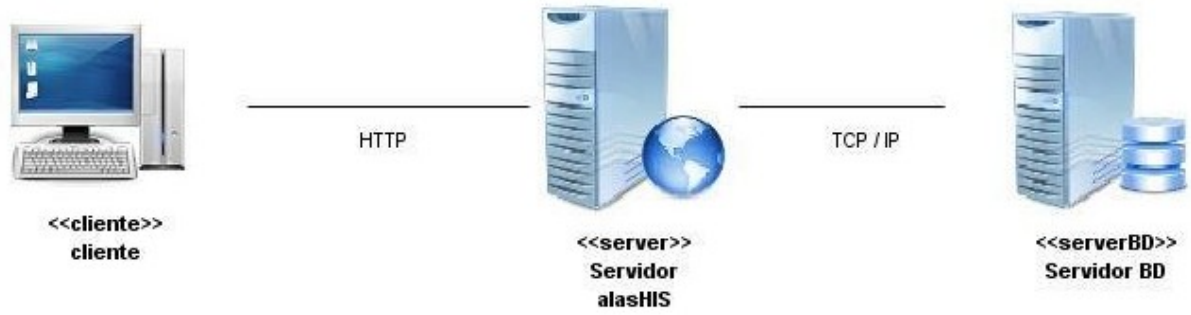

Figura 2.2: Diagrama de Despliegue

2.7. Seguridad

Para realizar la firma digital del documento clínico es necesario obtener la identidad digital del usuario firmante, debido a esto es muy importante la correcta manipulación de dicha identidad para que no sea comprometida. La clave privada, que forma parte de la identidad digital del usuario y es la usada para realizar la firma, por ninguna razón debe poder ser manipulada por el administrador del sistema ni por ninguna otra persona. Solo el sistema alas HIS se encargará de administrar dichas identidades de manera transparente, permitiendo crearlas y actualizarlas

cuando sea necesario.

Luego de que el documento clínico sea firmado se almacenará la firma junto con la información haciendo posible la detección de algún cambio en el documento original. También se confirmará la autoría y el nivel de confianza que se tiene sobre el certificado firmante del documento mediante la validación de la firma utilizando los certificados de confianza previamente configurados por el administrador del sistema.

2.8. Estrategias de codificación. Estándares y estilos a utilizar

Un estándar de codificación comprende todos los aspectos de la generación de código, de tal manera que sea prudente, práctico y entendible para todos los programadores. La utilización de las buenas prácticas de programación así como de una correcta codificación es de gran importancia para la calidad del producto, además de garantizar un buen rendimiento del mismo.

Para el desarrollo del sistema propuesto se utiliza un estándar de codificación acorde a la tecnología seleccionada garantizando la homogeneidad del estilo de programación durante la implementación del sistema. A continuación de describen algunos de los elementos que forman parte del estándar de codificación seleccionado:

En cuanto al idioma todas las palabras se deben usar en idioma español, pero sin acentuarse.

Para lograr una correcta identación, cuyo objetivo principal es el de lograr una estructura uniforme de los bloques de código, así como para los diferentes niveles de anidamiento se definieron los siguientes aspectos:

- Inicio y fin de bloque: se deben dejar dos espacios en blanco desde la instrucción anterior para el inicio y fin de bloque {}, así mismo para el caso de las instrucciones if, else, for, while, do while, switch, foreach.

- Aspectos generales: evitar el uso del tabulador; pues este puede variar según la PC o la configuración de dicha tecla. Los inicios ({) y cierre (}) de ámbito deber estar alineados debajo de la declaración a la que pertenecen y deben evitarse si hay sólo una instrucción. No se debe colocar ({) en la línea de un código cualquiera, pues esto requiere una línea propia.

Los comentarios, separadores, líneas, espacios en blanco y márgenes fueron otros de los elementos a considerar en el estándar, con estos se logra establecer un modo común para

comentar el código de forma tal que se comprenda con sólo leerlo una vez.

- Ubicación de comentarios: estos se ubican al inicio de cada clase o función y al final de cada bloque de código, especificando el objetivo de la misma así como el tipo de dato y el objetivo de cada parámetro.
- Líneas en blanco: esta se emplean antes y después de métodos, clases y estructuras.
- Espacios en blanco: se recomienda entre operadores lógicos y aritméticos para lograr una mayor legibilidad en el código.

Aspectos generales:

- No comentar cada línea de código. Cuando el comentario se aplica a un grupo de instrucciones debe estar seguido de una línea en blanco. Cuando se comente una sola instrucción se suprime la línea en blanco o se escribe a continuación de la instrucción.
- No se usa un espacio en blanco después del corchete abierto y antes del cerrado de un arreglo, después del paréntesis abierto y antes del cerrado y antes de un punto y coma.

En cuanto al uso de variables y contantes se tuvo en cuenta los siguientes elementos:

- Apariencia de variables: el nombre de las variables debe comenzar con la primera letra en minúscula, en caso de que sea un nombre compuesto se empleará la notación CamellCasing, lo cual especifica que la palabra de inicio del identificador comienza con minúscula. Si el identificador está compuesto por más de una palabra entonces estás deben comenzar con mayúscula.
- Apariencia de constantes: se declaran con todas sus letras en mayúscula.
- Aspectos generales: los nombres de las variables y constantes deben tener nombres que con sólo leerlas se conozca el propósito.

