PROYECTO DE FIN DE CARRERA

ESQUEMA CRIPTOGRÁFICO PARA GESTIONAR DE FORMA SEGURA HISTORIALES MÉDICOS A TRAVÉS DE UNA RED DE COMUNICACIONES.

Autor
José Manuel Vázquez Araújo
Ingeniería en Informática

Consultor
Jordi Castellà Roca

Junio 2.008

Agradecimientos

A mi tutor, Jordi Castellà Roca, por estar ahí cuando ha sido necesario, resolviendo diligentemente las dudas planteadas.

A mi mujer, Raquel, por soportar estoicamente mi mal humor los frecuentes días en los que las cosas no funcionaban como yo esperaba; y por callar los fines de semana de sol perdidos por culpa del PFC.

A Internet, sin el cual este proyecto no hubiese salido adelante.

Resumen

El historial médico de una persona es una recopilación de documentos que aportan información normalizada sobre aspectos personales y sobre las alteraciones o enfermedades que padece o ha padecido a lo largo de su vida. Según la propia OMS[1], el historial debe contener una información clara, legible, intelegible, fiable y concisa.

A día de hoy, en la mayor parte de los centros hospitalarios, el historial está constituido por una serie de documentos con soporte físico (papel, placas Rx, gráficas, etc.) que conforman una carpeta y que acompaña a cada paciente durante toda su vida, siempre que sea atendido en el centro sanitario en el que se encuentra su historial.

Este mecanismo de gestión de los historiales plantea una serie de problemas:

- Necesidad de desplazamiento físico al centro en el que se encuentra el historial para su consulta (o desplazar el historial).
- El cambio de centro hospitalario de atención de un paciente implica tener que mover el historial.
- En situaciones de urgencia, no hay inmediatez de acceso al historial de un paciente.

Las nuevas tecnologías permiten cambiar este modelo de gestión de los historiales, orientándolo hacia una gestión electrónica, que dota a estos documentos de cierta ubicuidad, solucionando los problemas que arriba se planteaban. No es difícil encontrar informaciones al respecto en Internet, referidas a este cambio de modelo, como ocurre por ejemplo con el Servicio Andaluz de Salud.

El paso siguiente a la digitalización de los historiales médicos es ofrecerlos a través de la red de redes, Internet, tanto a los propios pacientes, para que consulten datos que les resultan de interés sin tener que desplazarse a su centro hospitalario, como a otros centros hospitalarios, para que otros médicos diferentes al asignado al paciente tengan acceso a su historial en caso de cambio de centro hospitalario o de urgencia médica.

La dificultad que se plantea en el tratamiento automatizado de este tipo de datos es que la normativa en vigor en España[2] cataloga de nivel alto a los ficheros que incorporan información relacionada con la salud de las personas, lo que implica aplicar al historial médico las máximas medidas de seguridad que la normativa exige.

A lo largo del presente trabajo se construye una aplicación de gestión de historiales médicos para lo que previamente se han estudiado los requisitos de seguridad exigibles a la misma, como condicionantes fundamentales para su puesta en funcionamiento.

1 OMS: Organización Mundial de la Salud.

2 Ley Orgánica 15/1999, de 13 de diciembre, de Protección de Datos de Carácter Personal y REAL DECRETO 1720/2007, de 21 de diciembre, por el que se aprueba el Reglamento de desarrollo de la Ley Orgánica 15/1999.

Palabras clave

Apache-Tomcat
Axis
Certificado
Criptosistema simétrico
Criptosistema asimétrico
Clave Pública
Clave Privada
Eclipse
Historial médico
Http
Java
MySQL
PKI
Rampart
RMI
Seguridad Informática
SGBD
SOA
Tcpmon
UML
Web Services
WS
WS-*
WS-Security
WS-I BP
WSS4J
XML
XMLSignature
XMLEncryption
X509v3

Área PFC

Seguridad Informática

Índice de contenido

Índice de figuras

Capítulo 1

1.-Introducción.

1.1.-Justificación del PFC.

Internet es una plataforma de comunicaciones y de información que ha hecho accesibles todo tipo de servicios en prácticamente cualquier punto del planeta. Su expansión está siendo imparable y son cada vez más las líneas de negocio presentes en la red que aprovechan las ventajas que brinda este sistema de comunicación:

- Inmediatez.
- Bajo coste.
- Disponibilidad 24 horas.
- Interactividad.
- Acceso a nuevos mercados y clientes.

Comercio, Banca, Juegos, Servicios Públicos, etc. Internet facilita especialmente la vida a los clientes de estos servicios, pues evita desplazamientos y les permite realizar gestiones en horarios en los que estos establecimientos suelen tener sus ventanillas y mostradores cerrados.

Sin embargo no todo son ventajas; Internet es inherentemente inseguro, es una red plagada de elementos hostiles: virus, troyanos, publicidad, ventanas emergentes inacabables, etc., son muchas las personas reacias a utilizar los servicios presentes en Internet por miedo a tener problemas con los elementos antes mencionados.

Resulta por tanto un requisito fundamental, para cualquier servicio que se ponga a disposición de las personas sobre Internet, que cuente con las debidas garantías en cuanto a seguridad.

La seguridad informática es una disciplina que tiene como objetivo conseguir que los recursos que componen los sistemas de información de una entidad sean utilizados de acuerdo a los principios y las normas que los rigen, manteniéndolos en un estado libre de peligro, daño o riesgo. Es pues un concepto muy amplio, pero válido y aplicable a servicios que se ofrecen sobre Internet.

En este PFC se plantea construir un sistema de gestión de historiales médicos que ofrezca la confianza suficiente a los potenciales usuarios del sistema, para que hagan uso del mismo sin temor a que su intimidad y datos personales sean visibles por terceras personas.

1.2.-Objetivos del PFC.

El objetivo principal del PFC es la construcción de una plataforma informática de gestión de historiales médicos de personas.

Son requisitos fundamentales del sistema a construir:

- Permitir el acceso telemático a los datos contenidos en los historiales, tanto a los propios pacientes como a los médicos responsables de su salud.
- Utilizar Internet como red de comunicación.
- Cumplir la Ley Orgánica 15/1999, de 13 de diciembre, de Protección de Datos de Carácter Personal.
- Contar con suficientes mecanismos de seguridad que garanticen, a un nivel adecuado, los siguientes requerimientos de seguridad:
    1. Confidencialidad: La información intercambiada sólo debe poder ser entendida e interpretada por los actores definidos en el sistema conforme a las reglas de negocio establecidas.
    2. Integridad: No debe ser posible modificar la información intercambiada sin conocimiento de los actores.
    3. Autenticación: No puede existir ninguna duda respecto de la identidad de las personas que intervienen en el sistema.
    4. Autorización: Estará controlado el acceso a la información existente en el sistema, de manera que sólo las personas autorizadas podrán consultar los datos contenidos en los historiales médicos.
    5. No repudio: Las decisiones de los actores, que se plasman en entrada, salida y modificación de las informaciones contenidas en los historiales, no podrán ser rechazadas por los mismos alegando no haberlas solicitado o realizado.
    6. Privacidad: El sistema debe garantizar que la información contenida en los historiales sólo está disponible para las personas que deben poder acceder a ella.
    7. Auditable: Las acciones de los actores del sistema deben quedar auditadas y tales informaciones guardadas de acuerdo a los requisitos que la normativa sobre protección de datos personales establece.
    8. Disponibilidad: El servicio que da la plataforma debe estar disponible las 24 horas del día todos los días del año, con unos tiempos máximos de recuperación ante fallos que se determinen.

1.3.-Enfoque y método seguido.

Metodología.

En la construcción del sistema se utilizará una metodología iterativa e incremental, de forma que se irán sucediendo las fases tradicionales en el desarrollo de sistemas de información (análisis, diseño, construcción, pruebas unitarias, integración, pruebas de integración) para cada uno de los módulos que se vayan construyendo.

Previamente se llevará a cabo un Estudio de viabilidad, que analizará el contexto tecnológico en el que se integrará el sistema a construir y que determinará una serie de requerimientos funcionales y no funcionales que tendrán que ser tenidos en cuenta en las fases posteriores. En esta fase se analizarán diferentes alternativas posibles, los riesgos que implican y se tomará una decisión valorada de qué alternativa es la más viable.

Se utilizará en la medida de lo posible diagramas UML para representar los casos de uso, clases, secuencias, diagramas de estado y de componentes.

En la primera iteración, durante la etapa de análisis, se determinarán los actores del sistema y los casos de uso principales, que se irán refinando posteriormente a medida de que se vayan construyendo los diferentes componentes del sistema.

Plataforma de desarrollo.

En cuanto a la plataforma de desarrollo y de ejecución de los productos obtenidos se seguirá el principio de optar en primer lugar por herramientas que sigan la filosofía código abierto ; caso de que no existiesen, se utilizarán soluciones libres. Solamente se optará por herramientas comerciales si fuese imprescindible su utilización. En el Anexo I se describe la plataforma utilizada, indicándose los procedimientos de instalación de sus distintos componentes en caso de ser conveniente por su dificultad o peculiaridades.

Enfoque.

En un principio, se tenía pensado integrar la aplicación a construir en en entorno basado en una arquitectura orientada a servicios (SOA). Para ello se había considerado que los servicios a ofrecer utilizarían tecnología Web Services para ofrecer a los diferentes clientes del sistema, B2C[3] o bien B2B[4], las funcionalidades que se determinen en el análisis funcional del sistema.

Las ventajas ofrecidas usando Web Services en vez de otras tecnologías como puede ser RMI se analizarán más adelante.

En torno a los Web Services han surgido una serie de iniciativas que favorecen enormemente la interoperabilidad de estos sistemas.

Entre ellos cabe destacar la WS-I, Organización para la interoperabilidad de los Web Services, que tiene como objetivo promover la interoperabilidad de los web Services a través de todas las plataformas, sistemas operativos y lenguajes de programación. Esta organización

3 Business to Business: comercio entre empresas
4 Business to Consumer: comercio empresa-consumidor

ha definido una especificación, WS-I BP[5], que se ha constituido en estándar de 'facto' respecto a la interoperabilidad de Web Services.

En el ámbito de la seguridad en el entorno Web Services, está realizando un importante trabajo de estandarización el consorcio OASIS, con el conjunto de estándares WS-* cuya pila de protocolos se aprecia en el siguiente gráfico[6].

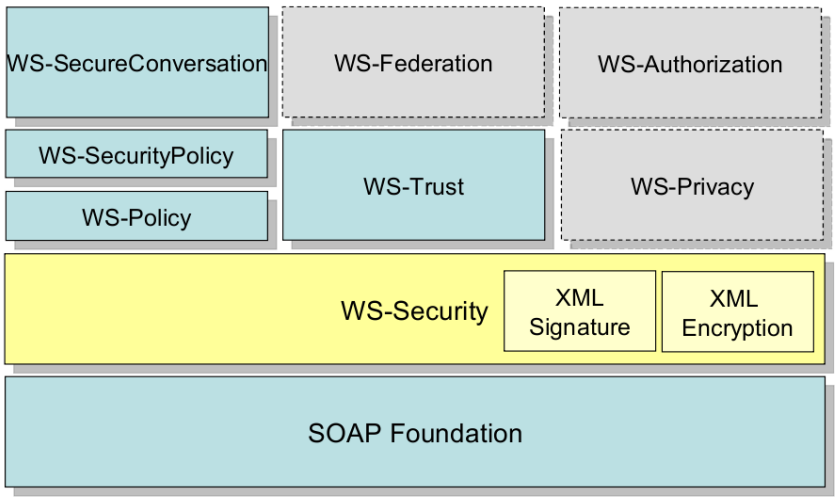


Ilustración 1: Pila WS-Security

El enfoque inicial del PFC está orientado a plasmar, sobre Web Services, y utilizando los estándares anteriores, la aplicación de gestión de historiales médicos. El objetivo es ofrecer unos WS altamente interoperables y a su vez seguros, en la medida que aprovechaban los estándares de seguridad de la pila WS-*. Estándares cuya robustez está basada en que son abiertos y han sido probados e inspeccionados no sólo en el mundo académico, también reciben el apoyo de importantes empresas del ámbito de las tecnologías de la información.

La utilización de estos estándares se ve facilitada por la existencia de numerosos frameworks y herramientas de desarrollo que los integran e implementan, facilitando a los desarrolladores el concentrarse en la lógica del negocio que se pretende construir y dejando a la herramienta la tarea de automatizar otros aspectos como es el de la seguridad. Generalmente el programador define, mediante programación declarativa (a través de wizards o de ficheros xml de configuración) cuales son las políticas de seguridad que debe utilizar la aplicación que está construyendo, encargándose posteriormente una herramienta específica de generar el código necesario.

Con esta finalidad, se han instalado y probado los IDE[7] siguientes:

- Netbeans 6.0 (www.netbeans.org). Se trata de un excelente entorno de desarrollo en Java, que en el campo de los web services ofrece la implementación de los estándares WS-* realizada por el proyecto Tango e integrada perfectamente con los servidores de aplicaciones Glassfishv2, Tomcat y Jboss.

5 WS-I Basic Profile 1.0 Final. Actualmente en borrador la versión 2.0
6 Fuente: http://www.nljug.org/pages/events/content/jfall_2007/sessions/00028/slides/
7 Entorno Integrado de Desarrollo.

- Eclipse 3.3.2 (www.eclipse.org). Es otro IDE extraordinario para trabajar en Java. Cuenta con numerosos plugins, entre ellos WTP (Web Tools Platform), que permite desarrollar Web Services para Axis2.

Los framworks que ofrecen un desarrollo rápido con web services seguros y que han sido probados son:

- Rampart 1.3 (http://ws.apache.org/axis2/modules). Es un módulo que se integra en Axis2, basado a su vez en Apache WSS4J y que proporciona funcionalidad especificada en los estándares WS-Security.
- WSIT (https://wsit.dev.java.net/). Se trata de una plataforma que se integra en los servidores Glashfishv2 o Tomcat y que utiliza la implementación XWSS 3.0 del propio proyecto Tango, ofreciendo igualmente las capacidades definidas en los estándares WS-Security.

Una vez conocidos los requerimientos del PFC, se considera necesario realizar un estudio de viabilidad previo, para determinar si sobre la plataforma de web services descrita, y con la pila de estándares WS-* se pueden implementar los protocolos definidos en el enunciado del PFC.

En cuanto a la infraestructura PKI a utilizar, se considera la idea de utilizar alguna ya existente, por ejemplo el dnie[8] y aquellas otras expedidas por la FNMT, de las que se puede importar el certificado raíz de la CA. El objetivo pretendido es no tener que realizar gestión de certificados (expedición, renovación, revocación, etc.) y basase la aplicación en el trabajo de PKI,s que tienen un uso bastante generalizado en España.

8 Documento Nacional de Identidad electrónico

1.4.-Planificación del proyecto.

A continuación se presenta la planificación de realización del PFC, que a lo largo de su ejecución se ha ido adaptando a la situación real del Proyecto.

Ilustración 2: Planificación PFC

1.5.-Productos obtenidos.

Los productos obtenidos son:

- Aplicación Paciente: Es una aplicación cliente Java, con su correspondiente interfaz gráfica que facilitará a los actores Paciente las funcionalidades definidas en el análisis de requisitos. Incluirá el almacén de certificados y de claves que se determine, según la PKI utilizada.
- Aplicación Médico: Aplicación cliente Java que ofrecerá a los actores Médico las funcionalidades que se definan posteriormente. Tendrá una interfaz gráfica e incluirá el almacén de claves y certificados correspondiente.
- Aplicación Gestor: Aplicación cliente Java, con interfaz gráfica que utilizarán los actores Administrativo . Incluye el almacén de claves y de certificados necesarios.
- Aplicación Servidor: Aplicación servidor Java, sin interfaz gráfica, que contendrá la lógica de la gestión de historiales y se comunicará con la base de datos en la que se almacenarán los datos.
- Scripts para:
  - Creación de tablas en la base de datos.
  - Gestión de claves y certificados (creación de claves y de certificados para la PKI).
- Documentación de la aplicación.

1.6.-Descripción de los demás capítulos de la memoria.

Capítulo 2:

Estudio de viabilidad. Se analiza el contexto en el que se construirá el sistema, los requisitos principales, tanto funcionales como no funcionales y las tecnologías utilizables. En definitiva se plantean alternativas posibles y se decide al respecto.

Capítulo 3:

Infraestructura de clave pública. Del estudio de viabilidad se desprende la necesidad de utilizar una infraestructura de clave pública. En este capítulo se estudia qué es una PKI y se determinan los componentes que serán utilizados en este proyecto.

Capítulo 4:

Esquema criptográfico. Se definen los protocolos de seguridad que serán utilizados y que garantizan los requisitos de seguridad que se hayan fijado.

Capítulo 5:

Representación de los datos. XML. Se opta por utilizar XML como forma de representación de la información intercambiada. Se analiza la información intercambiada, los estándares utilizados y su representación final.

Capítulo 6:

Protocolos de comunicación. En función del análisis de viabilidad se utilizará para el intercambio de información entre las aplicaciones cliente y el servidor RMI o Web Services. En este capítulo se analiza la tecnología utilizada y se diseña, construye, prueba e integra el módulo de comunicaciones.

Capítulo 7:

Persistencia de los datos. La información que utilice la aplicación se guardará en una base de datos. En este capítulo se describe el proceso de construcción de la base de datos y su integración con los demás módulos.

Capítulo 8:

Interfaces gráficas. Corresponde este capítulo al diseño, construcción y prueba de las interfaces gráficas que utilizarán los usuarios del sistema.

Capítulo 9:

Conclusiones.

Capítulo 2

2.-Estudio de Viabilidad.

Para la construcción del sistema de gestión de historiales existen una serie de condicionantes que deben ser considerados y analizados pues determinarán en etapas posteriores las decisiones a tomar.

En primer lugar se analizarán los condicionantes tecnológicos, debido a que normalmente constituyen un elemento dado, en el que los sistemas o aplicaciones nuevas deben integrarse.

A continuación se estudian los condicionantes de seguridad, puesto que este es el marco en el que se desarrolla este PFC.

Finalmente se detallan los condicionantes funcionales, siguiendo una metodología RUP[9] simplificada.

9 Rational Unified Process

2.1.-Condicionantes Tecnológicos.

Se pretende conseguir una plataforma tecnológica estable y altamente escalable, que permita crecer para albergar más servicios, y que a su vez presente bajo acoplamiento entre componentes, de forma que cambios en dicha plataforma supongan el mínimo impacto en los demás componentes.

Teniendo en cuenta lo anterior, los componentes estructurales del sistema serán:

Cliente Web: Se contempla la posibilidad de que se acceda al sistema desde un navegador Web (Iexplorer, Mozilla, etc.).

Cliente Java: Se facilitará a Médicos y Pacientes un cliente con una interfaz más amigable que la ofrecida mediante navegador Web. Para ello se les facilitará una aplicación Java, descargable y actualizable vía Web mediante Java Web Start.

Servidor Web: Al que se conectarán los clientes Web.

WS: Servicios Web, que serán utilizables por las aplicaciones Java cliente y por posibles sistemas de terceros (otros centros hospitalarios, administraciones públicas, etc.).

Negocio: Para desacoplar presentación de información y lógica de negocio, existirá una capa, separada de la de presentación, que será la encargada de los procesos de negocio y de comunicarse con la capa de datos. La capa de Negocio presentará una fachada única para la capa superior.

Integración: Se desacoplará la capa de Negocio de la de Datos a través de una capa de Integración, para facilitar futuros cambios de componentes.

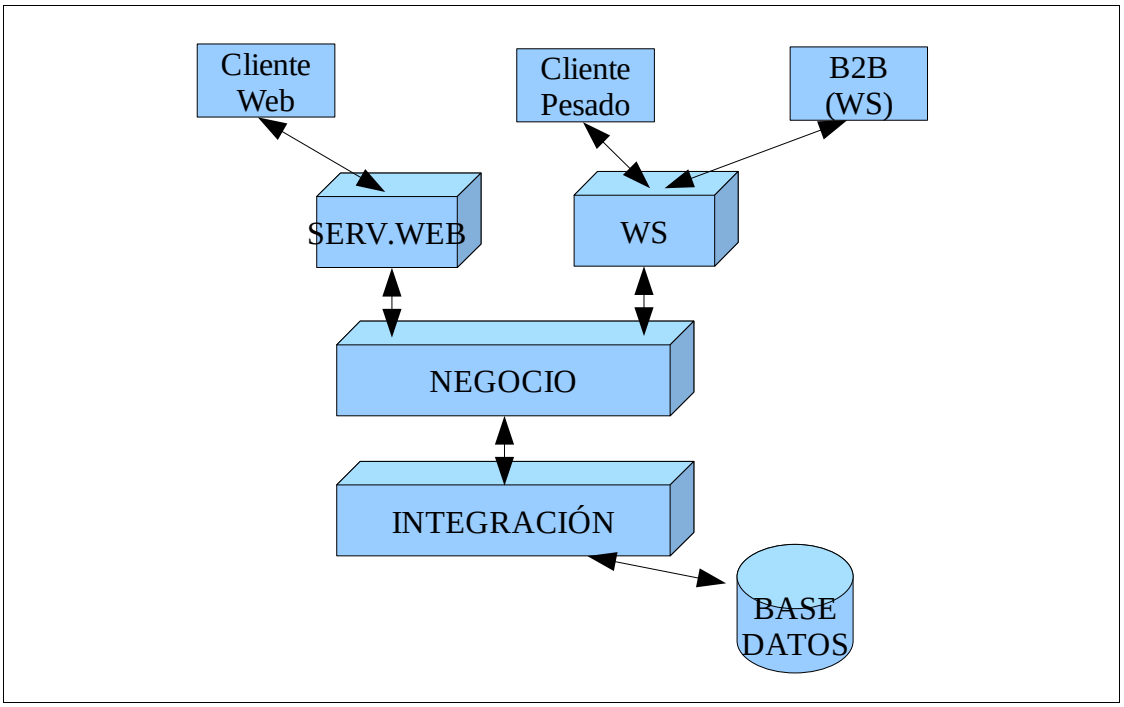


Ilustración 3: Arquitectura Sistema I

El lenguaje de programación será Java en todas las capas, en su última especificación que cuente con una implementación estable.

Si se pone nombre a las tecnologías que se emplearán, tenemos:

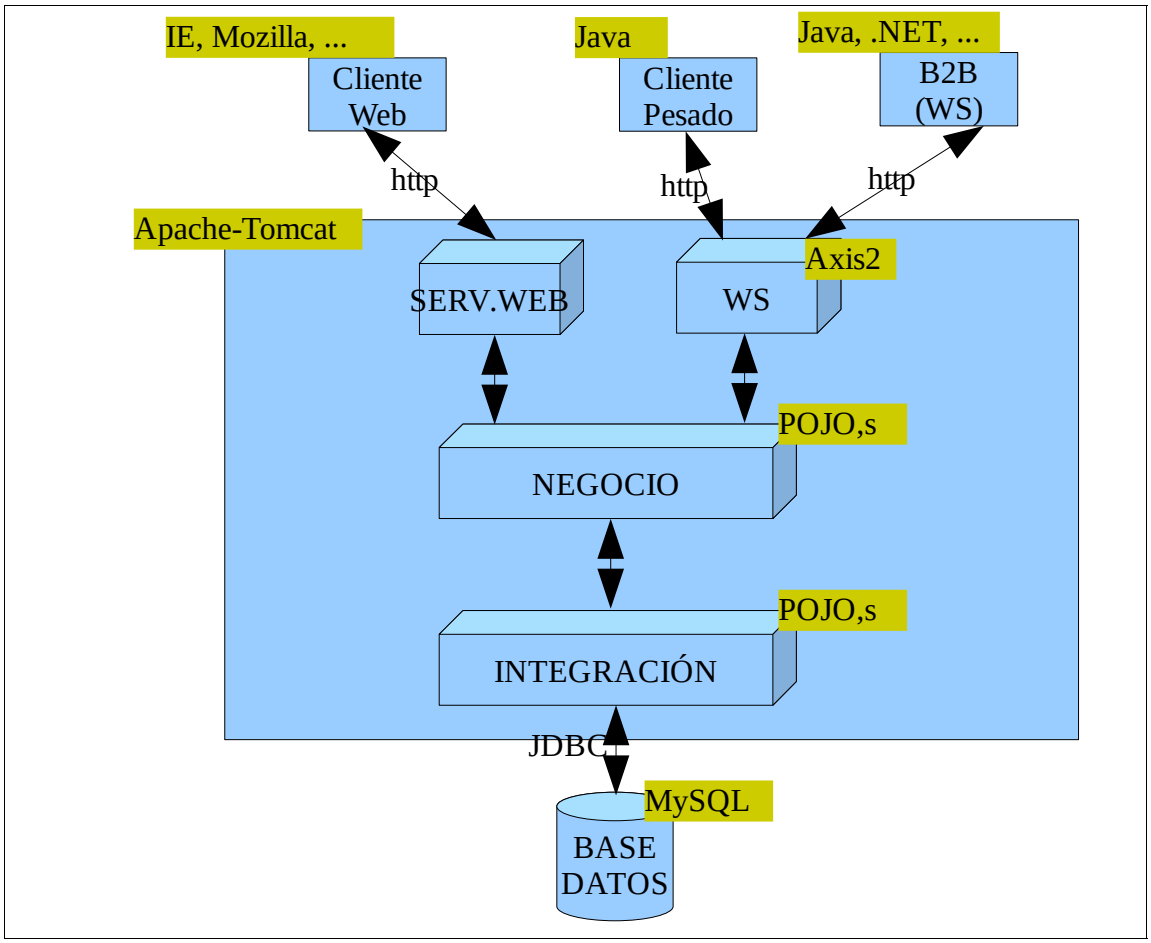


Ilustración 4: Arquitectura Sistema II

Java: En todos los desarrollos Java se utilizará el JDK 6. Es el último kit implementado por Sun, que incorpora mejoras y nuevas Apis en relación con la versión anterior.

Apache-Tomcat: El servidor de aplicaciones que se utilizará será Apache-Tomcat en su versión estable más actual, la 6.0.16. Se ha elegido este servidor por:

- Es Open Source.
- Consolidado y generalizado su uso por todo tipo de empresas y entidades.
- Existe mucha información sobre el mismo en Internet.
- Es fiable.
- Es seguro.
- Integrable fácilmente con Axis2 para la utilización de Web Services.

Web Services: La elección de Web Services frente a a otros protocolos/sistemas de comunicación es debida a que proporcionan una serie de ventajas:

- Bajo acoplamiento. Facilita la interoperabilidad entre plataformas diferentes (Java, .Net, etc.).
- La utilización de XML ofrece flexibilidad e independencia de las tecnologías subyacentes.
- El uso de Web Services no requiere abandonar otras tecnologías, sólo es necesario añadir un front-end de Web Services ante ellas. Esto permite una adopción gradual de esta tecnología.
- Existencia de muchas herramientas y toolkits para crear Web Services.
- La modularidad en el desarrollo de Web Services favorece su utilización.
- Las arquitecturas orientadas a servicios (SOA) encuentran una buena base en los Web Services.
- Utilización de protocolos extendidos en Internet (http) posibilita utilizar las infraestructuras existentes (Servidores de Aplicaciones, Firewalls, etc.).

Axis2: Es un framework desarrollado dentro de los proyectos Apache, por tanto sigue la filosofía Open Source, y se integra perfectamente en el servidor Apache-Tomcat. Existen otros frameworks que permiten la creación y publicación de Web Services, pero se ha optado por Axis2 debido a que:

- Es Open Source.
- Integrable con Apache-Tomcat.
- Está ampliamente extendido su uso.
- Más eficiente que la anterior versión, Axis 1, especialmente en el tratamiento de documentos XML a través de AXIOM, un eficaz Simple API for SOAP and XML .
- Permite fácilmente la incorporación de módulos (ejemplo que se verá más adelante con Rampart) que extienden su capacidad.
- Proporciona un generador de código potente, tanto de código del lado del servidor como para generar clientes.

Axis2 identifica las dos acciones básicas que tienen lugar en el procesamiento SOAP: envío y recepción de mensajes SOAP. Proporciona dos pipes a través de las cuales fluye la información para conseguir las acciones básicas antes mencionadas. Son denominadas In pipe y Out pipe , en base a las cuales se construyen cualquier tipo de patrón de intercambio de mensajes (MEP[10]).

Axis2 proporciona extensibilidad a través de manejadores o handlers . En función de las necesidades, y en el contexto que sea necesario (operación, servicio o global), se conforma una cadena de handlers que procesan el mensaje a enviar hasta que finalmente es tratado por el handler Transport Sender . De igual manera, en la recepción, el mensaje SOAP es procesado en primer lugar por el handler Transport Listener , pasa por la cadena de los

10 Message Exchange Patterns

demás handlers y finalmente lo procesa el handler Message Receiver .

Los handler se integran en diferentes fases, diferenciadas si es el flujo de entrada o de salida. Un handler se ubica en una fase concreta, en una fase coexisten de forma ordenada varios handler.

En el flujo In tenemos las fases:

1. **Transport Phase** Procesan información específica de transporte.
2. **Pre-Dispatch Phase**- Extrae información del mensaje y la coloca en el MessageContext para procesarla, relacionada con acciones de direccionamiento.
3. **Dispatch Phase** Se determina el servicio y operación encargados de atender el mensaje entrante.
4. **User Defined Phases** En este punto es posible incorporar fases definidas por el usuario.
5. **Message Validation Phase** Se procede a la validación del mensaje SOAP.
6. **Message Processing Phase** En este punto se ejecuta la lógica de la operación SOAP a la que va destinado el mensaje.

En el flujo Out tenemos las fases:

1. **Message Initialize Phase** Fase inicial de salida.
2. **User Phases** - En este punto es posible incorporar fases definidas por el usuario.
3. **Transports Phase** Ejecuta las acciones relacionadas con el transporte. Es la fase final.

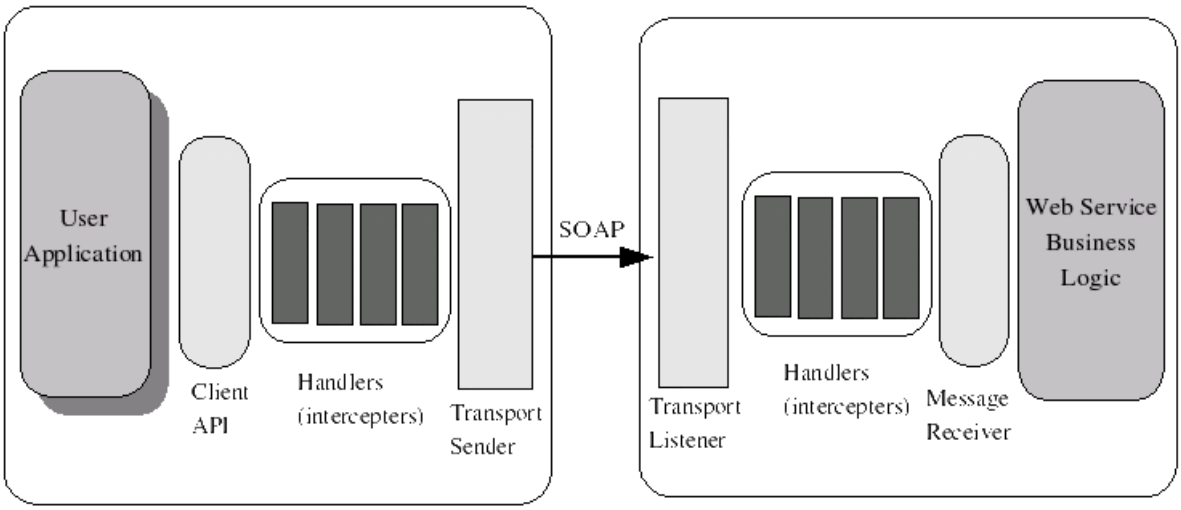


Ilustración 5: Esquema de procesamiento en Axis2

Existen dos formas principales de extender la funcionalidad de Axis2:

1. Insertando handlers de usuario en las fases existentes, o en fases definidas por el usuario. Para ello se construyen los correspondientes handler y se define su posición en la cadena en los ficheros de configuración de Axis2.
2. Creando módulos nuevos. Los módulos están constituidos por un conjunto de handler y un descriptor que indica las fases en las que deben ser incluidos.

Rampart: Es un módulo que implementa las especificaciones relacionadas con seguridad de la pila WS-*. En la versión actual implementa en concreto WS-Security, WS-Trust, WS-SecureConversation, y WS-SecurityPolicy.

Rampart es desarrollado en el seno del proyecto Apache.

Utiliza la librería WSS4J, que pertenece también al proyecto Apache y es la encargada de la implementación de WS-Security, ofreciendo soporte para:

- XML Security
  - XML Signature
  - XML Encryption
- Tokens
  - Username Tokens
  - Timestamps
  - X.509 binary certificates
  - SAML Tokens

WSS4J depende de la implementación Apache XML Security.

POJO,s: La lógica de negocio la implementaremos utilizando POJO,s.

El acceso a los datos se realizará utilizando el patrón DAO[11].

Http: Aunque SOAP puede ser transportado sobre diversos protocolos, se utilizará el más habitual, http, que facilita la utilización de equipamiento y tecnologías preexistentes, como ocurre con firewalls específicos para tratar este protocolo.

MySql: Se trata de un sistema gestor de base de datos que se ofrece bajo licencia GNU GPL, aunque para empresas que deseen adquirir e incorporar privativamente este producto, pueden adquirirlo bajo una licencia específica.

Es un SGBD relacional, multihilo y multiusuario, con una gran aceptación en el mundo del software libre (más de seis millones de instalaciones).

11 Data Access Object

2.2.-Condicionantes de Seguridad.

En el apartado 1.2 de este PFC se especifican los requerimientos de seguridad exigidos. Para alcanzar tales requisitos será necesario utilizar:

- criptografía; con su uso garantizaremos la confidencialidad y la privacidad de los datos.
- firma digital; que permitirá garantizar la integridad de los mensajes y el no repudio de las actividades de los actores. El objetivo de auditabilidad lo alcanzaremos almacenando a tales efectos las peticiones de los usuarios con su correspondiente firma.
- certificados digitales; a efectos de autenticación, objetivo que se alcanzará utilizando a su vez firma digital. También se utilizarán los certificados para conseguir el objetivo de autorización, verificándose que el actor, identificado por su certificado, que solicita algún servicio tiene permiso para lo que pide.

Los elementos anteriores conducen a la utilización de una Infraestructura de Clave Pública (PKI) para gestionar los certificados de los diferentes actores. Posteriormente se determinará los protocolos de intercambio de información y la forma de uso para tal fin de los elementos que componen la PKI.

¿Seguridad a nivel de transporte, seguridad a nivel de mensaje?

Se puede obtener y alcanzar los objetivos de seguridad antes definidos en diferentes niveles de la pila de protocolos de comunicación utilizados. A continuación se analizarán los dos habitualmente utilizados y será seleccionado el más adecuado para el PFC.

A nivel de transporte el procedimiento que generalmente se utiliza para conseguir seguridad en las comunicaciones es el uso de SSL.

Ventajas:

- Ofrece seguridad punto a punto.
- Confidencialidad, integridad y autenticación.
- A día de hoy se consigue por Hardware en muchos equipos (rápido y eficiente).

Inconvenientes:

- No ofrece seguridad end-to-end.
- Todo se cifra, todo se firma.
- No ofrece No-repudio.
- Autorización en capas superiores.

A nivel de mensaje, que corresponde con el nivel de aplicación en la pila de protocolos de comunicación, existen múltiples formas de conseguir seguridad; en este

proyecto se plantea utilizar el estándar WS-Security implementado sobre Web Services.

Ventajas:

- Utilizables diferentes protocolos de transporte.
- Seguridad end-to-end.
- Firma y cifra selectiva.
- Confidencialidad, integridad, autenticación, autorización al mismo nivel.
- Sí ofrece No-repudio.

Inconvenientes:

- No soluciones hardware (de momento).
- Más complejo.

En principio se podría utilizar seguridad a nivel de transporte para conseguir confidencialidad, integridad y autenticación. Puesto que no se plantea el uso de intermediarios o proxys, no habría problemas con el hecho de que es una seguridad punto a punto. Si se combina esta opción con otros componentes a nivel superior, se pueden alcanzar los objetivos de seguridad propuestos. No obstante, pensando en la escalabilidad y en la integración en un mismo nivel de todos los componentes de seguridad, se optará por la seguridad completa a nivel de mensaje.

Seguridad Integral:

No se puede pensar en una aplicación de este tipo sin dar solución a un entorno óptimo en el que se integrará, que aporte seguridad siguiendo los principios modernos de seguridad en profundidad. Sin entrar en el detalle, a continuación se esquematiza un entorno apto para albergar la aplicación de gestión de historiales que se construirá a lo largo de este Proyecto.

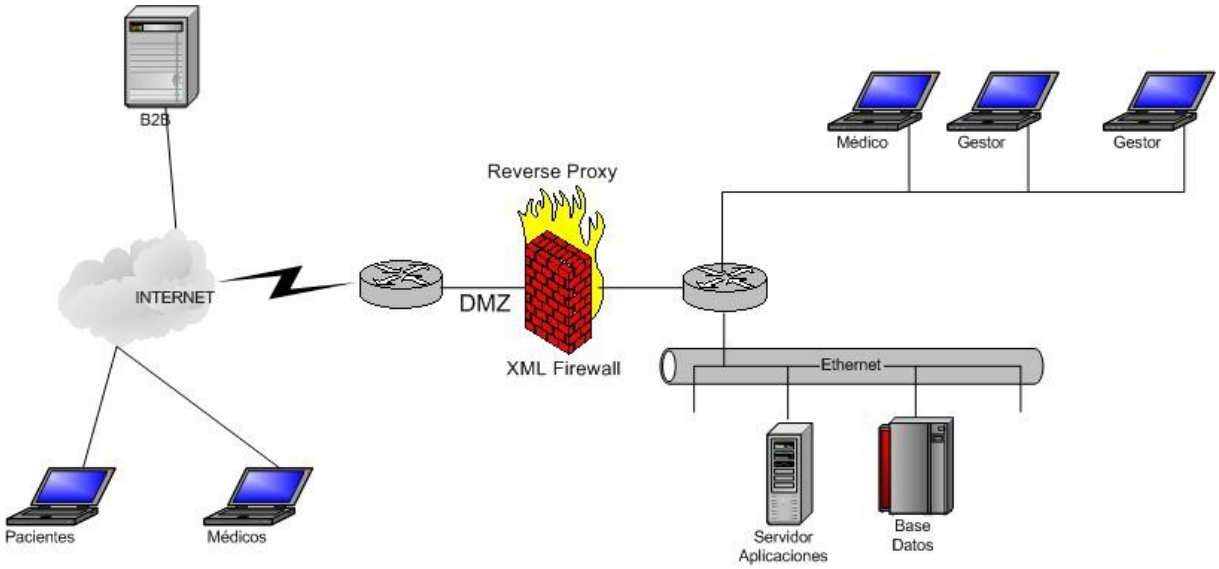


Ilustración 6: Seguridad Integral

2.3.-Condicionantes Funcionales.