Para el uso de clases y objetos se definieron un conjunto de elementos que garantizan nombrar las clases e instancias de la misma forma para toda la aplicación:

- Apariencia de clases y objetos: los nombres de las clases deben comenzar con la primera letra en mayúscula y el resto en minúscula, en caso de que sea un nombre compuesto se

empleará notación PascalCasing que especifica que la palabra de inicio del identificador comienza con mayúscula. Si el identificador está compuesto por más de una palabra entonces estás deben comenzar con mayúscula seguido de Underscoard: Carácter ASCII representado como "_".

- Para el caso de las instancias se comenzará con un prefijo que identificará el tipo de dato, este se escribirá en minúscula.
- Apariencia de atributos: en caso de que sea un nombre compuesto se empleará notación CamellCasing.
- Apariencia de las funciones: el nombre de las funciones con verbos que denoten la acción que realiza y se empleará la notación PascalCasing. Las funcionen que obtienen algún dato emplean el prefijo get y si fijan algún valor se emplea el prefijo set
- Declaración de parámetros en funciones: se declaran agrupados por tipos, defiendo primero los string y luego los numéricos, además se agrupan tenido en cuenta los valores por defecto.

Aspectos generales: el nombre que se emplea para las clases, objetos, atributos y funciones debe permitir que con sólo leerlo se conozca el propósito de los mismos.

Conclusiones

En este capítulo se describió la arquitectura y la seguridad del sistema. También se definió la estrategia de codificación enmarcada en el estándar a utilizar, se especificaron los requerimientos no funcionales, además se elaboraron el diagrama de despliegue con el objetivo de visualizar la estructura física del sistema y el diagrama del modelo de dominio basándose en los conceptos del dominio identificados.

CAPÍTULO 3: DESCRIPCIÓN Y ANÁLISIS DE LA SOLUCIÓN

Introducción

En el presente capítulo se realiza el análisis y descripción de la solución propuesta, donde se definen los diagramas de clases y el modelo de datos del sistema. Se describen las principales clases y entidades asociadas a cada diagrama, confeccionándose además, el diagrama de componentes, el cual permite demostrar la dependencia entre los componentes del sistema.

3.1. Modelo de datos del sistema

Un modelo de datos es una serie de conceptos que puede utilizarse para describir un conjunto de datos y las operaciones para manipularlos. En un enfoque más amplio, un modelo de datos permite describir los elementos que intervienen en una realidad o en un problema dado y la forma en que se relacionan dichos elementos entre sí.

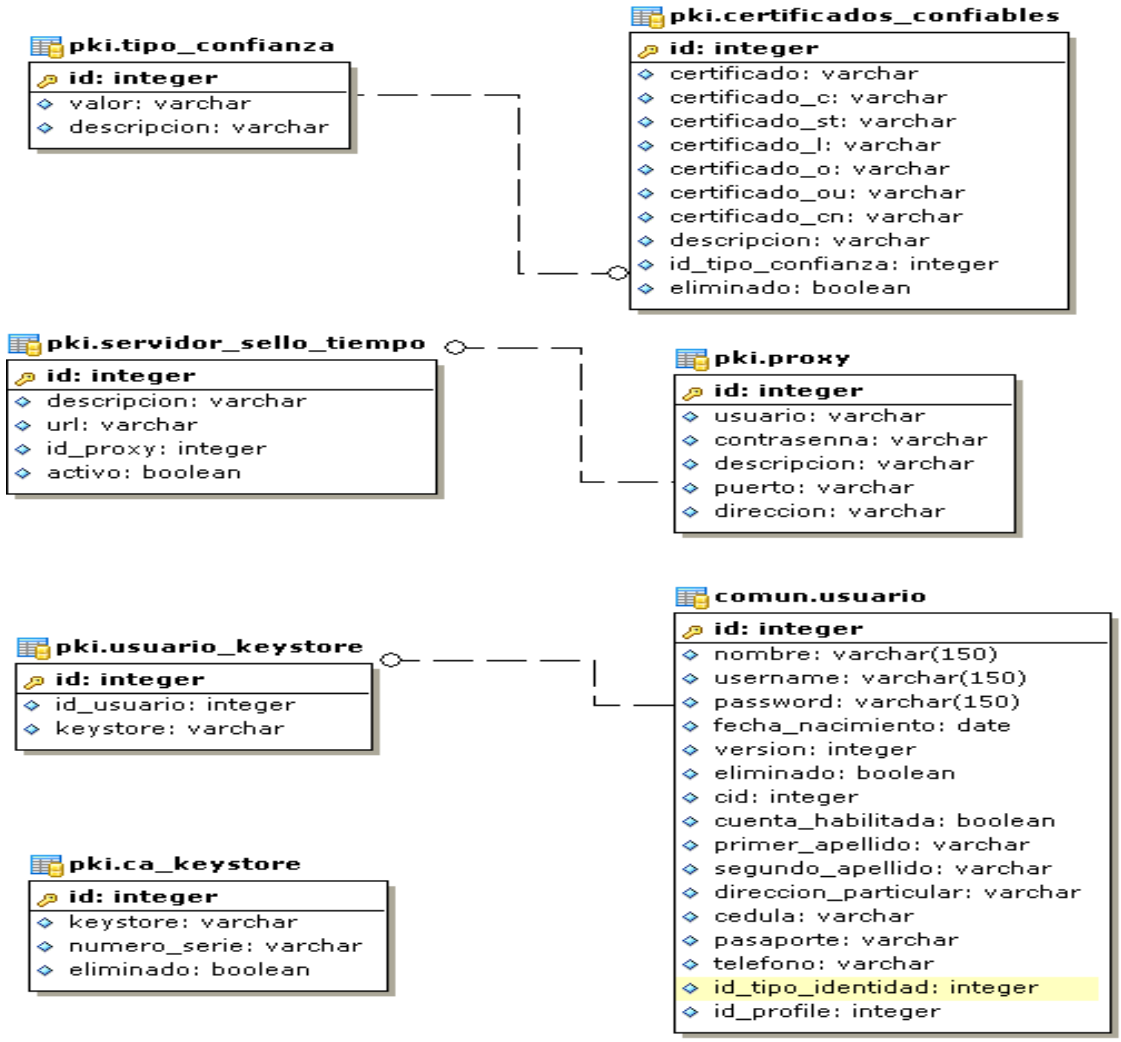

Figura 3.1: Modelo de datos

3.2. Descripción de las tablas presentes en la solución

Nombre: pki.certificados_confiables	
Descripción: Tabla para registrar los certificados declarados como confiables.	
Atributo	Tipo
id	int
certificado	varchar
certificado_c	varchar
certificado_st	varchar
certificado_l	varchar
certificado_o	varchar
certificado_ou	varchar
certificado_cn	varchar
eliminado	boolean
id_tipo_confianza	int
descripcion	varchar
Eliminado	boolean

Tabla 3.1: Descripción de la tabla certificados_confiables

Nombre: pki.usuario_keystore	
Descripción: Tabla para registrar los almacenes de claves de los usuarios.	
Atributo	Tipo
id	int
id_usuario	int
keystore	varchar
fechacreacion	date
fechavencimiento	date

Tabla 3.2: Descripción de la tabla usuario_keystore

Nombre: pki.proxy	
Descripción: Tabla para registrar los proxys para acceder a los servicios fuera de la red interna.	
Atributo	Tipo
id	int
descripcion	varchar
puerto	varchar
direccion	varchar

Tabla 3.3: Descripción de la tabla proxy

Nombre: pki.servidor_sello_tiempo	
Descripción: Tabla para registrar los servidores de sello de tiempo.	
Atributo	Tipo
id	int
descripcion	varchar
url	varchar
id_proxy	int
activo	boolean

Tabla 3.4: Descripción de la tabla servidor_sello_tiempo

Nombre: pki.tipo_confianza	
Descripción: Tabla para registrar los tipos de confianza para los certificados digitales.	
Atributo	Tipo
id	int
valor	varchar
descripcion	varchar

Tabla 3.5: Descripción de la tabla tipo_confianza

Nombre: pki.ca_keystore	
Descripción: Tabla para registrar el certificado y la clave privada de las entidades del sistema.	
Atributo	Tipo
id	int
keystore	varchar
numero_serie	varchar
eliminado	boolean

Tabla 3.6: Descripción de la tabla ca_keystore

3.3. Diagrama de paquetes

Un diagrama de paquetes muestra cómo un sistema está dividido en agrupaciones lógicas mostrando las dependencias entre esas agrupaciones. En la imagen siguiente se representa el diagrama de paquetes para el sistema.

Figura 3.2: Diagrama de paquetes

Para un mayor entendimiento estos cuatro paquetes principales están conformados por siete subpaquetes:

- GenericAPI: Sirve de base para el resto de componentes. Esta librería contiene la funcionalidad base común al resto de componentes así como las definiciones de las interfaces que permite la comunicación, extensibilidad y adaptación de los componentes entre sí.

- CertAPI: Permite el acceso a almacenes de certificados. Estos accesos serán usados para realizar labores de criptografía como firma o comprobación de confianza.

- OcspAPI: Permite la consulta del estado de los certificados mediante el protocolo OCSP (Online Certifate Status Protocol).

- TsaAPI: Contiene la lógica necesaria para obtener y manejar sellos de tiempo de autoridades de sellado a través de la implementación del apéndice A.1 de la especificación RFC 2560 para tratar las peticiones y obtener dichos sellos de tiempo vía http teniendo en cuenta las normas descritas en el punto 3.4 de la especificación RFC 3161.

- TrustAPI: Proporciona implementaciones concretas de validadores de confianza, de acuerdo con las interfaces definidas en GenericAPI.

- XadesAPI: Proporciona la funcionalidad necesaria para el manejo de ficheros con metadatos tipo XML, uso de certificados, y para la construcción de firmas electrónicas XAdES, así como para su validación. Todas las firmas generadas son de acuerdo a las especificaciones definidas por el European Telecommunications Standards Institute (ETSI).