En este apartado se realiza el análisis de requisitos funcionales del sistema, definiéndose los actores, casos de uso principales y el modelo de objetos.

2.3.1.-Actores.

Se han identificado los siguientes actores:

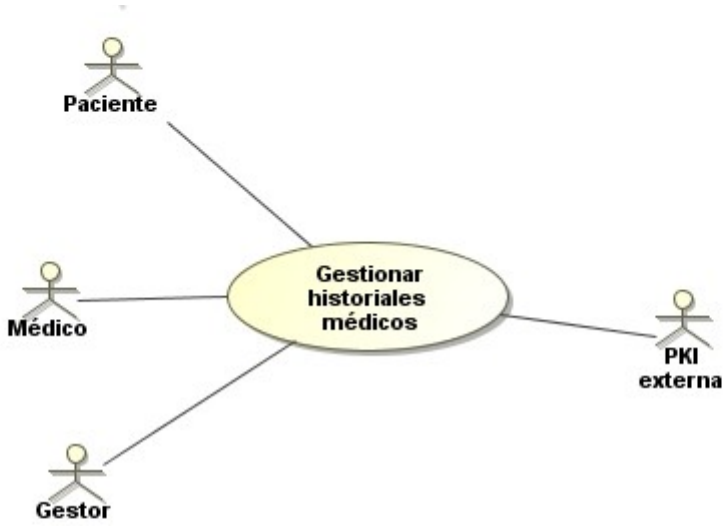


Ilustración 7: Actores

1. Paciente
    - Descripción: Son todas las personas que reciben asistencia médica, cuyos datos personales y de atención componen los historiales médicos.
    - Actividades:
        - Consultar sus datos personales.
        - Consultar sus datos médicos, que incluyen médico asignado y el resultado de las diferentes visitas realizadas.

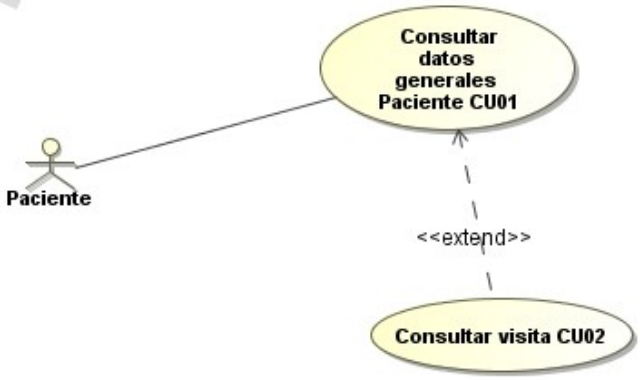


Ilustración 8: Casos de Uso del Paciente

2. Médico
    - Descripción: Personal sanitario que atiende a los pacientes con capacidad para incluir visitas en su historial.
    - Actividades:
        - Consultar datos personales Pacientes asignados.
        - Consultar datos médicos de los Pacientes asignados.
        - Añadir datos médicos (nueva visita) al historial de los Pacientes asignados.

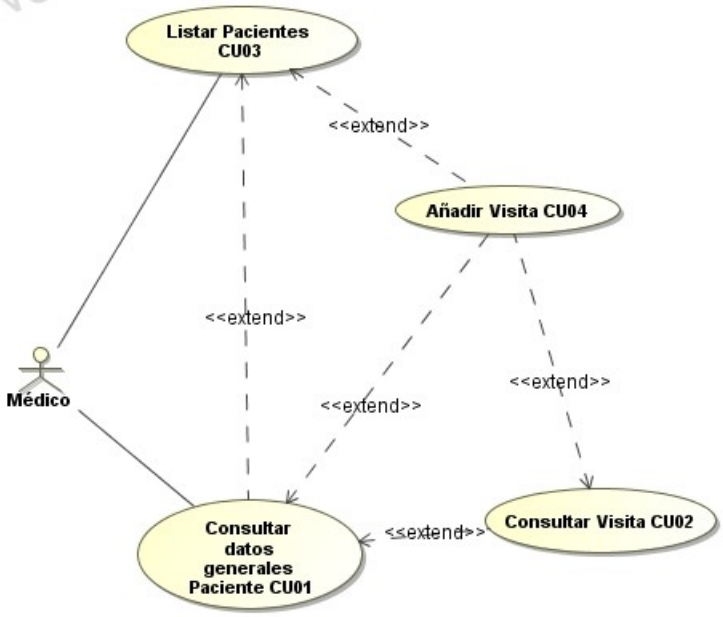


Ilustración 9: Casos de Uso del Médico

3. Gestor:
    - Descripción: Personal administrativo de los centros de atención hospitalaria
    - Actividades:
        - Alta nuevos Pacientes.
        - Baja Pacientes.
        - Alta nuevos Médicos.
        - Baja Médicos.
        - Modificación de los datos personales de los Pacientes.
        - Modificación de los datos personales de los Médicos.
        - Asignar Médico a cada Paciente.
        - Listar Pacientes.
        - Listar Médicos.
        - Listar Pacientes de un Médico.

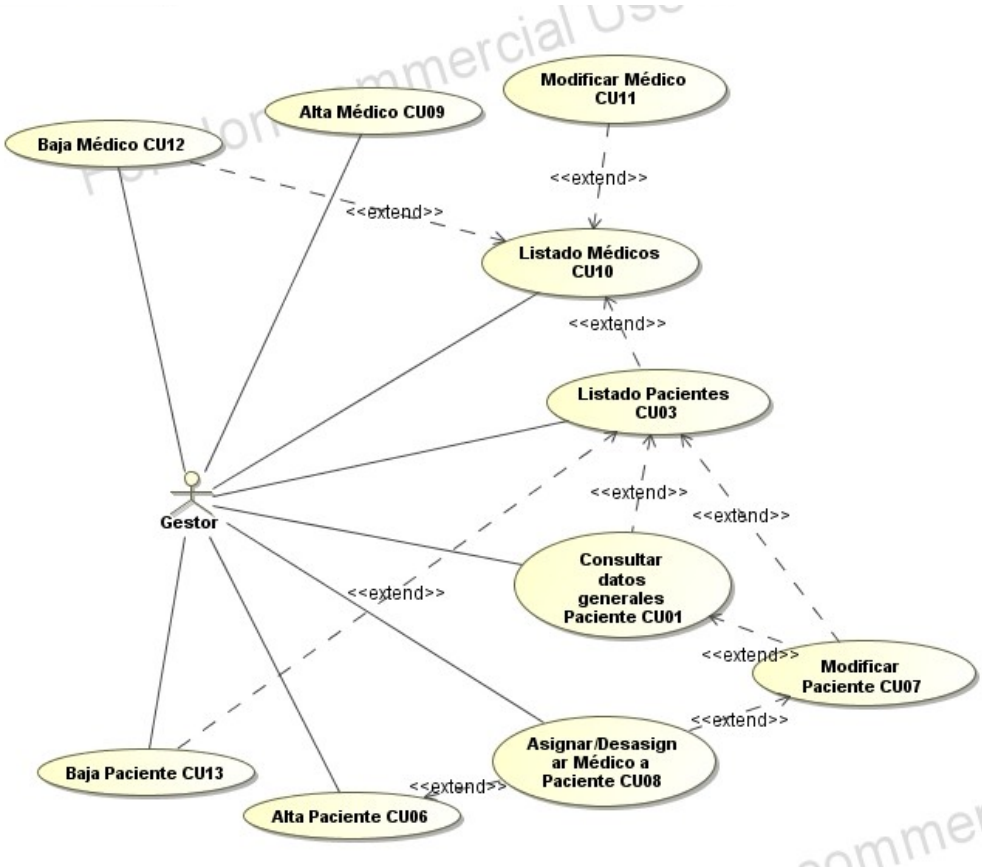


Ilustración 10: Casos de Uso del Gestor

4. PKI externa
    - Descripción: En caso de utilizar una Infraestructura externa de clave pública, sería considerada como actor del sistema.
    - Actividades:
        - Verificar certificados
        - Crear certificados
        - Revocar certificados

Control de acceso a los datos por rol del actor:

Según el perfil de cada actor, los usuarios de la aplicación tendrán acceso a:

Paciente:

Visualización:

-Datos personales propios.

-Historial personal, incluyendo el contenido de las visitas propias.

-Listado de médicos que tiene asignados (nombre y especialidad).

Modificación, creación:

-Ninguno.

Médico:

Visualización:

-Listado de los pacientes que tiene asignados (dni y nombre).

-Datos personales de los pacientes que tiene asignados.

-Historial personal, incluyendo acceso completo a las visitas, de los pacientes asignados.

Creación:

-De visitas a los pacientes que tiene asignados.

Modificación:

-Ninguno.

Gestor:

Visualización:

-Listado de todos los pacientes (dni y nombre).

-Listado de todos los médicos (dni y nombre).

-Datos personales de todos los pacientes.

-Listado de médicos asignados a un paciente.

-Parte general (grupo sanguíneo y alergias) del historial de los pacientes.

-Datos personales de los médicos.

Creación:

-Pacientes (datos personales).

-Médicos (datos personales).

-Asignación de médicos a pacientes.

Modificaciones:

-Datos personales pacientes (no modificable el dni).

-Datos personales médicos (no modificable dni).

-Eliminar asignaciones de médicos a pacientes.

2.3.2.-Casos de Uso.

Han sido identificados 13 casos de uso, que recogen las diferentes formas de utilizar la aplicación por parte de los actores identificados.

Se han definido 3 prioridades, en cada una de las cuales ha sido incluido cada caso de uso.

Posteriormente, cada prioridad dará lugar a una iteración en el desarrollo de la aplicación.

CU01:

Nombre: Consultar datos generales Paciente	**Código: CU01**
Descripción: CU utilizado por Médico, Paciente y Gestor para consultar los datos generales del Paciente.	
Prioridad:	1
Actores:	Paciente, Médico, Gestor
Precondiciones:	El actor ha entrado en la aplicación cliente y se ha autenticado ante ella con el rol adecuado. Si el actor es Paciente, no necesitará indicar el IdPaciente pues sólo puede consultar sus propios datos. Si el actor es Médico, indicará el IdPaciente, o partirá de un listado de Pacientes (CU03) en el que seleccionará uno. Si el actor es Gestor, indicará el IdPaciente, o partirá de un listado de Pacientes (CU05)
Postcondiciones:	El actor visualiza los datos generales de un Paciente, incluyendo los descriptores de sus visitas y los médicos asignados si es Médico o Paciente. Si es Gestor, sólo verá los datos generales y los médicos asignados. Puede: Volver atrás. Visualizar visita (Médico, Paciente). Añadir Visita (Médico). Modificar datos personales (Gestor). Salir de la Aplicación.
PSEUDOCÓDIGO:	1. El actor solicita los datos personales generales añadiendo al mensaje el Id del Paciente: M= ConsultarDatosPersonales + IdPaciente. 2. El cliente[12] añade a M un nonce . 3. El cliente firma M usando el certificado del actor. 4. El cliente cifra el mensaje usando la clave pública del servidor. 5. El cliente envía el mensaje firmado, cifrado y en unión de un Timestamp y del certificado del actor. 6. El servidor recibe el mensaje. 7. El servidor descifra el mensaje. 8. El servidor verifica el certificado del actor.

12 La aplicación cliente

	9. El servidor valida la integridad del mensaje firmado. 10. El servidor valida el Timestamp . 11. El servidor verifica que el actor está autorizado a consultar los datos solicitados. 12. El servidor obtiene los datos personales generales del Paciente IdPaciente, M'. 13. El servidor firma M' usando su certificado. 14. El servidor cifra el mensaje usando la clave pública del actor. 15. El servidor envía el mensaje firmado, cifrado y en unión de un Timestamp y de su certificado. 16. El cliente recibe el mensaje. 17. El cliente descifra el mensaje. 18. El cliente verifica el certificado del Servidor. 19. El cliente valida la integridad del mensaje firmado. 20. El cliente valida el Timestamp . 21. El cliente presenta la información recibida al actor.
EXTENSIONES:	8.1; 9.1; 10.1; 11.1; Si no hay verificación o validación, se devuelve un mensaje de error. 18.1; 19.1; 20.1; Si no hay verificación o validación, se produce un mensaje de advertencia. 21.1 El actor (Médico, Paciente) puede solicitar información adicional relacionada con una de las visitas (CU02). 21.2 El actor (Médico) puede añadir una visita al Paciente (CU04). 21.3 El actor (Gestor) puede modificar los datos del Paciente (CU07).

CU02:

Nombre: Consultar datos visita Paciente	**Código: CU02**
Descripción: CU utilizado por un Paciente o Médico para consultar los datos de una visita.	
Prioridad:	1
Actores:	Paciente, Médico
Precondiciones:	El actor ha entrado en la aplicación cliente y se ha autenticado ante ella con el rol adecuado. Caso Uso CU01
Postcondiciones:	El actor visualiza los datos de una visita de un Paciente. Puede: Volver atrás. Añadir nueva visita (SÓLO MÉDICO). Salir de la Aplicación.
PSEUDOCÓDIGO:	1. El actor solicita los datos de una visita al mensaje el Id del Paciente: M= ConsultarVisita + IdPaciente + IdVisita. 2. El cliente añade a M un nonce . 3. El cliente firma M usando el certificado del actor.

	4. El cliente cifra el mensaje usando la clave pública del servidor. 5. El cliente envía el mensaje firmado, cifrado y en unión de un Timestamp y del certificado del actor. 6. El servidor recibe el mensaje. 7. El servidor descifra el mensaje. 8. El servidor verifica el certificado del actor. 9. El servidor valida la integridad del mensaje firmado. 10. El servidor valida el Timestamp . 11. El servidor verifica que el actor está autorizado a consultar los datos solicitados. 12. El servidor obtiene los datos de la Visita IdVisita, M'. 13. El servidor firma M' usando su certificado. 14. El servidor cifra el mensaje usando la clave pública del actor. 15. El servidor envía el mensaje firmado, cifrado y en unión de un Timestamp y de su certificado. 16. El cliente recibe el mensaje. 17. El cliente descifra el mensaje. 18. El cliente verifica el certificado del Servidor. 19. El cliente valida la integridad del mensaje firmado. 20. El cliente valida el Timestamp . 21. El cliente presenta la información recibida al actor.
EXTENSIONES:	8.1; 9.1; 10.1; 11.1; Si no hay verificación o validación, se devuelve un mensaje de error. 18.1; 19.1; 20.1; Si no hay verificación o validación, se produce un mensaje de advertencia. 21.1 El actor (MÉDICO) puede añadir una visita a un Paciente (CU04).

CU03:

Nombre: Consultar lista Pacientes	**Código: CU03**
Descripción: CU utilizado por un Médico o Gestor para consultar los Pacientes.	
Prioridad:	1
Actores:	Médico, Gestor
Precondiciones:	El actor se ha autenticado ante el cliente con el rol de Médico o Gestor.
Postcondiciones:	El actor visualiza los Pacientes del Médico. Puede: Volver atrás. Salir de la Aplicación Visualizar datos personales de un Paciente. Añadir Visita a un Paciente (Medico). Modificar Paciente (Gestor). Dar de baja un Paciente (Gestor).
PSEUDOCÓDIGO:	1. El actor solicita la lista de Pacientes: M= ConsultarPacientes, IdMedico .

	2. El cliente añade a M un nonce . 3. El cliente firma M usando el certificado del actor. 4. El cliente cifra el mensaje usando la clave pública del servidor. 5. El cliente envía el mensaje firmado, cifrado y en unión de un Timestamp y del certificado del actor. 6. El servidor recibe el mensaje. 7. El servidor descifra el mensaje. 8. El servidor verifica el certificado del actor. 9. El servidor valida la integridad del mensaje firmado. 10. El servidor valida el Timestamp . 11. El servidor verifica que el actor está autorizado a consultar los datos solicitados. 12. El servidor obtiene la lista de Pacientes del IdMedico. 13. El servidor firma M' usando su certificado. 14. El servidor cifra el mensaje usando la clave pública del actor. 15. El servidor envía el mensaje firmado, cifrado y en unión de un Timestamp y de su certificado. 16. El cliente recibe el mensaje. 17. El cliente descifra el mensaje. 18. El cliente verifica el certificado del Servidor. 19. El cliente valida la integridad del mensaje firmado. 20. El cliente valida el Timestamp . 21. El cliente presenta la información recibida al actor.
EXTENSIONES:	8.1; 9.1; 10.1; 11.1; Si no hay verificación o validación, se devuelve un mensaje de error. 18.1; 19.1; 20.1; Si no hay verificación o validación, se produce un mensaje de advertencia. 21.1 El actor puede solicitar los datos personales de un Paciente (CU01). 21.1 El actor (Medico) puede añadir una visita a un Paciente (CU04). 21.1 El actor (Gestor) puede dar de baja un Paciente (CU13)

CU04:

Nombre: Añadir Visita a Paciente	**Código: CU04**
Descripción: CU utilizado por un Médico para añadir una nueva visita a un Paciente.	
Prioridad:	1
Actores:	Médico
Precondiciones:	El actor se ha autenticado ante el cliente con el rol de Médico. Caso de Uso CU03. Caso de Uso CU02. Caso de Uso CU01.
Postcondiciones:	El actor ha añadido una visita a un Paciente.

	Puede: Volver atrás. Salir de la Aplicación.
PSEUDOCÓDIGO:	1. El actor rellena un formulario M= AñadeVisita , con los datos de la nueva visita. 2. El cliente añade a M el IdPaciente. 3. El cliente añade a M un nonce . 4. El cliente firma M usando el certificado del actor. 5. El cliente cifra el mensaje usando la clave pública del servidor. 6. El cliente envía el mensaje firmado, cifrado y en unión de un Timestamp y del certificado del actor. 7. El servidor recibe el mensaje. 8. El servidor descifra el mensaje. 9. El servidor verifica el certificado del actor. 10. El servidor valida la integridad del mensaje firmado. 11. El servidor valida el Timestamp . 12. El servidor verifica que el actor está autorizado a incluir en la información del Paciente la nueva visita. 13. El servidor guarda la nueva Visita del Paciente, devolviendo M'= Ok . 14. El servidor firma M' usando su certificado. 15. El servidor cifra el mensaje usando la clave pública del actor. 16. El servidor envía el mensaje firmado, cifrado y en unión de un Timestamp y de su certificado. 17. El cliente recibe el mensaje. 18. El cliente descifra el mensaje. 19. El cliente verifica el certificado del Servidor. 20. El cliente valida la integridad del mensaje firmado. 21. El cliente valida el Timestamp . 22. El cliente confirma al actor la creación de la visita.
EXTENSIONES:	9.1; 10.1; 11.1; 12.1; Si no hay verificación o validación, se devuelve un mensaje de error. 19.1; 20.1; 21.1; Si no hay verificación o validación, se produce un mensaje de advertencia.