3.4. Diagrama de clases del diseño

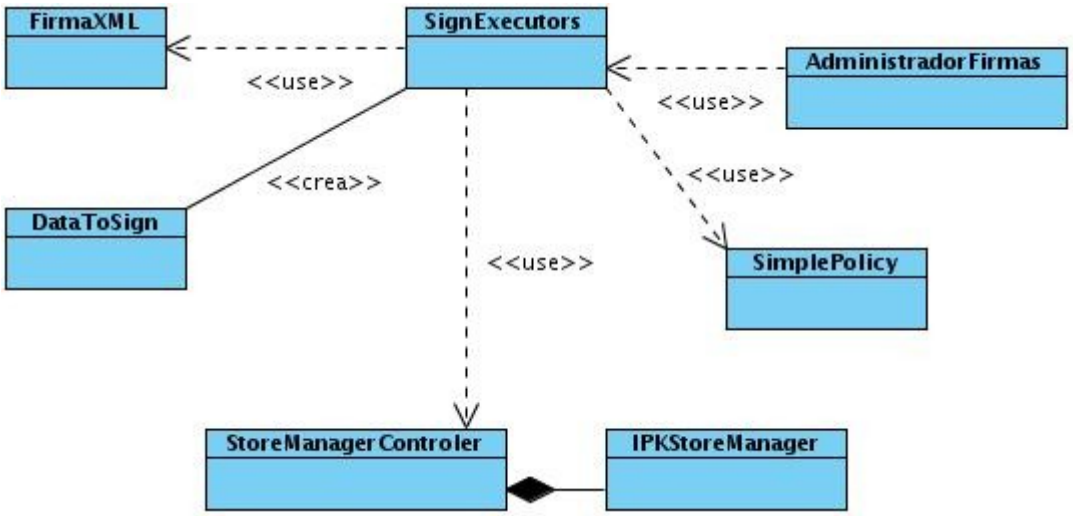

Figura 3.3: Diagrama de clases para el CU Realizar firma digital

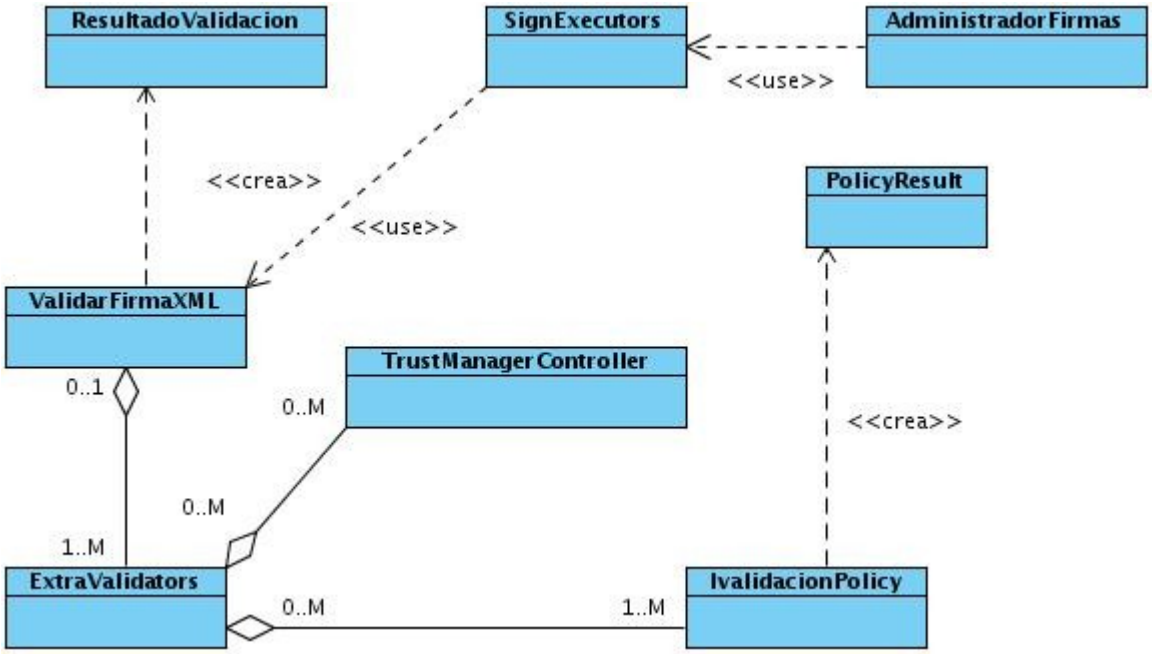

Figura 3.4: Diagrama de clases para el CU Validar firma digital

3.5. Descripción de las clases e interfaces más importantes

3.5.1. Clase DataToSign

Clase encargada de almacenar toda la información necesaria para realizar el proceso de firma digital del documento CDA.

3.5.2. Clase FirmaXML

Clase encargada de generar la firma digital del documento CDA.

3.5.3. Clase SignExecutors

Clase que implementa la interfaz ISignFacade que provee métodos para implementar la firma y validación de los documentos CDA.

3.5.4. Interfaz IPKStoreManager

Interfaz que define métodos para manipular los almacenes de claves y certificados.

3.5.5. Interfaz ITrustServices

Interfaz que define los métodos para manipular los certificados de confianza para el sistema.

3.5.6. Clase TrustManagerController

Clase que implementa las interfaces ItrustServices y TrustAdapter para proporcionar las funcionalidades necesarias para la gestión de los certificados de confianza.

3.5.7. Clase ResultadoValidacion

Clase que almacena la información referente a la validación del documento CDA.

3.5.8. Clase ValidarFirmaXML

Clase que ejecuta la validación de una firma digital y devuelve los resultados encapsulados en un objeto de tipo ResultadoValidacion.

3.5.9. Interfaz IvalidacionPolicy

Interfaz que define los métodos para la validación de la firma digital implementada a través de una política de validación específica.

3.5.10. Clase PolicyResult

Clase que encapsula el resultado de la validación de la política implementada.

3.5.11. Clase ExtraValidators

Clase que encapsula todos los posibles validadores para la firma.

3.5.12. Clase SimplePolicy

Clase que implementa la interfaz IvalidacionPolicy y obtiene un objeto de tipo PolicyResult donde se valida la política usada para la validación de los documentos.

3.5.13. Clase RaFunctions

Clase encargada de crear los certificados y las claves privadas tanto de los usuarios como de las entidades.

3.6. Patrones de diseño

Un patrón de diseño es una solución a un problema de diseño. Para que una solución sea considerada un patrón debe poseer ciertas características. Una de ellas es que debe haber comprobado su efectividad resolviendo problemas similares en ocasiones anteriores. Otra es que debe ser reusable, lo que significa que es aplicable a diferentes problemas de diseño en distintas circunstancias.