CU06:

Nombre: Alta Paciente	**Código: CU06**
Descripción: CU utilizado por un Gestor para dar de alta a un Paciente.	
Prioridad:	3
Actores:	Gestor
Precondiciones:	El actor se ha autenticado ante el cliente con el rol de Gestor.
Postcondiciones:	El actor ha dado de alta un Paciente. Puede: Volver atrás. Asignar Médico a Paciente

	Salir de la Aplicación.
PSEUDOCÓDIGO:	1. El Gestor envía al servidor los datos con el paciente. 2. El servidor verifica que el rol del actor es correcto. 3. El servidor verifica que no existe el paciente. 4. El servidor da de alta el nuevo paciente. 5. El servidor crea el historial del paciente. 6. El servidor crea una lista de médicos asignados al paciente vacía. 7. El servidor responde al Gestor.
EXTENSIONES:	Asignar Médico a Paciente (CU08)

CU07:

Nombre: Modificar Paciente	**Código: CU07**
Descripción: CU utilizado por un Gestor para modificar los datos de un Paciente.	
Prioridad:	3
Actores:	Gestor
Precondiciones:	El actor se ha autenticado ante el cliente con el rol de Gestor. Caso de Uso CU01. Caso de Uso CU03.
Postcondiciones:	El actor ha modificado los datos personales de Paciente. Puede: Volver atrás. Asignar Médico a Paciente Salir de la Aplicación.
PSEUDOCÓDIGO:	1. El Gestor envía al servidor los datos modificados del paciente. 2. El servidor comprueba el rol del actor. 3. El servidor verifica que el paciente existe. 4. El servidor introduce los nuevos datos. 5. El servidor responde al Gestor.
EXTENSIONES:	Asignar Médico a Paciente (CU08)

CU08:

Nombre: Asignar/Desasignar Médico a Paciente	**Código: CU08**
Descripción: CU utilizado por un Gestor para asignar/des_asignar un Médico a un Paciente.	
Prioridad:	2
Actores:	Gestor
Precondiciones:	El actor se ha autenticado ante el cliente con el rol de Gestor. Caso de Uso CU07. Caso de Uso CU06.
Postcondiciones:	El actor ha asignado un Médico a un Paciente.

	Puede: Volver atrás. Salir de la Aplicación.
PSEUDOCÓDIGO:	1. El actor rellena un formulario M= AsignarMedico| Des_asignarMedico , indicando el IdMedico a asignar| Des_asignarMedico . 2. El cliente añade a M el IdPaciente. 3. El cliente añade a M un nonce . 4. El cliente firma M usando el certificado del actor. 5. El cliente cifra el mensaje usando la clave pública del servidor. 6. El cliente envía el mensaje firmado, cifrado y en unión de un Timestamp y del certificado del actor. 7. El servidor recibe el mensaje. 8. El servidor descifra el mensaje. 9. El servidor verifica el certificado del actor. 10. El servidor valida la integridad del mensaje firmado. 11. El servidor valida el Timestamp . 12. El servidor verifica que el actor es un Gestor. 13. El servidor asigna|desasigna el Medico IdMedico al IdPaciente, devolviendo M'= Ok . 14. El servidor firma M' usando su certificado. 15. El servidor cifra el mensaje usando la clave pública del actor. 16. El servidor envía el mensaje firmado, cifrado y en unión de un Timestamp y de su certificado. 17. El cliente recibe el mensaje. 18. El cliente descifra el mensaje. 19. El cliente verifica el certificado del Paciente. 20. El cliente valida la integridad del mensaje firmado. 21. El cliente valida el Timestamp . 22. El cliente confirma al actor la asignación|des_asignación.
EXTENSIONES:	9.1; 10.1; 11.1; 12.1; Si no hay verificación o validación, se devuelve un mensaje de error. 19.1; 20.1; 21.1; Si no hay verificación o validación, se produce un mensaje de advertencia.

CU09:

Nombre: Alta Médico	**Código: CU09**
Descripción: CU utilizado por un Gestor para dar de alta un Médico.	
Prioridad:	3
Actores:	Gestor
Precondiciones:	El actor se ha autenticado ante el cliente con el rol de Gestor.
Postcondiciones:	El actor ha creado un Médico. Puede: Volver atrás. Salir de la Aplicación.

PSEUDOCÓDIGO:	1. El Gestor envía al servidor los datos con el médico. 2. El servidor verifica que el rol del actor es correcto. 3. El servidor verifica que no existe el médico. 4. El servidor da de alta el nuevo médico. 5. El servidor crea una lista de pacientes vacía asignada al médico. 6. El servidor responde al Gestor.
EXTENSIONES:	

CU10:

Nombre: Listar Médicos	**Código: CU10**
Descripción: CU utilizado por un Gestor para listar los Médicos.	
Prioridad:	3
Actores:	Gestor
Precondiciones:	El actor se ha autenticado ante el cliente con el rol de Gestor.
Postcondiciones:	El actor visualiza los datos de los Médicos. Puede: Volver atrás. Modificar Médico. Listar Pacientes de un Médico. Dar de baja un Médico Salir de la Aplicación.
PSEUDOCÓDIGO:	1. El Gestor solicita la lista con todos los médicos. 2. El servidor verifica el rol del actor. 3. El servidor envía el listado con todos los médicos.
EXTENSIONES:	Listar Pacientes de un Médico (CU03) Modificar datos de un Médico (CU11) Baja de de un Medico (CU12)

CU11:

Nombre: Modificar Medico	**Código: CU11**
Descripción: CU utilizado por un Gestor para modificar los datos de un Medico.	
Prioridad:	3
Actores:	Gestor
Precondiciones:	El actor se ha autenticado ante el cliente con el rol de Gestor. Caso de Uso CU11.
Postcondiciones:	El actor ha modificado los datos personales del Médico. Puede:

	Volver atrás. Salir de la Aplicación.
PSEUDOCÓDIGO:	1. El Gestor envía al servidor los datos modificados del médico. 2. El servidor comprueba el rol del actor. 3. El servidor verifica que el médico existe. 4. El servidor introduce los nuevos datos. 5. El servidor responde al Gestor.
EXTENSIONES:	

CU12:

Nombre: Baja Medico	**Código: CU12**
Descripción: CU utilizado por un Gestor para dar de baja un Medico.	
Prioridad:	3
Actores:	Gestor
Precondiciones:	El actor se ha autenticado ante el cliente con el rol de Gestor. Caso de uso CU10
Postcondiciones:	El actor ha dado de baja a un Médico. Puede: Volver atrás. Salir de la Aplicación.
PSEUDOCÓDIGO:	1. El Gestor solicita la baja de un médico. 2. El servidor verifica el rol del actor. 3. El servidor verifica que existe el médico. 4. El servidor introduce una fecha de baja al médico. 5. El servidor devuelve mensaje de ok.
EXTENSIONES:	

CU13:

Nombre: Baja Paciente	**Código: CU13**
Descripción: CU utilizado por un Gestor para dar de baja un Paciente.	
Prioridad:	3
Actores:	Gestor
Precondiciones:	El actor se ha autenticado ante el cliente con el rol de Gestor. Caso de Uso CU13
Postcondiciones:	El actor ha dado de baja a un Paciente. Puede: Volver atrás. Salir de la Aplicación.

PSEUDOCÓDIGO:	1. El Gestor solicita la baja de un paciente. 2. El servidor verifica el rol del actor. 3. El servidor verifica que existe el paciente. 4. El servidor introduce una fecha de baja al paciente. 5. El servidor devuelve mensaje de ok.
EXTENSIONES:	

2.3.3.-Diagrama de clases.

Se han identificado las siguientes entidades, necesarias para la aplicación:

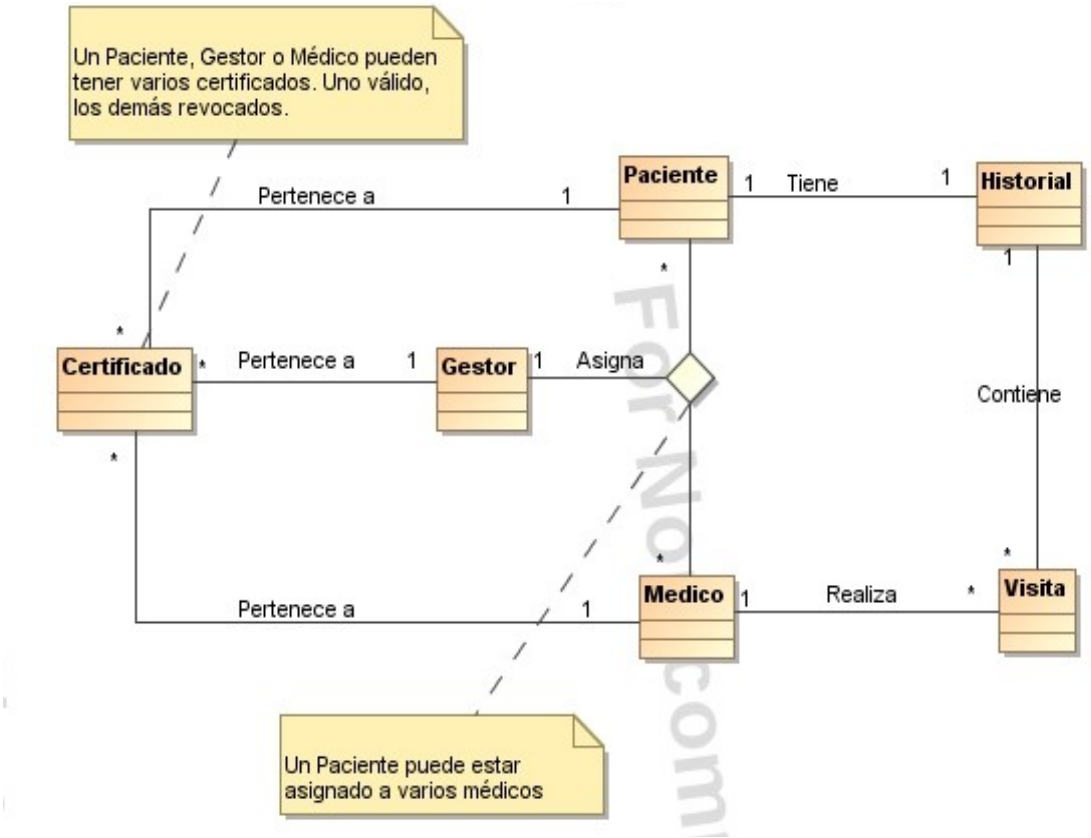


Ilustración 11: Diagrama de clases

Descripción de las clases:

- Paciente: Cada Paciente quedará identificado por su DNI. Se guardarán además los siguientes datos personales:
  - Nombre y apellidos.
  - Dirección.
  - Teléfono.
  - Fecha nacimiento.
  - Nº Seguridad Social.
  - Fecha de alta.
  - Fecha de baja.

- Médico: Cada Médico quedará identificado por su DNI. Se guardan los siguientes datos:
    - Nombre y apellidos.
    - Dirección.
    - Teléfono.
    - Fecha nacimiento.
    - Especialidad.
    - Nº Colegiado.
    - Fecha de alta.
    - Fecha de baja.

- Gestor: Los gestores son personal administrativo. Se guardarán los datos personales siguientes:
    - Nombre y apellidos.
    - Dirección.
    - Teléfono.
    - Fecha nacimiento.
    - Categoría profesional.
    - Puesto Trabajo.
    - Fecha de alta.
    - Fecha de baja.

- Historial: Cada Paciente tiene un único Historial. Este documento contiene unos datos generales, y un conjunto de Visitas. Los datos generales médicos son:
    - Grupo sanguíneo.
    - Alergias.

- Visita: Contiene la información de cada consulta médica. Pertenece a un único Historial. Es realizada por un único médico. Contiene los siguientes datos:
    - Tema
    - Anamnesis.
    - Diagnóstico.
    - Tratamiento.
    - Fecha.

- Certificado: Cada usuario del sistema tiene su propio Certificado digital. Los certificados pueden estar revocados por diferentes motivos. Cada usuario tendrá sólo un Certificado activo.

Capítulo 3

3.-Infraestructura de clave pública.

3.1.-Introducción.

En el Capítulo 2 se determinan una serie de requisitos de seguridad que implican la utilización de cifrado de datos, firma digital y gestión de certificados digitales. Esto nos conduce a plantear la utilización de una Infraestructura de Clave Pública (PKI en inglés) como solución para tales requerimientos.

Una PKI es un conjunto de hardware, software, políticas de seguridad y procedimientos de actuación que tienen como fin la gestión de certificados digitales.

Un **Certificado Digital** es un documento digital mediante el cual un tercero de confianza garantiza la vinculación entre la identidad de un sujeto o entidad y la clave pública contenida en el certificado. Son estos certificados digitales, más concretamente, las claves pública y privada asociadas al mismo, la base sobre la que se fundamentan las cualidades de seguridad proporcionadas por una PKI: confidencialidad, integridad, autenticación y no repudio.

3.2.-Componentes de una PKI.

Los componentes necesarios para una gestión eficiente de certificados digitales son:

1. Autoridad de Certificación (CA[13]): Es la entidad en la que está depositada la confianza por parte de los usuarios de la PKI, por tanto es la que emite y revoca los certificados, garantizando que la clave pública incluida en un certificado pertenece al titular del mismo, cuya identidad se encuentra también en el certificado.
2. Autoridad de Validación (VA[14]): Es la entidad responsable de verificar la validez de los certificados emitidos por la CA.
3. Autoridad de Registro (RA[15]): Se trata de la Autoridad que se encarga de verificar que la clave pública de un certificado pertenece a un subscriptor concreto, cuya identidad está incorporada en el certificado.
4. Autoridad de sellado de Tiempo (TSA[16]): Son entidades de confianza encargadas de asociar, mediante su firma, a un mensaje un instante temporal concreto.
5. Subscriptores: Son las entidades que poseen un certificado emitido por una CA y sus correspondientes claves.
6. Usuarios: Son las entidades que requieren los servicios de la PKI para validar documentos firmados por Subscriptores, o para enviar a éstos documentos cifrados. No requieren por tanto poseer certificado digital propio.
7. Repositorios: Son los almacenes en los que se guardan los certificados:
   - Lista de certificados revocados (RCL[17]): Es un listado con los certificados que han perdido validez antes del tiempo de caducidad previsto en el propio certificado.
   - Repositorio de Certificados: Almacén de los certificados digitales de los subscriptores de una CA.

13 Certificate Authority
14 Validation Authority
15 Registration Authority
16 Time Stamp Authority
17 Revoked Certificate List

3.3.-Certificados digitales.

Como se ha visto más arriba, un **Certificado Digital** es un documento digital mediante el cual un tercero de confianza garantiza la vinculación entre la identidad de un sujeto o entidad y la clave pública contenida en el certificado.

Un certificado digital incluye normalmente:

- Clave privada del subscriptor.
- Identidad del subscriptor.
- Periodo de validez del certificado.
- Localización de una RCL.
- Firma del certificado realizada con la clave privada de la CA.

El formato más extendido de certificado pertenece a un estándar de la ITU-T[18], el X.509. Este estándar define, entre otras cosas, formatos de los certificados, de las RCL, de los atributos de los certificados, etc. La versión v3, más utilizada, tiene la siguiente estructura:

Certificate

- Version
- Serial Number
- Algorithm ID
- Issuer
- Validity
    - Not Before
    - Not After
- Subject
- Subject Public Key Info
    - Public Key Algorithm
    - Subject Public Key
- Issuer Unique Identifier (Optional)
- Subject Unique Identifier (Optional)
- Extensions (Optional)
    - ...

- Certificate Signature Algorithm
- Certificate Signature

18International Telecommunication Union (Telecommunication Standardization Sector)

Los certificados normalmente están almacenados en ficheros; los formatos más habituales son:

- `.CER` - CER certificado codificado, en ocasiones una secuencia de certificados.
- `.DER` - DER certificado codificado.
- `.PEM` - (Privacy Enhanced Mail) certificado DER codificado en Base64, encerrado entre "-----BEGIN CERTIFICATE-----" y "-----END CERTIFICATE-----"
- `.P7B`
- `.P7C` - PKCS#7 Estándar de firma, sin datos firmados.
- `.PFX` - Personal inFormation eXchange
- `.P12` - PKCS#12, contenedor de claves privadas y/o certificados.

3.4.-Protocolos de gestión en una PKI.

Para llevar a cabo la gestión propia de una PKI son necesarios una serie de protocolos, que se muestran a continuación:

1. Generación de claves. Las PKI se basan en la criptografía de clave pública; por ello es necesario crear, para cada certificado, un par de claves, una pública y otra privada. La propia CA dispone de su par de claves que utilizará posteriormente.
2. Petición de certificado. El subscriptor que pretende obtener un certificado se identifica ante una RA y le presenta la clave pública para que la RA verifique a quién se le va expedir el certificado.
3. Certificación: La RA presenta a la CA la petición de certificado, una vez ha validado los datos proporcionados por el subscriptor. La CA firma la clave pública del subscriptor y sus datos personales, y le asigna un periodo de validez al certificado. Utiliza para firmar su propia clave privada.

   La CA dispone de un certificado autofirmado, que contiene su clave pública CA y sus propios datos, firmados con la clave privada de la CA. Este certificado es el que podrán utilizar los usuarios de la PKI para verificar que un certificado ha sido emitido por la CA.
4. Revocación de certificados: Por pérdida, olvido de claves de acceso, etc. se puede solicitar a la CA que revoque un certificado antes de que caduque. Esto supone la inclusión del certificado en una RCL.
5. Renovación: Un subscriptor solicita a la CA que renueve un certificado por caducidad del mismo.

3.5.-PKI para el PFC.

¿Se necesita una PKI propia para este PFC?

Cuando son pocos los usuarios de un sistema, entonces gestionar una PKI propia es posible. El problema surge cuando son miles los subscriptores para los que hay que crear y gestionar sus certificados digitales. En el caso que nos ocupa, una aplicación cuyos potenciales usuarios son el conjunto de médicos, administrativos y pacientes de un sistema sanitario, lo recomendable sería ceder la gestión de la PKI a una entidad especializada en la materia.

Con la extensión en el uso del dnie[19], el sistema debería ser capaz de utilizar los certificados digitales de esta y otras PKIs, lo que mejoraría la experiencia de los usuarios, al poder utilizar un único documento electrónico con múltiples propósitos.

Con fines didácticos, el sistema desarrollado en este PFC utilizará su propia PKI, aunque deberá estar preparado para poder ser utilizado con certificados digitales expedidos por otras PKIs.

Para la creación de la PKI se han utilizado dos herramientas; openssl y keytool. A continuación se especifican las órdenes empleadas en la creación de claves y de certificados de los usuarios:

openssl genrsa -des3 -out CA.key 2048

Resultado: se generan las claves privada y pública, protegidas con TRIPLEDES, con una longitud de clave de 2048 bits, guardadas en el fichero CA.key.

openssl req -new -sha1 -x509 -key CA.key -out CA.crt -days 360 -config openssl.cnf

Resultado: se crea el certificado en formato x509, usando las claves contenidas en el fichero CA.key, válido por 360 días y usando el fichero de configuración openssl.cnf

keytool -genkey -alias paciente -keyalg RSA -dname "CN=Jose Paciente Paciente, C=ES, L=Barcelona, ST=Barcelona, O=UOC, OU=Paciente, DNQ=00000000-P" -keystore paciente.jks

Resultado: se crean las claves para el subscriptor con identificación "CN=Jose Paciente Paciente, C=ES, L=Barcelona, ST=Barcelona, O=UOC, OU=Paciente, DNQ=00000000-P" y se almacenan en el keystore paciente.jks. La clave RSA por defecto es de 1024 bits.

keytool -certreq -keystore paciente.jks -storepass uoc0506 -alias paciente -file paciente.cert.req

Resultado: se crea el requerimiento de certificado para el alias paciente

19 documento nacional de identidad electrónico

openssl x509 -req -in paciente.cert.req -days 180 -CA CA.crt -CAkey CA.key -CAcreateserial -extfile openssl.cnf -extensions usr_cert -out paciente.crt

Resultado: se crea el certificado para el paciente, válido por 180 días

keytool -import -file CA.crt -keystore paciente.jks -storepass uoc0506 -alias ca

Resultado: se añade el certificado de la CA al keystore

keytool -import -file paciente.crt -keystore paciente.jks -storepass uoc0506 -alias paciente

Resultado: se añade el certificado del paciente al keystore

keytool -list -v -keystore paciente.jks

Resultado: se muestra en pantalla el contenido (modo verbose) del keystore

El proceso anterior de creación de certificados debe repetirse para cada uno de los actores.

keytool -import -file servidor.crt -keystore paciente.jks -storepass uoc0506 -alias servidor

Resultado: se añade el certificado del servidor al keystore

Capítulo 4

4.-Esquema criptográfico.

Teniendo en cuenta los requerimientos de seguridad mencionados en los apartados anteriores, se propone utilizar un esquema criptográfico de intercambio de mensajes apoyado en los estándares XML Signature y XML Encryption. Esto facilitará el trabajo posterior, puesto que ambos estándares están definidos en JSR,s[20] en concreto JSR-105 (XML Signature) y JSR-106 (XML Encryption). Existen numerosas implementaciones de estos estándares, lo que dará agilidad a los desarrollos necesarios.

Aunque se detallará más adelante, parte de las tareas criptográficas serán realizadas por el framework utilizado para la transmisión de información entre cliente y servidor (Axis2).

Una cuestión importante es la autenticación mutua entre actores, de tal manera que, tanto cliente como servidor tengan la certeza de estar tratando con el interlocutor adecuado. Aunque se planteaba la utilización del protocolo de Needham-Schroeder, al final se ha considerado más adecuado utilizar la firma digital en cada mensaje, para garantizar la autenticación de los actores. La ventaja del procedimiento elegido es que, con un único mensaje, el emisor está autenticado, mientras que, con el protocolo de Needham-Schroeder son necesarios tres mensajes previos para obtener autenticación entre los interlocutores (actor y servidor).

20 Java Specification Request

4.1.-Esquemática de los procedimientos de firma y cifra utilizados.

El procesamiento de firma se realizará conforme al estándar JSR-105 (XML Signature). Las Java Specification Requests (JSR) constituyen el modo en el que la comunidad que mantiene y construye el lenguaje Java define las especificaciones de este lenguaje. La JSR-105 en concreto se basa a su vez en la especificación para firma digital del W3C. Esquemáticamente, el proceso es el siguiente:

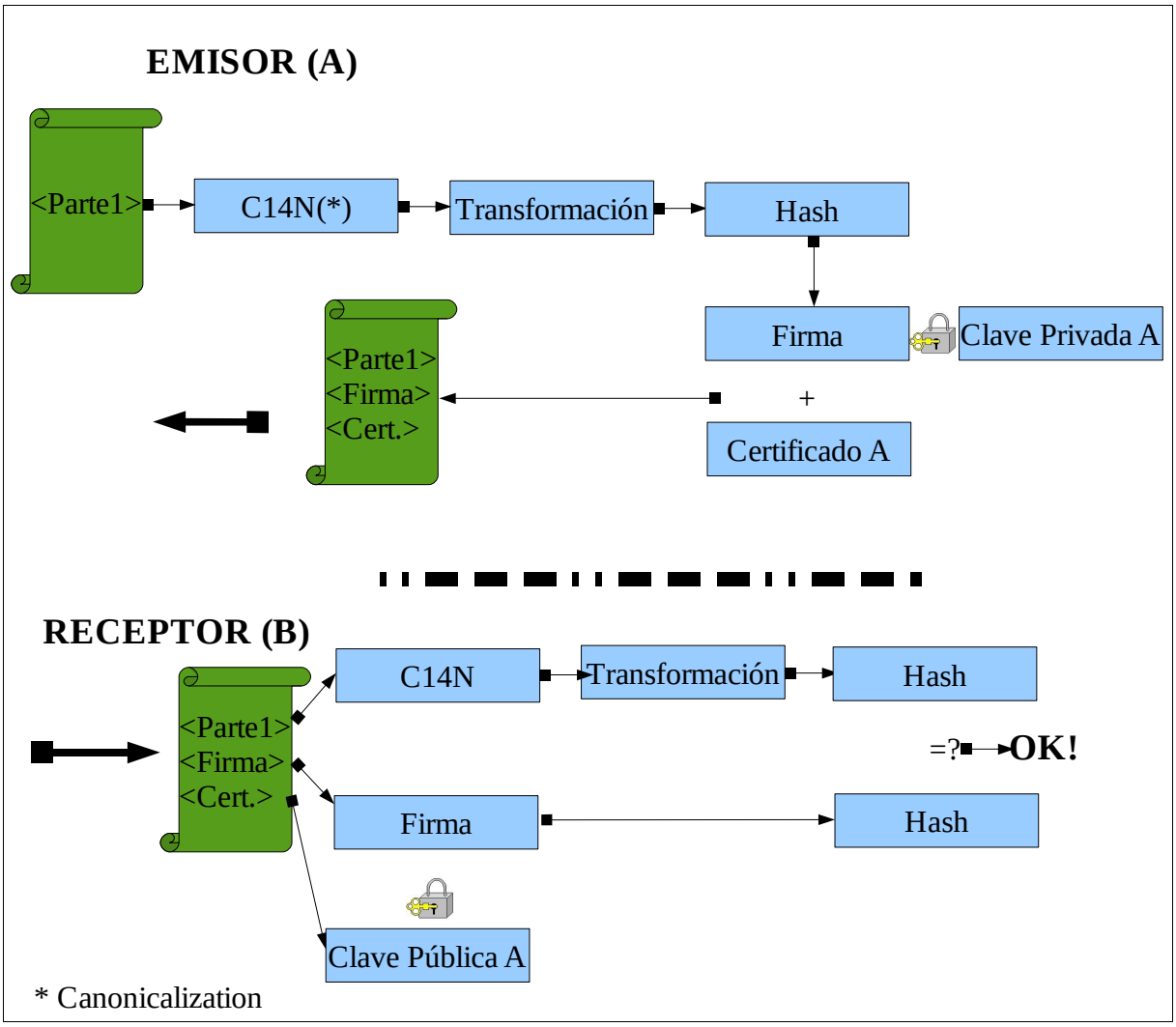


Ilustración 12: XML Signature

La parte del mensaje a firmar es transformada (proceso de canonicalización) para posteriormente calcular un hash de la misma. Este hash es cifrado con la clave pública del emisor, lo que constituye la firma del mensaje.

El proceso de verificación transforma nuevamente el mensaje, obtiene el hash y lo compara con el hash descifrado con la clave pública del emisor.

El procesamiento de firma se realizará conforme al estándar JSR-106 (XML Encryption). Esquemáticamente el proceso es el siguiente:

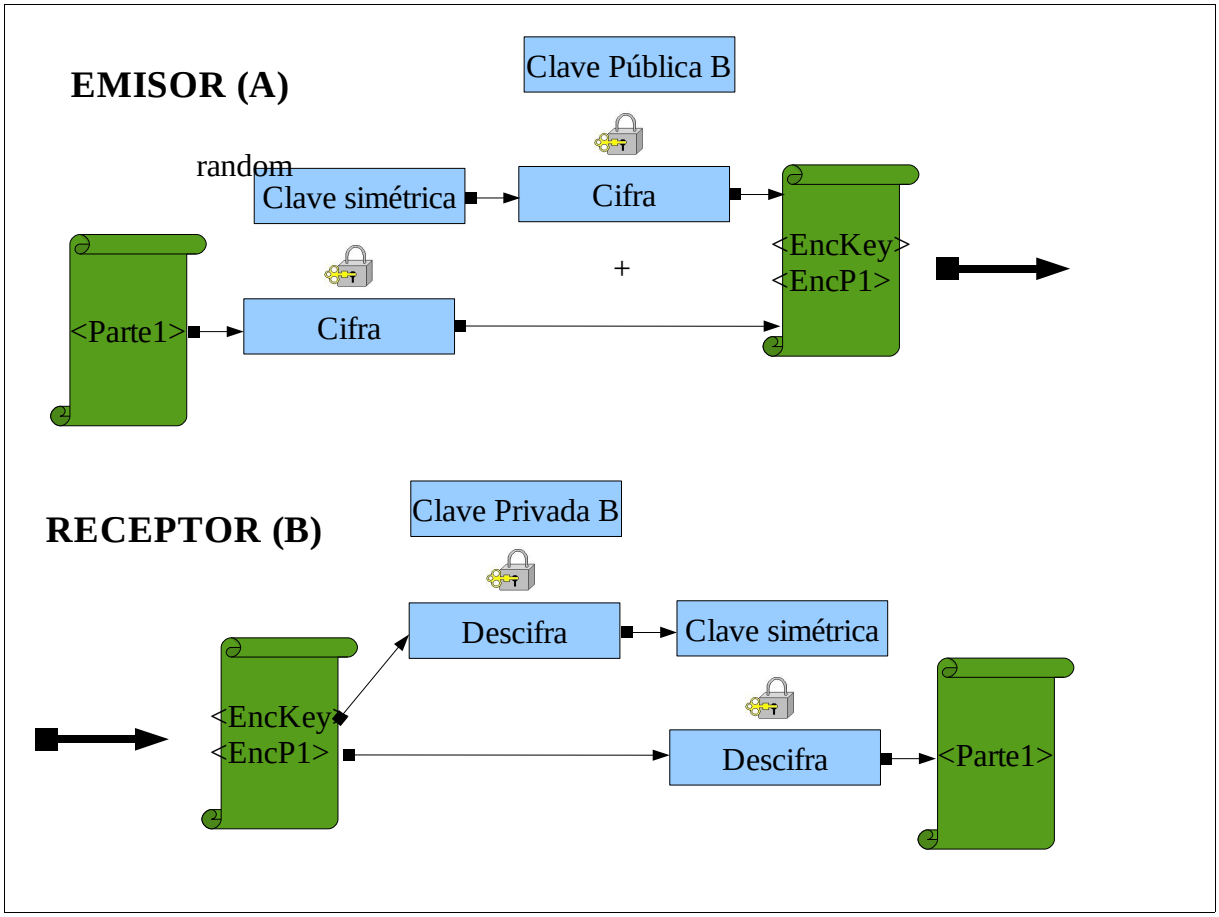


Ilustración 13: XML Encryption

En realidad se cifra el mensaje con una clave y algoritmo simétrico (computacionalmente miles de veces más eficiente que el cifrado con clave asimétrica) y se envía la clave usada para el cifrado simétrico cifrada con el algoritmo asimétrico de clave pública.

4.2.-Notación utilizada.

La notación utilizada será:

" K: clave de criptosistema simétrico .

" E_K (M): cifrado simétrico mensaje M con clave K.

" D_K (C): descifrado simétrico del criptograma C con la clave K.

" ($P_{Entidad}$, $S_{Entidad}$): pareja de claves asimétricas del usuario Entidad, P la clave pública, S la privada.

" $S_{Entidad}$ [M]: Firma digital mensaje M con la clave privada S.

" $P_{Entidad}$ [M]: Cifraje del mensaje M con la clave asimétrica P perteneciente a Entidad.

" H(M): función resumen sobre el mensaje M.

" $Cert_{Entidad}$: Certificado digital perteneciente a Entidad.

4.3.-Organización de la información.

En la base de datos estará protegida la información clínica de los pacientes, de forma que, salvo los usuarios autorizados, no se podrá relacionar datos personales de un paciente con sus visitas médicas. De igual manera, estará protegida la relación médico y pacientes asignados. Los datos generales del historial (grupo sanguíneo y alergias) no quedan protegidos.

En el Capítulo 2 se describen las clases necesarias para implementar el sistema y sus atributos principales.

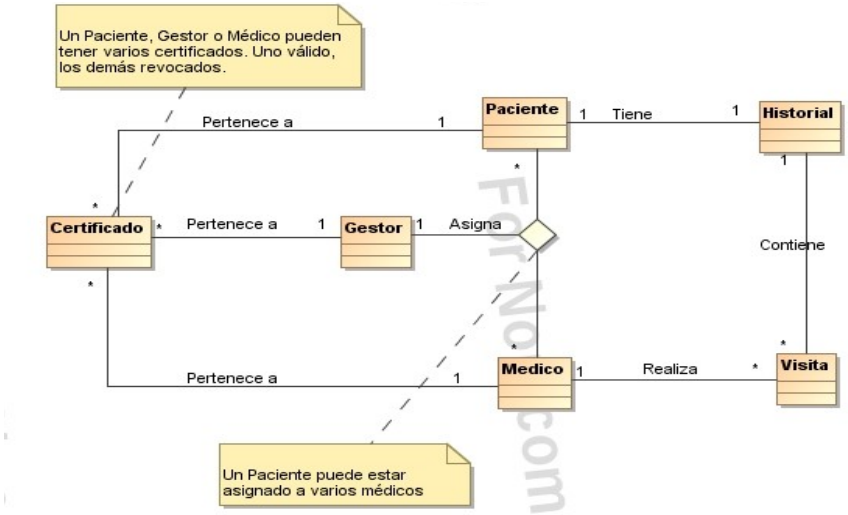


Ilustración 14: Clases del Sistema

La relación (1:n) Historial-Vistas queda protegida creándose:

ListaVisitaProtegida, que contiene ($E_K(L)$, $S_{servidor}[H(E_K(L))]$, $P_{servidor}[K]$):

- $E_K(L)$=Lista de descriptores (Identificadores de cada visita), cifrada con algoritmo simétrico y una clave de sesión K.
- $S_{servidor}[H(E_K(L))]$=Firma de un hash de la lista con la clave privada del servidor.
- $P_{servidor}[K]$=K cifrada con la clave pública del servidor.

La relación (n:n) Medico-Paciente queda protegida creándose:

ListaMedicosProtegida, asociada a cada Paciente, que contiene ($E_{K'}(L')$, $S_{servidor}[H(E_{K'}(L'))]$, $P_{servidor}[K']$):

- $E_{K'}(L')$=Lista de los médicos (Identificadores de cada médico) y rango de fechas límite de acceso a los datos del paciente, cifrada con algoritmo simétrico y una clave de sesión K'.
- $S_{servidor}[H(E_{K'}(L'))]$=Firma de un hash de la lista con la clave privada del servidor.
- K' cifrada con la clave pública del servidor.

ListaPacientesProtegida, asociada a cada Médico, que contiene ($E_{K''}(L'')$, $S_{servidor}[H(E_{K''}(L''))]$, $P_{servidor}[K'']$):

- $E_{K''}(L'')$=Lista de los pacientes (Identificadores de cada paciente), cifrada con algoritmo simétrico y una clave de sesión K".
- $S_{servidor}[H(E_{K''}(L''))]$=Firma de un hash de la lista con la clave privada del servidor.
- $P_{servidor}[K'']$=K" cifrada con la clave pública del servidor.

Respecto a cómo guardar y proteger el acceso a las tres listas mencionadas más arriba, se planteaban inicialmente dos opciones:

1. Cifrar las claves de sesión con las claves públicas de los usuarios con autorización al contenido de las listas.
   - Ventajas: Se podría enviar a los actores las listas cifradas, sin temor a que usuarios no autorizados accedan a su contenido. Con esto se descarga de una tarea intensiva en proceso (descifrado de las listas) al servidor, cediendo esta función a los clientes.
   - Inconvenientes:
     - Complica la política de gestión de las claves de sesión: si no se cambia tras cada consulta, se ha entregado la clave de sesión a un actor, que podría atacar posteriormente la lista en base de datos con la misma clave. Si se cambia la clave de sesión después de cada consulta, se elimina la ventaja indicada en el punto anterior.
     - Cuantos más actores tengan derecho de acceso a una lista, más veces hay que cifrar la clave de sesión con un algoritmo asimétrico, proceso computacionalmente costoso.

2. Cifrar las claves de sesión únicamente con la clave pública del servidor.
   - Ventajas:
     - La clave de sesión no sale al exterior.
     - Gestión de la política de cambio de la clave de sesión más sencilla.
   - Inconvenientes: El proceso de descifrado de las listas se descarga en el servidor, teniendo un coste computacional muy elevado para el servidor.

Se estudió también la opción de utilizar, para verificar que el contenido de las listas no había sido modificado, un algoritmo HMAC en vez de firmar digitalmente las listas. La ventaja de usar un código HMAC[21] frente a la firma, radica en que es mucho más eficiente, pues no utiliza ningún algoritmo de clave pública en su proceso. La desventaja está en que es necesario utilizar una clave conocida en el procesado, que posteriormente habrá que mantener guardada cifrada, y gestionar su ciclo de vida. Por estas complicaciones adicionales se ha optado por utilizar la firma digital.

21 keyed-Hash Message Authentication Code (HMAC or KHMAC)

4.4.-Protocolos y procedimientos utilizados.

Los protocolos de intercambio de información llamarán a unos procedimientos que serán comunes a todos ellos.

Primero se definirán los procedimientos comunes, para luego pasar a detallar los protocolos.

4.4.1.-Procedimientos comunes.

Procedimiento 1 (Solicitud de un servicio). ***Solicita(M)-->M'***

El **actor** solicita el servicio M al **servidor**; ejemplos de solicitudes de servicio:

M=ConsultaDatosPersonales, IdPaciente.

M=ConsultaVisita, IdVisita.

M=AñadirVisita, IdPaciente, (anamnesi,...).

Pasos seguidos por el procedimiento:

1 Calcular N= nonce[22] .

2 Añadir N a M.

3 Calcular un T= Timestamp[23] .

4 Obtener un Hash de (M,N): H(M,N).

5 Firmar H(M,N) con la clave privada del actor: S_{actor} [H(M,N)].

6 Obtener un Hash de (T): H(T).

7 Firmar H(T) con la clave privada del actor: S_{actor} [H(T)].

8 Calcular una clave simétrica K.

9 Cifrar (M,N) con K: E_K (M,N).

10 Cifrar K con la clave pública del servidor: $P_{servidor}$ [K].

11 **Devuelve** M': $P_{servidor}$ [K], T, S_{actor} [H(M,N)], S_{actor} [H(T)], E_K (M,N), $Cert_{actor}$.

Explicación del protocolo:

- Se Utilizará un nonce incluido en el mensaje para añadir aleatoriedad; de esta forma se dificulta un ataque de texto en claro conocido puesto que M en determinadas ocasiones es fijo o fácil de conocer.