Para la construcción de la infraestructura se utilizaron los patrones:

- Abstract Factory (Fábrica abstracta): Permite trabajar con objetos de distintas familias de manera que las familias no se mezclen entre sí y haciendo transparente el tipo de familia concreta que se esté usando. Se implementó en las clases TrustFactory y TrustExtendFactory relacionadas con la gestión de certificados de confianza.

- Factory Method (Método de fabricación): Centraliza en una clase constructora la creación de objetos de un subtipo de un tipo determinado, ocultando al usuario la casuística para elegir el subtipo que crear. Se implementó en las clases SignFactory y PoliciesManager para la obtención de los servicios de firma y validación y para la gestión de políticas de validación respectivamente.

- Facade (Fachada): Provee de una interfaz unificada simple para acceder a una interfaz o grupo de interfaces de un subsistema. Se implementó a través de la inteface IsignFacade que define los métodos para la firma y validación digital.

3.7. Ficheros de configuración

Para la correcta implementación de los patrones señalados anteriormente se utilizaron ficheros especiales que permiten hacer más flexible y adaptable la solución propuesta.

3.7.1. trustservices.properties

Fichero donde se indica la factoría de servicios de confianza que se utilizará. Contiene un par <clave> = <clase>. Ejemplo:

services.trust.factory.class=org.component.javasign1.trust.TrustExtendFactory

3.7.2. trust.properties

Fichero de configuración de los validadores de confianza disponibles. Cada validador se indica mediante un prefijo de grupo y un sufijo que indica el tipo de validador. Los sufijos disponibles son:

- *.SignCerts: Validador de confianza de certificados
- *.CRLEmisor: Validador de confianza de emisores de CRLs
- *.OCSPProducer: Validador de confianza de OCSP Responder
- *.TSProducer: Validador de confianza de sellos de tiempo
- *.All: Validador de confianza que cumple con los anteriores

El valor de la propiedad indica la clase que gestiona el validador de confianza. Si no se quiere un validador en específico se debe indicar como valor de la propiedad “none”. Ejemplo:

Truster de confianza propio

alas.his.SignCerts=gehos.pki.actions.TrustManagerController

alas.his.CRLEmisor=gehos.pki.actions.TrustManagerController

alas.his.OCSPProducer=gehos.pki.actions.TrustManagerController

alas.his.TSProducer=gehos.pki.actions.TrustManagerController

alas.his.All=gehos.pki.actions.TrustManagerController

3.7.3. sign.properties

Fichero donde se indica la clase que implementa el interfaz ISignFacade que dará los servicios de firma y validación digital. Contiene un par <clave> = <clase> Ejemplo:

facade.sign.class=gehos.pki.actions.SignExecutors

Esto le indica al componente que la clase que realizará la firma y validación digital será gehos.pki.actions.SignExecutors.

3.7.4. policy.properties

En este fichero se indica qué clases son las encargadas de validar políticas específicas. Formato: <clave>=<clase> donde clave puede ser cualquier cadena de caracteres que no contenga un código especial (por ejemplo un “hash” en hexadecimal de la policy), y clase es la clase que implementa el interfaz IvalidacionPolicy. Ejemplo:

alas.his=gehos.pki.actions.SimplePolicy

3.8. Diagrama de componentes

Un diagrama de componentes muestra las organizaciones y dependencias lógicas entre componentes software, sean éstos componentes de código fuente, binarios o ejecutables. Desde el punto de vista del diagrama de componentes se tienen en consideración los requisitos relacionados con la facilidad de desarrollo, la gestión del software, la reutilización, y las restricciones impuestas por los lenguajes de programación y las herramientas utilizadas en el desarrollo. Los elementos de modelado dentro de un diagrama de componentes serán

componentes y paquetes.

A continuación se muestra el diagrama de componentes para el sistema.