22 Cadena aleatoria, para impedir ataques de texto en claro conocido o de repetición de mensaje.
23 Marca de tiempo, para impedir ataques de repetición.

- El Timestamp dificulta los ataques de repetición, ya que marca una franja horaria de validez del mensaje. Se firma para evitar modificaciones del mismo. Dado que va en la cabecera SOAP del mensaje, se firma separadamente del cuerpo del mensaje.
- El protocolo de intercambio de clave simétrica que se utilizará: RSA1_5
- Se cifra con clave simétrica por eficiencia. Se utiliza el algoritmo AES con longitud de clave 256 bits.
- Se firma el Hash, en vez del mensaje, igualmente buscando eficiencia. Se utilizará el algoritmo SHA1.
- Se envía el Certificado del Emisor para que el receptor no lo tenga que tener almacenado y poder obtener la clave pública del emisor a efectos de comprobar las partes firmadas.
- Objetivos alcanzados:
    - Integridad de (M,N) y de (T). Cualquier modificación de los otros componentes del mensaje no firmados ($P_{servidor}$ [K], E_K (M,N), $Cert_{actor}$) es detectada. Si se modifica $P_{servidor}$ [K] o E_K (M,N), una vez desencriptado (M,N) fallaría la verificación de su firma. $Cert_{actor}$ está firmado por el emisor del certificado, por lo tanto su integridad puede verificarse.
    - Confidencialidad de (M,N).
    - No repudio de (M,N) y de (T).
    - Autenticación a través de $Cert_{actor}$.
- Problemática del protocolo:
    - El certificado del emisor se envía en claro porque no se considera necesario ocultar la identidad de este actor. Si fuese necesario, se podría encriptar antes de enviarlo con la clave simétrica utilizada para cifrar el cuerpo del mensaje.

Procedimiento 2 (Recepción de solicitud): ***RecibeSolicitud(M')-->M''***

El **servidor** trata el mensaje del **actor**, validándolo y extrayendo la información necesaria para responder a la solicitud. Devuelve el contenido del mensaje M, La identidad y el rol del propietario del certificado usado para la firma del mensaje:

M"= (M, Id_{Cert}, rol_{Cert}).

Pasos seguidos por el servidor (ha recibido: $P_{servidor}$ [K], T, S_{actor} [H(M,N)], S_{actor} [H(T)], E_K (M,N), $Cert_{actor}$):

1 Verificar validez de $Cert_{actor}$. Firmado por CA confianza, no caducado, no revocado.

1.1 Si no válido, devuelve error.

2 Obtiene K=$S_{servidor}$ [$P_{servidor}$ [K]].

3 Obtiene (M,N)=D_K (E_K (M,N)).

4 Extrae P_{actor} de $Cert_{actor}$.

5 Calcula el Hash de (M,N): H(M,N). (M,N) obtenido en punto 3.

6 Verifica $H(M,N)=P_{actor}[S_{actor}[H(M,N)]]$?

6.1 Si no igual, devuelve error.

7 Calcula el Hash de (T): H(T).

8 Verifica que $H(T)=P_{actor}[S_{actor}[H(T)]]$?

8.1 Si no igual, devuelve error.

9 Verifica que T no ha caducado.

9.1 Si caducado, devuelve error.

10 Extrae el Id del actor de $Cert_{actor}$.

11 Extrae rol[24] del actor del $Cert_{actor}$.

12 Audita($P_{servidor}[K]$, T, $S_{actor}[H(M,N)]$, $S_{actor}[H(T)]$, $E_K(M,N)$, Id_{actor})[25].

13 Guarda $Cert_{actor}$ para la posterior contestación al actor.

14 Devuelve (M, Id_{actor}, rol_{actor}).

Explicación del protocolo:

- Se debe validar que el certificado utilizado es válido, no caducado ni revocado.
- El Timestamp tendrá un periodo de validez.
- Se usa el $Cert_{Emisor}$ con fines de Autorización.

Procedimiento 3 (Envío respuesta): ***EnviaRespuesta(M''')-->M^{IV}***

El **servidor** crea el mensaje a enviar, construyendo la respuesta M'", por ejemplo:

M'"=Listado de Pacientes.

M'"=Historial de Id_{actor}

Pasos seguidos por el procedimiento:

1 Calcular N'= nonce .

2 Añade N' a M'".

3 Calcular un T'= Timestamp .

4 Obtener un Hash de (M'",N'): H(M'",N').

5 Firmar H(M'",N') con la clave privada del servidor: $S_{servidor}[H(M'",N')]$.

6 Obtener un Hash de (T'): H(T').

7 Firmar H(T') con la clave privada del servidor: $S_{servidor}[H(T')]$.

24 Si el solicitante es médico, paciente o gestor.

25 Se auditarán las acciones que se considere necesario

8 Calcular una clave simétrica K'.

9 Cifrar (M''',N') con K': $E_{K'}$ (M''',N').

10 Cifrar K' con la clave pública del actor: P_{actor} [K'].

11 Devuelve M^{IV}: P_{actor} [K'], T', $S_{servidor}$ [H(M''',N')], $S_{servidor}$ [H(T')], $E_{K'}$ (M''',N'), $Cert_{servidor}$.

Explicación del protocolo:

- Es equivalente al utilizado por el solicitante. La clave pública del actor se extrae del certificado del actor, obtenido en el procedimiento 2.

Procedimiento 4 (Recepción de solicitud): ***RecibeSolicitud(M^{IV})-->M'''***

El **actor** trata el mensaje del **servidor**, validándolo y extrayendo la información necesaria para presentarla al usuario.

Pasos seguidos por el actor (ha recibido: P_{actor} [K'], T', $S_{servidor}$ [H(M''',N')], $S_{servidor}$ [H(T')], $E_{K'}$ (M''',N'), $Cert_{servidor}$):

1 Verifica validez de $Cert_{servidor}$. Firmado por CA confianza, no caducado, no revocado.

1.1 Si no válido, devuelve error.

2 Obtiene K'=S_{actor} [P_{actor} [K']].

3 Obtiene (M''',N')=$D_{K'}$ ($E_{K'}$ (M''',N')).

4 Extrae $P_{servidor}$ de $Cert_{servidor}$.

5 Calcula el Hash de (M''',N'): H(M''',N'). (M''',N') obtenido en punto 3.

6 Verifica H(M''',N')=$P_{servidor}$ [$S_{servidor}$ [H(M''',N')]]?

6.1 Si no igual, devuelve error.

7 Calcula el Hash de (T'): H(T').

8 Verifica H(T')=$P_{servidor}$ [$S_{servidor}$ [H(T')]]?

8.1 Si no igual, devuelve error.

9 Verifica que T' no ha caducado.

9.1 Si caducado, devuelve error.

10 Devuelve (M''').

Procedimiento 5 (Auditoría): *Audita(M^V)-->void*

El servidor guarda con fines de auditoría las solicitudes de los actores. Recibe M^V=($P_{servidor}$ [K], T, S_{actor} [H(M,N)], S_{actor} [H(T)], E_K (M,N), Id_{actor})

1. Guarda en base de datos la información recibida.

Explicación del protocolo:

- Es necesario que se guarde determinada información para auditoría; por ejemplo, la incorporación de una visita por parte de un médico. Con la información guardada encriptada con la clave pública del servidor, posteriormente se puede obtener cuándo y qué solicitó cada actor.

4.4.2.-Protocolos utilizados por los actores del sistema.

1. **<u>Autenticación del usuario ante la aplicación cliente:</u>**
    - Al iniciar la aplicación cliente, se solicitará al usuario que introduzca su Id (dni) y la clave de acceso a la clave privada contenida en el keystore donde a su vez están los certificados propio y del servidor. La clave de acceso al propio keystore debe ser conocida por la aplicación.
    - La aplicación cliente verifica que los datos de acceso son correctos:
        - La clave de acceso es correcta.
        - El Id proporcionado corresponde con el existente en el certificado del usuario y clave privada que se encuentra en el keystore.
    - La aplicación cliente obtendrá el rol del usuario contenido en el certificado y proporcionará al usuario un menú ajustado a citado rol. Si el certificado contenido en el keystore pertenece a otra PKI en la que el rol no está especificado (dnie, por ejemplo), interrogará al usuario para que lo especifique.

2. <u>Consultar datos generales de un paciente.</u>

Esta opción es común a todos los actores (paciente, médico, gestor).

1 El actor(A) ejecuta el procedimiento 1 ***Solicita(M)*** con:

M=ConsultaDatosPersonales, IdPaciente

Si el actor tiene rol paciente entonces IdPaciente es conocido por la aplicación cliente.

Si el actor tiene rol medico o gestor entonces IdPaciente lo introduce directamente o partiendo de un listado de pacientes.

2 A Obtiene M' como resultado de ejecutar el paso anterior, con:

$M'=P_{servidor}$ [K], T, S_{actor} [H(M,N)], S_{actor} [H(T)], E_K (M,N), $Cert_{actor}$.

3 A Envía M' al servidor(S).

4 S recibe M' y ejecuta el procedimiento 2 ***RecibeSolicitud(M').***

5 S obtiene M" como resultado del paso anterior, con:

M"= (M, Id_{Cert}, rol_{Cert}).

6 S obtiene de M que la operación solicitada es: ConsultaDatosPersonales del paciente IdPaciente.

7 S determina si: (IdPaciente=Id_{Cert} AND rol_{Cert}= paciente)?

7.1 Obtiene DP= datos personales.

7.2 Obtiene H= historial médico del paciente.

7.3 Obtiene (E_K(L), $S_{servidor}$ [H(E_K(L))], $P_{servidor}$ [K])=ListaVisitasProtegida asociada a H.

7.4 Verifica: H(E_K(L))=$P_{servidor}$ [$S_{servidor}$ [H(E_K(L))]]? (firma sobre el Hash de L).

7.5 Obtiene K=$S_{servidor}$ [$P_{servidor}$ [K]].

7.6 Obtiene L=D_K (E_K(L)).

7.7 Crea M'"=DP + H + L

7.8 Invoca el procedimiento 3 ***EnviaRespuesta(M''').***

7.9 Obtiene del paso anterior M^{IV}.

7.10 Envía a A M^{IV}.

7.11 Pasa a 9.

8 S determina si: (IdPaciente!=Id_{Cert} AND (rol_{Cert}= gestor OR rol_{Cert}= medico))?

8.1 Si cierto:

8.1.1 Obtiene DP= datos personales.

8.1.2 Obtiene H= historial médico del paciente.

8.1.3 Crea M'"=DP + H

8.1.4 Si rol_{Cert}= medico

8.1.4.1 Obtiene ($E_{K'}(L')$, $S_{servidor}[H(E_{K'}(L'))]$, $P_{servidor}[K']$)= ListaMedicosProtegida asociada a IdPaciente.

8.1.4.2 Verifica: $H(E_{K'}(L'))=P_{servidor}[S_{servidor}[H(E_{K'}(L'))]]$? (firma sobre el Hash de L').

8.1.4.3 Obtiene $K'=S_{servidor}[P_{servidor}[K']]$.

8.1.4.4 Obtiene $L'=D_{K'}(E_{K'}(L'))$.

8.1.4.5 Si Id_{Cert} pertenece a L'.

8.1.4.5.1 Obtiene ($E_K(L)$, $S_{servidor}[H(E_K(L))]$, $P_{servidor}[K]$)= ListaVisitasProtegida asociada a H.

8.1.4.5.2 Verifica: $H(E_K(L))=P_{servidor}[S_{servidor}[H(E_K(L))]]$? (firma sobre el Hash de L).

8.1.4.5.3 Obtiene $K=S_{servidor}[P_{servidor}[K]]$.

8.1.4.5.4 Obtiene $L=D_K(E_K(L))$.

8.1.4.5.5 Añade a M'"=DP + H + L'

8.1.5 Invoca el procedimiento 3 ***EnviaRespuesta(M''').***

8.1.6 Obtiene del paso anterior M^{IV}.

8.1.7 Envía a A M^{IV}.

8.1.8 Pasa a 9.

8.2 Si falso, devuelve error.

9 A recibe M^{IV}. y ejecuta el procedimiento 4 ***RecibeSolicitud(M^{IV}).***

10 A obtiene M'" del paso anterior.

11 A presenta al usuario la información contenida en M'".

3. <u>Consultar una visita de un paciente.</u>

Esta opción es común a paciente y médico autorizado (asignado al paciente). Previamente el actor conoce el Id de la Visita a consultar y el IdPaciente.

1 El actor(A) ejecuta el procedimiento ***Solicita(M)*** con:

M=ConsultaVisita, IdPaciente, IdVisita.

Si el actor tiene rol paciente entonces IdPaciente es conocido por la aplicación cliente. IdVisita lo obtiene de la consulta de su historial.

Si el actor tiene rol medico entonces IdPaciente , IdVisita lo introduce directamente o partiendo de un listado de consulta previo.

2 A Obtiene M' como resultado de ejecutar el paso anterior, con:

$M'=P_{servidor}$ [K], T, S_{actor} [H(M,N)], S_{actor} [H(T)], E_K (M,N), $Cert_{actor}$.

3 A Envía M' al servidor(S).

4 S recibe M' y ejecuta el procedimiento 2 ***RecibeSolicitud(M').***

5 S obtiene M" como resultado del paso anterior, con:

M"= (M, Id_{Cert}, rol_{Cert}).

6 S obtiene de M que la operación solicitada es: ConsultaVisita IdVisita del paciente IdPaciente.

7 S determina:

7.1 (IdPaciente=Id_{Cert})?

7.1.1 Obtiene H= historial médico del paciente.

7.1.2 Obtiene (E_K(L), $S_{servidor}$ [H(E_K(L))], $P_{servidor}$ [K])=ListaVisitasProtegida asociada a H.

7.1.3 Verifica: H(E_K(L))=$P_{servidor}$ [$S_{servidor}$ [H(E_K(L))]]? (firma sobre el Hash de L).

7.1.4 Obtiene K=$S_{servidor}$ [$P_{servidor}$ [K]].

7.1.5 Obtiene L=D_K (E_K(L)).

7.1.6 (IdVisita) pertenece a L?

7.1.6.1 Si cierto, obtiene la V=Visita con Id=IdVisita.

7.1.6.1.1 Crea M'"=V

7.1.6.2 Si falso, devuelve error.

7.1.7 Invoca el procedimiento 3 ***EnviaRespuesta(M''').***

7.1.8 Obtiene del paso anterior M^{IV}.

7.1.9 Envía a A M^{IV}.

7.1.10 Pasa a 8.

7.2 (rol_{Cert}= medico)?

7.2.1 Obtiene H= historial médico del paciente.

7.2.2 Obtiene ($E_{K'}(L')$, $S_{servidor}[H(E_{K'}(L'))]$, $P_{servidor}[K']$)= ListaMedicosProtegida asociada a IdPaciente.

7.2.3 Verifica: $H(E_{K'}(L'))=P_{servidor}[S_{servidor}[H(E_{K'}(L'))]]$? (firma sobre el Hash de L').

7.2.4 Obtiene $K'=S_{servidor}[P_{servidor}[K']]$.

7.2.5 Obtiene $L'=D_{K'}(E_{K'}(L'))$.

7.2.5.1 Id_{Cert} pertenece a L?

7.2.5.1.1 Si cierto, ($E_K(L)$, $S_{servidor}[H(E_K(L))]$, $P_{servidor}[K]$)= ListaVisitasProtegida asociada a H.

7.2.5.1.1.1 Verifica: $H(E_K(L))=P_{servidor}[S_{servidor}[H(E_K(L))]]$? (firma sobre el Hash de L).

7.2.5.1.1.2 Obtiene $K=S_{servidor}[P_{servidor}[K]]$.

7.2.5.1.1.3 Obtiene $L=D_K(E_K(L))$.

7.2.5.1.1.3.1 (IdVisita) pertenece a L?

7.2.5.1.1.3.1.1 Si cierto, obtiene la V=Visita con Id=IdVisita.

7.2.5.1.1.3.1.1.1 Crea M'''=V.

7.2.5.1.1.3.1.2 Si falso, devuelve error.

7.2.5.1.2 Si falso, devuelve error.

7.2.6 Invoca el procedimiento 3 ***EnviaRespuesta(M''').***

7.2.7 Obtiene del paso anterior M^{IV}.

7.2.8 Envía a A M^{IV}.

7.3 Resto de casos, devuelve error.

8 A recibe M^{IV}. y ejecuta el procedimiento 4 ***RecibeSolicitud(M^{IV}).***

9 A obtiene M''' del paso anterior.

10 A presenta al usuario la información contenida en M'''.

4. <u>Consultar lista de pacientes.</u>

Esta opción es realizada por un médico o un gestor.

1 El actor(A) ejecuta el procedimiento ***Solicita(M)*** con:

M=ConsultaPacientes, IdMedico.

2 A Obtiene M' como resultado de ejecutar el paso anterior, con:

$M'=P_{servidor}$ [K], T, S_{actor} [H(M,N)], S_{actor} [H(T)], E_K (M,N), $Cert_{actor}$.

3 A Envía M' al servidor(S).

4 S recibe M' y ejecuta el procedimiento 2 ***RecibeSolicitud(M').***

5 S obtiene M" como resultado del paso anterior, con:

$M"= (M, Id_{Cert}, rol_{Cert})$.

6 S obtiene de M que la operación solicitada es: ConsultaPacientes.

7 S determina:

7.1 (rol_{Cert}= medico OR rol_{Cert}= gestor)?

7.1.1 Obtiene (E_K(L), $S_{servidor}$ [H(E_K(L))], $P_{servidor}$ [K])= ListaPacientesProtegida asociada a IdMedico.

7.1.2 Verifica: H(E_K(L))=$P_{servidor}$ [$S_{servidor}$ [H(E_K(L))]]? (firma sobre el Hash de L).

7.1.3 Obtiene K=$S_{servidor}$ [$P_{servidor}$ [K]].

7.1.4 Obtiene L=D_K (E_K(L)).

7.1.5 Para cada IdPaciente contenido en L.

7.1.5.1 Obtener Datos Personales del Paciente.

7.1.5.2 Añade a M'"=DP.

7.1.6 Invoca el procedimiento 3 ***EnviaRespuesta(M''').***

7.1.7 Obtiene del paso anterior M^{IV}.

7.1.8 Envía a A M^{IV}.

7.2 Resto de casos, devuelve error.

8 A recibe M^{IV}. y ejecuta el procedimiento 4 ***RecibeSolicitud(M^{IV}).***

9 A obtiene M'" del paso anterior.

10 A presenta al usuario la información contenida en M'".

5. <u>Añadir Visita a Paciente.</u>

Esta opción es realizada por un médico.

1 El actor(A) ejecuta el procedimiento ***Solicita(M)*** con:

M=AñadirVisita, IdPaciente, Visita.

El IdPaciente lo obtiene el médico de un listado o lo introduce directamente.

2 A Obtiene M' como resultado de ejecutar el paso anterior, con:

$M'=P_{servidor}[K], T, S_{actor}[H(M,N)], S_{actor}[H(T)], E_K(M,N), Cert_{actor}$.

3 A Envía M' al servidor(S).

4 S recibe M' y ejecuta el procedimiento 2 ***RecibeSolicitud(M').***

5 S obtiene M" como resultado del paso anterior, con:

$M"= (M, Id_{Cert}, rol_{Cert})$.

6 S obtiene de M que la operación solicitada es: AñadirVisita V al paciente IdPaciente.

7 S determina:

7.1 (rol_{Cert}= medico)?

7.1.1 Obtiene ($E_K(L)$, $S_{servidor}[H(E_K(L))]$, $P_{servidor}[K]$)= ListaPacientesProtegida asociada a IdMedico.

7.1.2 Verifica: $H(E_K(L))=P_{servidor}[S_{servidor}[H(E_K(L))]]$? (firma sobre el Hash de L).

7.1.3 Obtiene $K=S_{servidor}[P_{servidor}[K]]$.

7.1.4 Obtiene $L=D_K(E_K(L))$. L=ListaPacientes perteneciente a Id_{Cert}.

7.1.5 (IdPaciente) pertenece a L?

7.1.5.1 Si cierto, entonces:

7.1.5.1.1 Obtiene ($E_{K'}(L')$, $S_{servidor}[H(E_{K'}(L'))]$, $P_{servidor}[K']$)= ListaVisitasProtegida asociada a IdPaciente.

7.1.5.1.2 Verifica: $H(E_{K'}(L'))=P_{servidor}[S_{servidor}[H(E_{K'}(L'))]]$? (firma sobre el Hash de L').

7.1.5.1.3 Obtiene $K'=S_{servidor}[P_{servidor}[K']]$.

7.1.5.1.4 Obtiene $L'=D_{K'}(E_{K'}(L'))$. Obtiene L'=ListaVisitas perteneciente a IdPaciente.

7.1.5.1.5 Añade IdVisita de V a L'.

7.1.5.1.6 Guarda V en BD.

7.1.5.1.7 Obtiene una clave K" de cifra simétrica.

7.1.5.1.8 Cifra L'; $E_{K"}(L')$.

7.1.5.1.9 Obtiene hash; $H(E_{K"}(L'))$.

7.1.5.1.10 Firma el hash; $S_{servidor}[H(E_{K"}(L'))]$.

7.1.5.1.11 Cifra K"; $P_{servidor}$ [K"]

7.1.5.1.12 Guarda en BD ($E_{K"}$(L'), $S_{servidor}$ [H($E_{K"}$(L'))], $P_{servidor}$ [K"])= ListaVisitasProtegida asociada a IdPaciente.

7.1.5.1.13 Crea M"'= OK .

7.1.5.1.14 Invoca el procedimiento 3 ***EnviaRespuesta(M''').***

7.1.5.1.15 Obtiene del paso anterior M^{IV}.

7.1.5.1.16 Envía a A M^{IV}.

7.1.5.2 Si falso, devuelve error.

7.2 Resto de casos, devuelve error.

8 A recibe M^{IV}. y ejecuta el procedimiento 4 ***RecibeSolicitud(M^{IV}).***

9 A obtiene M"' del paso anterior.

10 A presenta al usuario la información contenida en M"' (visita añadida a historial correctamente).

6. <u>Asignar Médico a Paciente.</u>

Esta opción es realizada por un gestor.

1 El actor(A) ejecuta el procedimiento ***Solicita(M)*** con:

M=AsignarMedicoPaciente, IdPaciente, IdMedico.

El IdPaciente, IdMedico lo obtiene el gestor de un listado o lo introduce directamente.

2 A Obtiene M' como resultado de ejecutar el paso anterior, con:

$M'=P_{servidor}$ [K], T, S_{actor} [H(M,N)], S_{actor} [H(T)], E_K (M,N), $Cert_{actor}$.

3 A Envía M' al servidor(S).

4 S recibe M' y ejecuta el procedimiento 2 ***RecibeSolicitud(M').***

5 S obtiene M" como resultado del paso anterior, con:

$M"= (M, Id_{Cert}, rol_{Cert})$.

6 S obtiene de M que la operación solicitada es: Asignar IdPaciente al IdMedico.

7 S determina:

7.1 (rol_{Cert}= gestor)?

7.1.1 Obtiene ($E_K(L)$, $S_{servidor}$ [$H(E_K(L))$], $P_{servidor}$ [K])= ListaPacientesProtegida asociada a IdMedico.

7.1.2 Verifica: $H(E_K(L))=P_{servidor}$ [$S_{servidor}$ [$H(E_K(L))$]]? (firma sobre el Hash de L).

7.1.3 Obtiene $K=S_{servidor}$ [$P_{servidor}$ [K]].

7.1.4 Obtiene $L=D_K$ ($E_K(L)$). L=ListaPacientes perteneciente a Id_{Cert}.

7.1.5 (IdPaciente) pertenece a L?

7.1.5.1 Si falso, entonces:

7.1.5.1.1 Añade IdPaciente a L.

7.1.5.1.2 Obtiene una clave K' de cifra simétrica.

7.1.5.1.3 Cifra L; $E_{K'}(L)$.

7.1.5.1.4 Obtiene hash; $H(E_{K'}(L))$.

7.1.5.1.5 Firma el hash; $S_{servidor}$ [$H(E_{K'}(L))$].

7.1.5.1.6 Cifra K'; $P_{servidor}$ [K']

7.1.5.1.7 Guarda en BD ($E_{K'}(L)$, $S_{servidor}$ [$H(E_{K'}(L))$], $P_{servidor}$ [K'])= ListaPacientesProtegida asociada a IdMedico.

7.1.5.1.8 Obtiene ($E_{K"}(L')$, $S_{servidor}$ [$H(E_{K"}(L'))$], $P_{servidor}$ [K"])= ListaMedicosProtegida asociada a IdPaciente.

7.1.5.1.9 Verifica: $H(E_{K"}(L'))=P_{servidor}$ [$S_{servidor}$ [$H(E_{K"}(L'))$]]? (firma sobre el Hash de L).

7.1.5.1.10 Obtiene $K"=S_{servidor}$ [$P_{servidor}$ [K"]].

7.1.5.1.11 Obtiene L'=$D_{K''}$ ($E_{K''}$(L')). L'=ListaMedicos perteneciente a IdPaciente.

7.1.5.1.12 Añade IdMedico a L'.

7.1.5.1.13 Obtiene una clave K''' de cifra simétrica.

7.1.5.1.14 Cifra L'; $E_{K'''}$(L').

7.1.5.1.15 Obtiene hash; H($E_{K'''}$(L')).

7.1.5.1.16 Firma el hash; $S_{servidor}$[H($E_{K'''}$(L'))].

7.1.5.1.17 Cifra K'''; $P_{servidor}$[K''']

7.1.5.1.18 Guarda en BD ($E_{K'''}$(L'), $S_{servidor}$[H($E_{K'''}$(L'))], $P_{servidor}$ [K'''])= ListaMedicosProtegida asociada a IdPaciente.

7.1.5.1.19 Crea M'''= OK .

7.1.5.1.20 Invoca el procedimiento 3 ***EnviaRespuesta(M''').***

7.1.5.1.21 Obtiene del paso anterior M^{IV}.

7.1.5.1.22 Envía a A M^{IV}.

7.1.5.2 Si cierto, devuelve IdPaciente ya asignado a IdMedico .

7.2 Resto de casos, devuelve error.

8 A recibe M^{IV}. y ejecuta el procedimiento 4 ***RecibeSolicitud(M^{IV}).***

9 A obtiene M''' del paso anterior.

10 A presenta al usuario la información contenida en M''' (IdPaciente asignado a IdMedico).

7. <u>Eliminar asignación Médico a Paciente.</u>

Esta opción es realizada por un gestor.

1 El actor(A) ejecuta el procedimiento ***Solicita(M)*** con:

M=Des_asignarMedicoPaciente, IdPaciente, IdMedico.

El IdPaciente, IdMedico lo obtiene el gestor de un listado o lo introduce directamente.

2 A Obtiene M' como resultado de ejecutar el paso anterior, con:

$M'=P_{servidor}$ [K], T, S_{actor} [H(M,N)], S_{actor} [H(T)], E_K (M,N), $Cert_{actor}$.

3 A Envía M' al servidor(S).

4 S recibe M' y ejecuta el procedimiento 2 ***RecibeSolicitud(M').***

5 S obtiene M" como resultado del paso anterior, con:

M"= (M, Id_{Cert}, rol_{Cert}).

6 S obtiene de M que la operación solicitada es: Des_Asignar IdPaciente al IdMedico.

7 S determina:

7.1 (rol_{Cert}= gestor)?

7.1.1 Obtiene (E_K(L), $S_{servidor}$ [H(E_K(L))], $P_{servidor}$ [K])= ListaPacientesProtegida asociada a IdMedico.

7.1.2 Verifica: H(E_K(L))=$P_{servidor}$ [$S_{servidor}$ [H(E_K(L))]]? (firma sobre el Hash de L).

7.1.3 Obtiene K=$S_{servidor}$ [$P_{servidor}$ [K]].

7.1.4 Obtiene L=D_K (E_K(L)). L=ListaPacientes perteneciente a Id_{Cert}.

7.1.5 (IdPaciente) pertenece a L?

7.1.5.1 Si cierto, entonces:

7.1.5.1.1 Elimina IdPaciente de L.

7.1.5.1.2 Obtiene una clave K' de cifra simétrica.

7.1.5.1.3 Cifra L; $E_{K'}$(L).

7.1.5.1.4 Obtiene hash; H($E_{K'}$(L)).

7.1.5.1.5 Firma el hash; $S_{servidor}$ [H($E_{K'}$(L))].

7.1.5.1.6 Cifra K'; $P_{servidor}$ [K']

7.1.5.1.7 Guarda en BD ($E_{K'}$(L), $S_{servidor}$ [H($E_{K'}$(L))], $P_{servidor}$ [K'])= ListaPacientesProtegida asociada a IdMedico.

7.1.5.1.8 Obtiene ($E_{K''}$(L'), $S_{servidor}$ [H($E_{K''}$(L'))], $P_{servidor}$ [K"])= ListaMedicosProtegida asociada a IdPaciente.

7.1.5.1.9 Verifica: H($E_{K''}$(L'))=$P_{servidor}$ [$S_{servidor}$ [H($E_{K''}$(L'))]]? (firma sobre el Hash de L).

7.1.5.1.10 Obtiene K"=$S_{servidor}[P_{servidor}[K"]]$.

7.1.5.1.11 Obtiene L'=$D_{K"}(E_{K"}(L'))$. L'=ListaMedicos perteneciente a IdPaciente.

7.1.5.1.12 Elimina IdMedico de L'.

7.1.5.1.13 Obtiene una clave K"' de cifra simétrica.

7.1.5.1.14 Cifra L'; $E_{K"'}(L')$.

7.1.5.1.15 Obtiene hash; $H(E_{K"'}(L'))$.

7.1.5.1.16 Firma el hash; $S_{servidor}[H(E_{K"'}(L'))]$.

7.1.5.1.17 Cifra K"'; $P_{servidor}[K"']$

7.1.5.1.18 Guarda en BD ($E_{K"'}(L')$, $S_{servidor}[H(E_{K"'}(L'))]$, $P_{servidor}[K"']$)= ListaMedicosProtegida asociada a IdPaciente.

7.1.5.1.19 Crea M"'= OK .

7.1.5.1.20 Invoca el procedimiento 3 ***EnviaRespuesta(M''').***

7.1.5.1.21 Obtiene del paso anterior M^{IV}.

7.1.5.1.22 Envía a A M^{IV}.

7.1.5.2 Si falso, devuelve IdPaciente no asignado a IdMedico .

7.2 Resto de casos, devuelve error.

8 A recibe M^{IV}. y ejecuta el procedimiento 4 ***RecibeSolicitud(M^{IV}).***

9 A obtiene M"' del paso anterior.

10 A presenta al usuario la información contenida en M"' (IdPaciente des_asignado a IdMedico).

8. Resto de protocolos correspondientes a Casos de Uso prioridad 3.

No se especificarán los protocolos de los restantes Casos de Uso, por seguir un esquema similar a los ya definidos.

Capítulo 5

5.-Representación de los datos. XML.

XML (*Extensible Markup Language)* es una especificación, recomendada por el World Wide Web Consortium (W3C), que permite la creación de lenguajes de marcas. Es por tanto un metalenguaje, extensible, que facilita a sus usuarios la definición de sus propios elementos.

Se trata de un subconjunto del Standard Generalized Markup Language (SGML), y ha dado lugar, mediante la adición de las oportunas reglas semánticas, a lenguajes como XHTML, MathML, MusicXML, etc. Está diseñado para ser fácilmente inteligible por las personas.

Su uso principal hoy en día es el intercambio de información estructurada entre sistemas.

Los documentos en XML tienen una estructura jerárquica, con un único elemento raíz, que incluye a todos los demás. Cada elemento queda contenido entre etiquetas, y las etiquetas deben estar correctamente anidadas, sin solapamientos. A su vez cada elemento puede tener una serie de atributos.

Un documento XML está formado por el prólogo y por el cuerpo del documento.

Prólogo: No es obligatorio. Contiene información relativa a la versión xml, tipo de documento, etc.

Cuerpo: Es obligatorio, y debe contener un y solo un elemento raíz.

Elementos

Los elementos XML pueden tener contenido (más elementos, caracteres o ambos), o bien ser elementos vacíos.

Atributos

Definen características o propiedades de los elementos a los que pertenecen. Siempre van entrecomillados.

Comentarios

Información adicional que no es tratada por el procesador.

Cualidades de los documentos XML:

- Bien formado: Un documento que cumple con las reglas de sintaxis XML. Un documento que no está bien formado no es un documento XML, ni puede ser aceptado por un parser.

- Válido: Documento bien formado y que además cumple con una serie de reglas semánticas definidas en un XMLSchema o un DTD.

5.1.-Estándares XML.

Como se comentaba más arriba, de la especificación XML han surgido otras especificaciones que utilizan XML como substrato.

Parte de las que se utilizarán en este PFC componen el conjunto de especificaciones WS-* que se muestra a continuación:

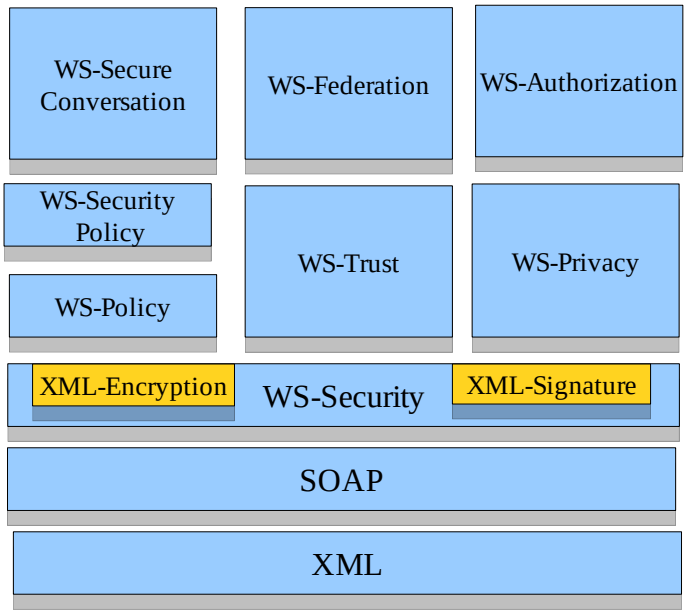


*Ilustración 15: Pila WS-**

En el anexo VII se puede consultar la sintaxis de parte de los protocolos que componen WS-*.

SOAP (Simple Object Access Protocol) es un protocolo que permite el intercambio de mensajes basados en XML sobre redes de ordenadores. Como se aprecia en la pila WS-*, es la base en la que se fundamentan los Web Services.

Las ventajas que ofrece SOAP son:

- Permite la utilización de diferentes protocolos de transporte, aunque el más utilizado es HTTP.
- Independiente de la plataforma.
- Independiente del lenguaje utilizado.
- Simple y extensible.

Desventajas de este protocolo frente a otros como CORBA[26]:

- Poco eficiente, pues es un lenguaje pensado para ser entendido fácilmente por las personas, lo que implica la utilización de mucha información no estrictamente necesaria para el procesamiento por máquinas de los mensajes.
- Roles fijos, al utilizar HTTP; una parte es cliente y la otra servidora.

XMLSignature es una recomendación del W3C que define la sintaxis en XML para contener firmas digitales. Otras definiciones de este estándar son: *XMLDsig*, *XML-DSig*, *XML-Sig*.

XMLSignature se puede utilizar para firma digital de cualquier tipo de contenido. Todo lo que es accesible vía una URL puede ser objeto de firma mediante XMLSignature.

Existen básicamente tres posibilidades; firmar un recurso que se encuentra fuera del documento que contiene la firma (**detached signature**), que se firme un recurso contenido en el documento que a su vez contiene la firma (**enveloped signature**) y firmar un recurso que es contenido en la propia firma (**enveloping signature**). En este PFC se utilizará firma en formato enveloped signature , como se verá en el ejemplo que sigue más abajo.

XMLEncryption es otra recomendación del W3C que define la sintaxis en XML para contener elementos cifrados y proporcionar la información suficiente para su descifrado.

WS-Security (Web Services Security) es un protocolo de comunicaciones que proporciona ciertos requerimientos de seguridad en Web Services. Es un estándar del consorcio OASIS, que promueve la adopción de estándares abiertos en la sociedad de la información. La versión actual del estándar es la 1.1 de febrero de 2.006.

Define la sintaxis para incorporar en la cabecera de los mensajes SOAP los contenidos apropiados para proporcionar integridad y confidencialidad a las partes del mensaje que se especifiquen.

WS-Policy, WS-SecurityPolicy, WS-SecureConversation. En principio no se utilizarán estos protocolos, aunque sería interesante, en trabajos posteriores, emplearlos.

WS-Policy permite identificar en el fichero descriptor del WS los requerimientos de policía de acceso al servicio. WS-SecurityPolicy identifica en el fichero WSDL exactamente los requerimientos de policía relacionados con la seguridad. De esta manera, cualquiera que acceda al fichero WSDL conocerá qué exige el WS para acceder a sus servicios. Estos protocolos facilitan además la automatización de la generación de clientes y la interoperabilidad de los WS.

26 Common Object Request Broker Architecture

WS-SecureConversation es un protocolo que permite el establecimiento de conversaciones seguras a nivel aplicación. Utilizando este protocolo se gana en eficiencia, pues se utilizan claves simétricas de cifrado para sucesivos mensajes, evitándose así los pasos, salvo en el mensaje inicial, de:

- generación de claves simétricas.
- cifrado de la clave simétrica con la clave pública del receptor.
- descifrado de la clave simétrica por el receptor con su clave privada.

Es de destacar que los pasos anteriores son computacionalmente costosos, por lo que el ahorro que se consigue con la utilización de WS-SecureConversation es notable.

5.2.-Datos XML de la aplicación.

La información intercambiada entre cliente y servidor Web Service estará en formato XML. Para realizar el trabajo de conversión de objetos Javabean a XML y viceversa, en este PFC se utilizarán el framework ADB (Axis2 Databinding Framework) de Apache y el modelado AXIOM (AXIs Object Model) también uno de los proyectos de Apache.

AXIOM está diseñado para optimizar el uso de memoria y de proceso en el tratamiento de datos en formato XML. Utiliza un parser que atiende al esquema StAX (Streaming API for XML) por lo que, a diferencia de DOM (Document Object Model) el documento se va construyendo bajo demanda, ahorrándose recursos cuando no es necesario representarlo internamente de forma completa. Otra ventaja de AXIOM es que permite la utilización de MTOM (Message Transfer Optimization Mechanism) que facilita la transmisión de mensajes XML comprimidos.

ADB sabe como tratar y convertir en elementos XML los tipos primitivos de Java, los arrays de éstos, así como los JavaBean cuyos atributos sean de los tipos anteriores. Sin embargo, falla cuando los atributos son clases complejas, como pueden ser del tipo *Vector, HashMap,* etc. Como consecuencia de lo anterior, los JavaBean utilizados en el PFC utilizarán atributos que ADB sepa manejar.

En el anexo VIII se incluyen ejemplos de cada uno de los objetos procesados que serán transferidos entre servidor y cliente, una vez convertidos en documentos XML mediante las clases proporcionadas por ADB.

Además de los datos intercambiados en XML, también se guardarán en la base de datos elementos en este formato. En el apartado 7.2 se indican qué campos de la base de datos se ven afectados y cual es su contenido.

5.3.-DTD ó Schemas.

No se utilizan DTD o Schemas para validar los datos utilizados en formato xml.

En principio esto no debe suponer ningún problema de seguridad en cuanto a los xml generados por la parte servidora, controlados por el framework de Axis2 respecto de los xml que salen hacia el cliente, y por las clases del paquete de integración respecto de los datos en formato xml guardados en la base de datos.

Sí existe una vulnerabilidad por el hecho de que no se validan los datos en formato xml que se pasan como parámetro al servidor de WS. Esta circunstancia se comentará en el apartado de conclusiones del PFC.

Capítulo 6

6.-Protocolos de comunicación.

Web Services es un conjunto de protocolos y estándares utilizados para el intercambio de información entre aplicaciones que se comunican a través de redes. Con ellos se pretende conseguir un bajo acoplamiento, de manera que plataformas distintas y lenguajes de programación diferentes puedan ser interoperables.

Las organizaciones más activas en cuanto a la definición de especificaciones y estándares son OASIS y el W3C.

En relación a la interoperabilidad, la organización WS-I, se ha encargado de definir diversos perfiles que los Web Services deben cumplir para ser interoperables.

La base de los estándares utilizados por los WS son:

- XML(Extensible Markup Language): Es el formato estándar en el que se expresan los datos intercambiados.
- SOAP(Simple Object Access Protocol) comentado en el capítulo 5
- XML-RPC (XML Remote Procedure Call): Protocolo que implementa llamadas a procedimientos remotos sobre XML.
- WSDL (Web Services Description Languages): WSDL es una especificación del W3C para describir servicios web como un conjunto de operaciones que intercambian mensajes en forma de documentos o como llamadas a procedimientos.
- UDDI (Universal Description, Discovery and Integration): Protocolo que facilita la publicación y el descubrimiento de servicios Web. Es una especificación de OASIS.

6.1.-Framework Axis2.

Axis2 es el framework elegido para la implementación de los Web Services. En otros capítulos se ha hablado sobre las excelencias de este WS, por lo que directamente se pasará en este apartado a prepararlo para nuestra aplicación.

De acuerdo a los protocolos de seguridad establecidos en el capítulo 4, será Axis2 el responsable de:

- Creación del Timestamp que acompañará a los mensajes.
- Firmado del Timestamp y del cuerpo del mensaje.
- Creación de la clave simétrica para el trasporte.
- Cifrado con la clave simétrica el cuerpo de los mensajes.
- Cifrado para envío de la clave simétrica con la clave pública del destinatario.
- Incorporación de la clave simétrica cifrada al mensaje.
- Incorporación al mensaje del certificado del remitente.
- Descifrado de los mensajes, usando la clave privada del receptor para descifrar la clave simétrica.
- Comprobación de las firmas sobre el cuerpo del mensaje y el Timestamp.

El comportamiento de Axis2 queda definido en los ficheros de configuración de cliente (axis2.xml) y servidor (services.xml). En el anexo IX se ha incluido la configuración de ambos ficheros.

Las acciones anteriores tienen su reflejo en el fichero de configuración services.xml de axis2, tal y como se muestra a continuación.

Detalles que conviene explicar de la configuración escogida son:

- <items>Timestamp Signature Encrypt</items>
  Son las acciones que se realizan sobre los mensajes entrantes/salientes; en primer lugar se añade un timestamp, que incorpora una marca de tiempo que fija una franja de validez temporal; en segundo lugar se añade la firma de los elementos Timestamp y Body (según se indica en el elemento <signatureParts>) y finalmente se cifra el elemento Body.
- <encryptionKeyTransportAlgorithm>
  http://www.w3.org/2001/04/xmlenc#rsa-1_5
  </encryptionKeyTransportAlgorithm>
  El algoritmo usado para el transporte de la clave simétrica de cifrado es RSA 1.5.
- <encryptionSymAlgorithm>
  http://www.w3.org/2001/04/xmlenc#aes256-cbc
  </encryptionSymAlgorithm>
  El algoritmo simétrico utilizado para cifrar el elemento Body es AES256.
- <passwordCallbackClass>servidor.PWCBHandler</passwordCallbackClass>
  La clave de acceso a la parte privada de la clave RSA la proporciona esta clase.
- <encryptionUser>useReqSigCert</encryptionUser>
  Se utiliza el propio certificado enviado por el cliente en el mensaje para cifrar la respuesta dada por el servidor.

6.2.-Diseño (Iteración 1)

En la primera iteración se diseña y construye el WS (servidor y cliente) sin capa de presentación ni Base de Datos, que atiende a los requerimientos definidos en los casos de uso de prioridad 1 (ver apartado 2.3.2).

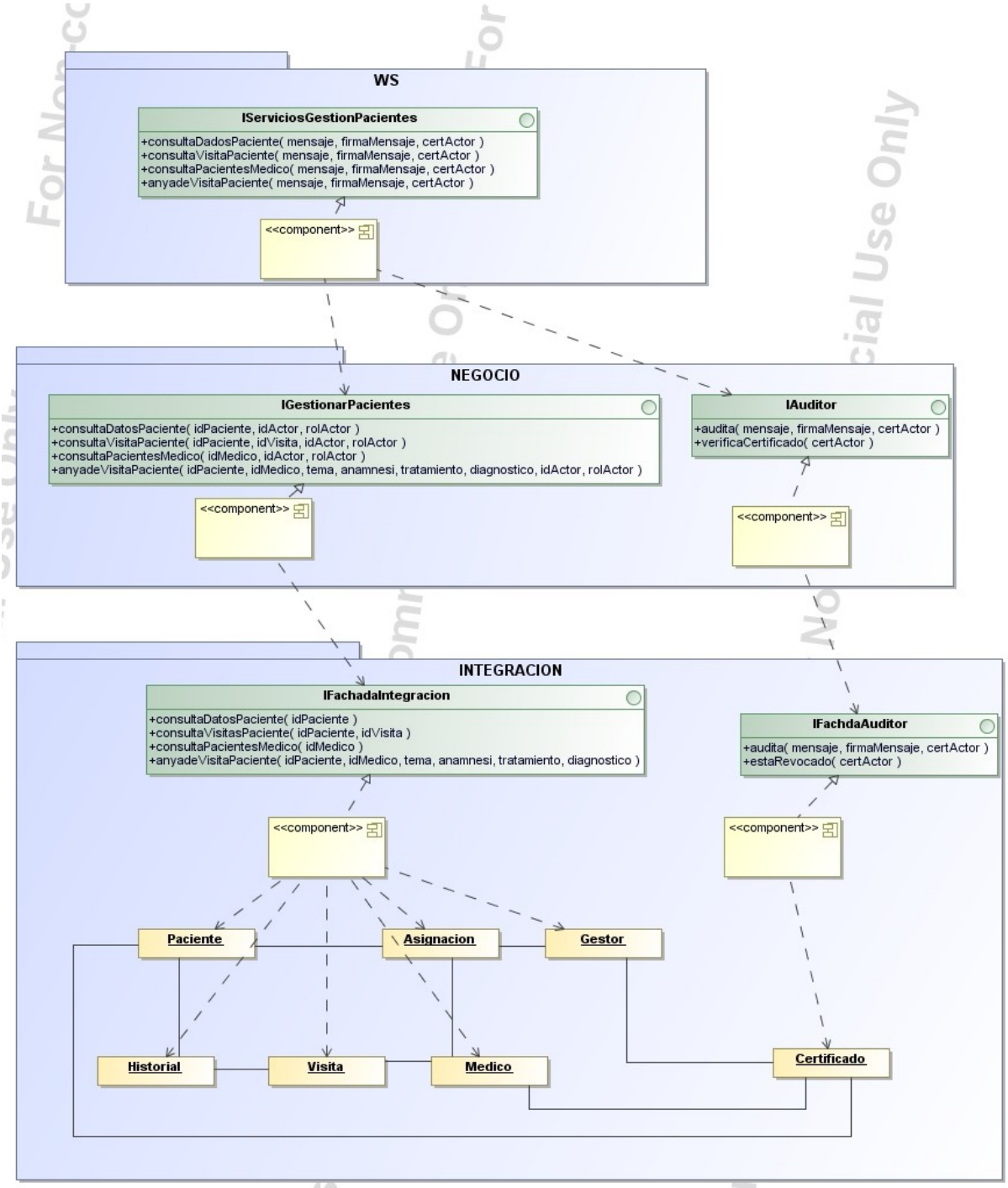


Ilustración 16: Diseño (1ª Iteración)

Descripción de los paquetes:

WS: Contiene las clases llamadas por el framework Axis2. Además de los parámetros de llamada de cada operación expuesta como Web Service, del contexto del mensaje (MessageContext) se puede extraer el mensaje completo una vez procesado por el módulo rampart (SOAPEnvelope) y un conjunto de objetos resultado del procesamiento de los elementos de seguridad de la cabecera del mensaje. De estos objetos son de interés para el procesamiento posterior:

-Certificado usado para la firma. De este certificado obtenemos el id y el rol del Actor.

-Timestamp generado por el cliente.

-Firma digital del mensaje y del timestamp. A efectos de auditoría.

El paquete WS utiliza las interfaces del paquete NEGOCIO.

Interfaces y Clases que lo integran:

1. IServiciosGestionPacientes: interfaz que define los métodos que implementará el WS en relación a la gestión realizada por pacientes y médicos.
2. ServiciosGestionPacientes: clase que implementa la interfaz anterior.
3. PWCBHandler: clase que gestiona el acceso a la clave utilizada en el almacén de claves y certificados.

NEGOCIO: Es llamado por el paquete WS; contiene la lógica propia del negocio e interacciona con la capa de integración. El interfaz de Auditoría muestra los métodos invocados desde WS para:

-Auditar las acciones de los actores.

-Verificar los certificados usados por los actores en cuanto a revocación, puesto que desde Axis2 sólo se valida su integridad y que no está caducado.

Clases que integran el paquete:

1. IGestorPacientes: interfaz que define los métodos que contienen la lógica de negocio de la gestión de los pacientes, vinculado a los actores médico y paciente.
2. GestorPacientes: implementa la interfaz anterior.
3. IAuditor: define los métodos con la lógica correspondiente a la auditoría de las acciones que los actores efectúan sobre la aplicación.
4. Auditor: clase que implementa la interfaz anterior.
5. PruebaGestorPacientes: clase de prueba que no utiliza el WS.

INTEGRACIÓN: Se encarga este paquete de interaccionar con la capa de persistencia (Base de Datos). En esta iteración no existe base de datos; se utilizará una clase que implementa el interfaz del paquete que crea en memoria un conjunto de objetos (Medico, Gestor, Paciente, Historial, Visita, etc.) a efectos de prueba.

Clases que lo integran:

1. IFachadaIntegración: define los métodos a implementar por las clases que accederán a la fuente de datos.
2. FachadaIntegracionTest1: implementa la clase anterior, creando los objetos en memoria. Clase para pruebas sin acceso a Base de Datos.
3. IFachadaAuditor: define los métodos a implementar por las clases que acceden a Base de Datos para guardar información relacionada con auditoría.

MODELO: Contiene las clases que definen los objetos del negocio. Se trata de VO (Value Object) que tienen los métodos get y set correspondientes para obtener y modificar los valores de los atributos de las clases.

En el cliente el diseño es más sencillo, en tanto que no se ha construido todavía la interfaz gráfica. Los paquetes utilizados en el cliente son:

CLIENTE: contiene las clases stub que llamarán al WS.

Las clases del paquete son:

1. PWCBHandler: clase que gestiona el acceso a la clave utilizada en el almacén de claves y certificados.
2. PacienteWSCliente: clase que contiene los métodos que invocan al servidor.
3. Prueba: clase de prueba del WS.

SERVIDOR.MODELO: paquete idéntico al contenido en el servidor con las clases del modelo de datos.

6.3.-Construcción y Pruebas (Iteración 1).

A continuación se especifican las clases construidas en esta primera iteración. Se muestra el árbol de los fuentes generado con el IDE eclipse.

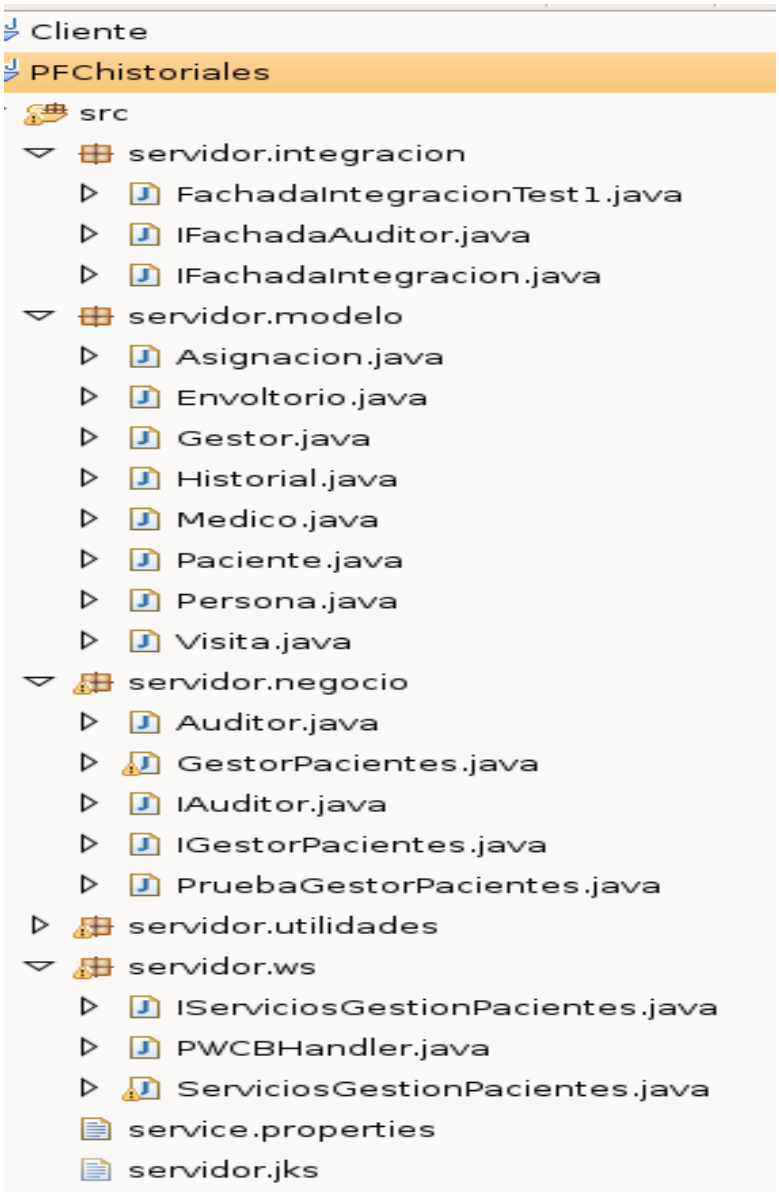


Ilustración 17: Clases servidor (1ª Iteración)

Por lo que respecta al cliente, las clases necesarias han sido:

Ilustración 18: Clases cliente (1ª Iteración)

Los casos de prueba para validación que se han definido para validar la aplicación, atendiendo a los requerimientos de la misma, son:

CASO	VALOR ESPERADO	RESULTADO
consultar paciente “0000000-P” actor: “0000000-P”; rol: “paciente”	se obtienen los datos personales del paciente	✔
consultar paciente “0000000-P” actor: “0000001-P”; rol: “paciente”	mensaje de error	✔
consultar paciente “0000000-P” actor: “0000000-M”; rol: “medico” paciente asignado al médico “0000000-M”	se obtienen los datos personales del paciente	✔
consultar paciente “0000000-P” actor: “0000001-M”; rol: “medico” paciente no asignado al médico “0000001-M”	mensaje de error	✔
consultar paciente “0000000-P” actor: “0000000-G”; rol: “gestor”	se obtienen los datos personales del paciente	✔
consultar visita “v1” paciente “0000000-P” actor: “0000000-P”; rol: “paciente”	se obtienen los datos de la visita	✔
consultar visita “v1” paciente “0000000-P” actor: “0000000-M”; rol: “medico” paciente asignado al médico “0000000-M”	se obtienen los datos de la visita	✔
consultar visita “v1” paciente “0000000-P” actor: “0000001-M”; rol: “medico” paciente no asignado al médico “0000001-M”	mensaje de error	✔
consultar visita “v1” paciente “0000000-P” actor: “0000000-G”; rol: “gestor”	mensaje de error	✔
consultar pacientes médico “0000000-M” actor: “0000000-M”; rol: “medico”	se obtienen listado de pacientes	✔
consultar pacientes médico “0000000-M” actor: “0000000-G”; rol: “gestor”	se obtienen listado de pacientes	✔
consultar pacientes médico “0000000-M” actor: “0000001-M”; rol: “medico”	mensaje de error	✔
añadir visita a paciente “0000000-P” actor: “0000000-M”; rol: “medico” paciente asignado al médico “0000000-M”	se añade la visita, se obtiene mensaje “ok”	✔
añadir visita a paciente “0000000-P” actor: “0000001-M”; rol: “medico” paciente no asignado al médico “0000001-M”	mensaje error	✔

Estos casos de prueba se ejecutan desde la capa cliente, con una clase específica creada al efecto, no utilizándose ningún framework de test (tipo Junit).

Capítulo 7

7.-Persistencia de los datos.

Los datos utilizados por la aplicación deben persistir a lo largo del tiempo. Para ello se utilizará un Sistema Gestor de Base de Datos que permita una custodia eficiente de los datos, con múltiples usuarios accediendo simultáneamente a la misma y que cumpla las propiedades ACID[27].

7.1.-MySQL.

El Sistema Gestor de Base de Datos elegido es MySQL, cuyas cualidades ya fueron mencionadas en el apartado 2.1 de este PFC.

Se utilizará el motor de almacenamiento InnoDB que soporta transacciones tipo ACID, bloqueo de registros e integridad referencial.

27**A**tomicity, **C**onsistency, **I**solation and **D**urability

7.2.-Modelado de datos.

Se mantienen las clases identificadas en los apartados anteriores. En el diseño que se aprecia en la ilustración más abajo, aparecen clases, relaciones y cardinalidades.

El historial es único para cada paciente, por lo que los atributos del mismo se podrían incluir en el paciente como atributos. No obstante se mantiene en una tabla diferente ante la posibilidad de que un paciente pudiese tener varios historiales (por ejemplo, uno diferente por centro hospitalario).