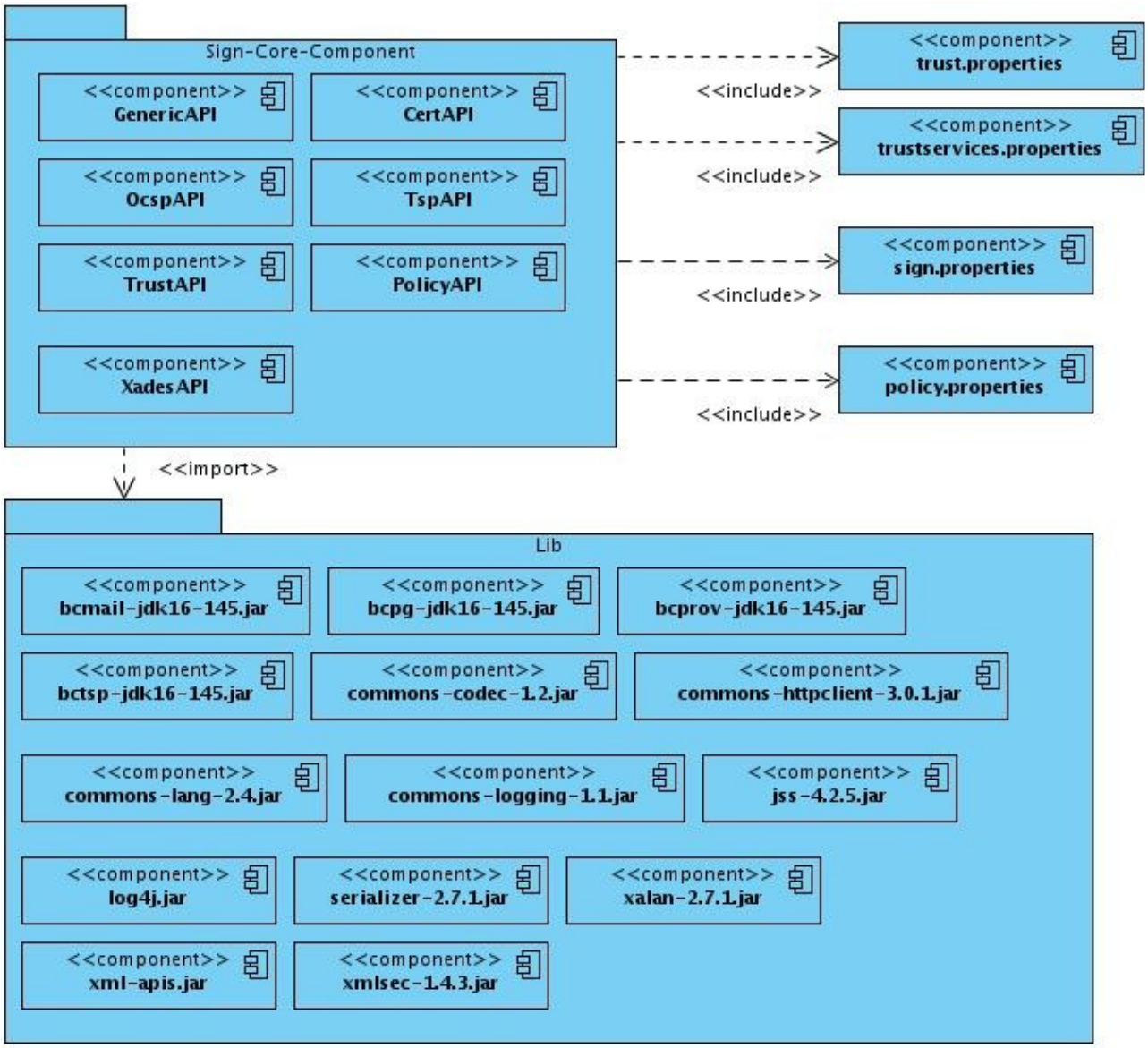

Figura 3.5: Diagrama de componentes

Conclusiones

En este capítulo se realizó el análisis y descripción de la solución propuesta. Se presentó el modelo de datos, el diagrama de paquetes y el diagrama de clases del diseño, además de brindar una breve descripción de las tablas y las principales clases utilizadas para modelar la infraestructura. Se describieron los patrones de diseño utilizados y los ficheros de configuración utilizados para la implementación de los mismos.

CAPÍTULO 4: MODELO DE PRUEBAS

Es vital para el éxito de un software que tenga un buen modelo de prueba. Para ello es necesario comprobar que el software realice correctamente las tareas indicadas en la especificación del problema. Una técnica de prueba es probar por separado cada módulo del software, y luego probarlo de forma integral, para así llegar al objetivo.

Se considera una buena práctica el que las pruebas sean efectuadas por alguien distinto al desarrollador que la programó; sin perjuicio de lo anterior el programador debe hacer sus propias pruebas. Una forma de organizar un área de pruebas, es que esté compuesta por personal inexperto y que desconozca el tema de pruebas, de esta forma, se evalúa que la documentación entregada sea de calidad, que los procesos descritos son tan claros que cualquiera puede entenderlos y el software hace las cosas tal y como están descritas. Para ello se realiza una caracterización en este capítulo de las pruebas de caja negra como método a utilizar, además de definir y describir los casos de prueba.

4.1. Prueba de caja negra

Para obtener un software excelente es necesario que se tenga un buen modelo de pruebas . Con las pruebas no se puede asegurar la ausencia de defectos en un software, pero si se puede demostrar que presenta defectos. Un buen diseño de prueba, sistemáticamente saca a la luz diferentes clases de errores, haciéndolo con la menor cantidad de tiempo y esfuerzo. Para garantizar la efectividad de las pruebas, deben ser realizadas por un equipo independiente al que realizó el sistema [21].

Las pruebas de caja negra se llevan a cabo sobre la interfaz del software y son completamente indiferentes al comportamiento interno y la estructura del programa. Con estas se pretende demostrar que las funcionalidades del sistema son operativas, que la entrada se acepta de forma adecuada, que se produce un resultado correcto y que se mantiene la integridad de la información externa. Para desarrollar la prueba de caja negra existen varias técnicas, entre ellas están:

- Técnica de la partición de equivalencia.
- Técnica del análisis de valores límites.
- Técnica de grafos de causa-efecto [22].

Técnica de partición de equivalencia

Divide el dominio de entrada de un programa en clases de datos de los que se pueden derivar casos de prueba. El diseño de estos casos de prueba para esta partición se basa en la evaluación de las clases de equivalencia para una condición de entrada. Una clase de equivalencia representa un conjunto de estados válidos o inválidos para condiciones de entrada y regularmente estas condiciones pueden ser un valor numérico, un rango de valores, un conjunto de valores relacionados o una condición lógica.

Técnica del análisis de valores límites

Los casos de prueba que exploran las condiciones límite producen mejor resultado que aquellos que no lo hacen. Las condiciones límite son aquellas que se hallan en los márgenes de la clase de equivalencia, tanto de entrada como de salida. La técnica del análisis de valores límites complementa a la de partición equivalente, pues en lugar de centrarse solamente en las condiciones de entrada, deriva los casos de prueba también para el campo de salida.

Técnica de grafos de causa-efecto

En este método se debe entender los objetos (objetos de datos, objetos de programa tales como módulos o colecciones de sentencias del lenguaje de programación) que se modelan en el software y las relaciones que conectan a estos objetos. Una vez que se ha llevado a cabo esto, el siguiente paso es definir una serie de pruebas que verifiquen que todos los objetos tienen entre ellos las relaciones esperadas. En este método:

- Se crea un grafo de objetos importantes y sus relaciones.
- Se diseña una serie de pruebas que cubran el grafo de manera que se ejerciten todos los objetos y sus relaciones para descubrir errores.

4.2. Descripción de los casos de prueba

Caso de uso: Firmar CDA

Escenarios del Firmar CDA	Descripción de la funcionalidad	Flujo central
EC 1: Firmar documento CDA	Realizar la firma de un documento CDA satisfactoriamente.	Se selecciona la opción Crear documento clínico electrónico (CDA). Se selecciona la opción Aceptar. Se firma el documento clínico electrónico creado. Muestra la interfaz Ver Detalles del documento clínico electrónico.
EC 2: El certificado de firma está desactualizado.	Al realizar la firma el sistema detecta que el certificado utilizado para firmar está desactualizado.	Se selecciona la opción Crear documento clínico electrónico (CDA). Se selecciona la opción Aceptar. Se muestra un mensaje indicando que el certificado ha caducado.
EC 3: El servidor de sello de tiempo está rechazando las conexiones.	Al realizar la firma el sistema detecta que hay un servidor de sello de tiempo que no responde.	Se selecciona la opción Crear documento clínico electrónico (CDA). Se selecciona la opción Aceptar. Se muestra un mensaje de error indicando que no se pudo obtener una respuesta desde el servidor de sello de tiempo.
EC 4: El servidor de sello de tiempo está solicitando un proxy.	Al realizar la firma el sistema detecta que hay un servidor de sello de tiempo que necesita un proxy para acceder al servicio.	Se selecciona la opción Crear documento clínico electrónico (CDA). Se selecciona la opción Aceptar. Se muestra un mensaje de error indicando que se necesita un proxy para acceder al

		servicio de sello de tiempo.

Tabla 4.1: Firmar CDA

ID Escenario	EC 1	EC 2	EC 3	EC 4
Escenario	Firmar documento CDA	El certificado de firma está desactualizado	El servidor de sello de tiempo está rechazando las conexiones	El servidor de sello de tiempo está solicitando un proxy
Variable 1 (CDA)	V	V	V	V
Variable 2 (Servidor de sello de tiempo)	V	V	I	V
Variable 3 (Proxy)	V	V	V	I
Variable 2 (Certificado)	V	I	V	V
Botón 1 (Aceptar)	NA	NA	NA	NA
Respuesta del sistema	Firma del documento CDA.	Muestra un mensaje indicando que el certificado ha caducado.	Muestra un mensaje de error indicando que no se pudo obtener una respuesta desde el servidor de sello de tiempo.	Se muestra un mensaje de error indicando que se necesita un proxy para acceder al servicio de sello de tiempo.

Tabla 4.2: Firmar CDA

Caso de uso: Validar CDA

Escenarios del Firmar CDA	Descripción de la funcionalidad	Flujo central
EC 1: Validar documento CDA	Realizar la validación de un documento CDA satisfactoriamente.	Se selecciona la opción Ver documento clínico electrónico (CDA). Se selecciona la opción Validar CDA. Se valida el documento clínico electrónico seleccionado. Muestra en la interfaz un mensaje indicando que el CDA es válido.
EC 2: El certificado de firma no está respaldado por un certificado de confianza.	Al realizar la validación el sistema detecta que el certificado utilizado para firmar no es de confianza.	Se selecciona la opción Ver documento clínico electrónico (CDA). Se selecciona la opción Validar CDA. Se valida el documento clínico electrónico seleccionado. Muestra en la interfaz un mensaje indicando que el certificado no es de confianza.
EC 3: Se ha modificado el documento clínico electrónico.	Al realizar la validación el sistema detecta que el CDA ha sido alterado.	Se selecciona la opción Ver documento clínico electrónico (CDA). Se selecciona la opción Validar CDA. Se valida el documento clínico electrónico seleccionado. Muestra en la interfaz un mensaje indicando que el CDA o el certificado de firma han sido alterados.

Tabla 4.3: Validar CDA

ID Escenario	EC 1	EC 2	EC 3
Escenario	Validar documento CDA	El certificado de firma no está respaldado por un certificado de confianza.	Se ha modificado el documento clínico electrónico.
Variable 1 (CDA)	V	V	I
Variable 2 (Certificado de firma)	V	I	V
Botón 1 (Validar)	NA	NA	NA
Respuesta del sistema	Valida el documento CDA	Muestra en la interfaz un mensaje indicando que el certificado no es de confianza.	Muestra en la interfaz un mensaje indicando que el CDA o el certificado de firma han sido alterados.

Tabla 4.4: Validar CDA

Conclusiones

En este capítulo se realizó el modelo de prueba. Atendiendo al método prueba, caja negra, obteniéndose los casos de prueba como artefactos generados del flujo de trabajo.

CONCLUSIONES GENERALES

Con este trabajo se realizó un estudio donde se identificaron un conjunto de estándares, herramientas y tecnologías que permitieron realizar el modelado de la solución. Los sistemas y trabajos analizados no cumplen con las características necesarias para poder utilizarlos en el desarrollo de la infraestructura. Se logró diseñar e implementar una infraestructura de firma y validación digital de los documentos clínicos electrónicos garantizando la integridad, autenticidad, autoría y temporalidad de los mismos, proporcionando una mayor seguridad a la información generada en las instituciones hospitalarias por el sistema alas HIS. Además se desarrolló un componente de firma y validación digital multiplataforma que puede ser integrado en cualquier aplicación de software desarrollada sobre la plataforma JAVA.

RECOMENDACIONES

El autor recomienda la implementación de la firma digital de los documentos clínicos electrónicos exportados al formato PDF a través del Visor de Historias Clínicas Electrónicas del sistema alas HIS.

REFERENCIAS

1: Sánchez Romero, Maikel. Infraestructura de software para el almacenamiento y consulta de la HC. Habana, Cuba. 2010.

2: Diccionario de la Lengua Española. [Disponible en http://buscon.rae.es/draeI/].

3: Lucena López, Manuel José. Criptografía y seguridad en computadores. 2004.

4: Adams, Carlisle;Lloyd, Steve. Understanding PKI: Concepts, Standards, and Deployment Considerations. 2002.

5: Hook, David. Beginning Cryptography with Java. 2005.

6: RSA Laboratories. Personal Information Exchange Syntax Standard. 2000.

7: Hl7 spain. [Disponible en http://www.hl7.es].

8: XML Signature Syntax and Processing. [Disponible en http://www.w3.org/TR/2008/REC-xmldsig-core-2008061].

9: XML Advanced Electronic Signature. [Disponible en http://www.w3.org/TR/XAdES/].

10: Viafirma, plataforma de autenticación y firma digital. [Disponible en http://www.viafirma.com/].

11: eParapher. [Disponible en http://maven.eparapher.com/index.html].

12: Quiñones Bondartchuk, Roberto. Firma Digital de Documentos utilizando Smart Cards. Universidad de las Ciencias Informáticas, Habana. 2009.

13: Miranda Ramos, Gianny. Componente de software para la firma digital de documentos jurídicos. La Habana, Cuba. 2010.

14: Matos Sterling, Adrian. Componente de firma digital para el proyecto Sistema de Gestión Fiscal. La Habana, Cuba. 2009.

15: Bouncy Castle. [Disponible en http://www.bouncycastle.org].

16: Apachec Commons. [Disponible en http://commons.apache.org/].

17: Juntao, Michael; Orshalick, Jacob; Heute, Thomas. Seam framework. 2009.

18: Roman, Ed; Ambler, Scott; Jewell, Tyler. Mastering enterprise JavaBeans. 2002.

19: Ka lok Tong, Kent. Begining JSF 2 API and Jboss Seam. 2009.

20: Summerville, Ian. Ingeniería de software. 2005.

21: Juristo, Natalia; Moreno, Ana; Vegas, Sira. Ténicas de evaluación de software. .

22: Pressman, R. S.. Ingeniería del software. Un enfoque práctico. 1997.

BIBLIOGRAFÍA

Adrian Matos Sterling. Componente de firma digital para el proyecto Sistema de Gestión Fiscal. La Habana, Cuba. 2009.

Ajay Vohra , Deepak Vohra . Pro XML Development with Java TM Technology . 2006.

Apache Commons. [Disponible en http://commons.apache.org/].

Bouncy Castle. [Disponible en http://www.bouncycastle.org].

Carlisle Adams, Steve Lloyd. Understanding PKI: Concepts, Standards, and Deployment Considerations. 2002.

David Hook. Beginning Cryptography with Java. 2005.

Diccionario de la Lengua Española. [Disponible en http://buscon.rae.es/draeI/].

Ed Roman, Scott Ambler y Tyler Jewell. Mastering enterprise JavaBeans. 2002.

eParapher. [Disponible en http://maven.eparapher.com/index.html].

Fernando López Hernández . Seguridad, criptografía y comercio electrónico con Java . 2007.

Gianny Miranda Ramos. Componente de software para la firma digital de documentos jurídicos. La Habana, Cuba. 2010.

HL7 Spain. [Disponible en http://www.hl7.es].

Ian Summerville. Ingeniería de software. 2005.

Jonathan B. Knudsen . Java Cryptography . 1998.

Juristo Natalia, Moreno Ana M., Vegas Sira. Ténicas de evaluación de software. .

Kent Ka lok Tong. Begining JSF 2 API and Jboss Seam. 2009.

Maikel Sánchez Romero. Infraestructura de software para el almacenamiento y consulta de la HC. Habana, Cuba. 2010.

Manuel José Lucena López. Criptografía y seguridad en computadores. 2004.

Michael Juntao, Jacob Orshalick y Thomas Heute. Seam framework. 2009.

Roberto Quiñones Bondartchuk. Firma Digital de Documentos utilizando Smart Cards. Universidad de las Ciencias Informáticas, Habana. 2009.

RSA Laboratories. Personal Information Exchange Syntax Standard. 2000.

R. S. Pressman. Ingeniería del software. Un enfoque práctico. 1997.

Sun Microsystems, Inc. JDK™ 6 Documentation. 2006.

Tim O'Brien . The Common Java Cookbook . 2011.

Viafirma, plataforma de autenticación y firma digital. [Disponible en http://www.viafirma.com/].

XML Advanced Electronic Signature. [Disponible en http://www.w3.org/TR/XAdES/].

XML Signature Syntax and Processing. [Disponible en http://www.w3.org/TR/2008/REC-xmldsig-core-2008061].

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS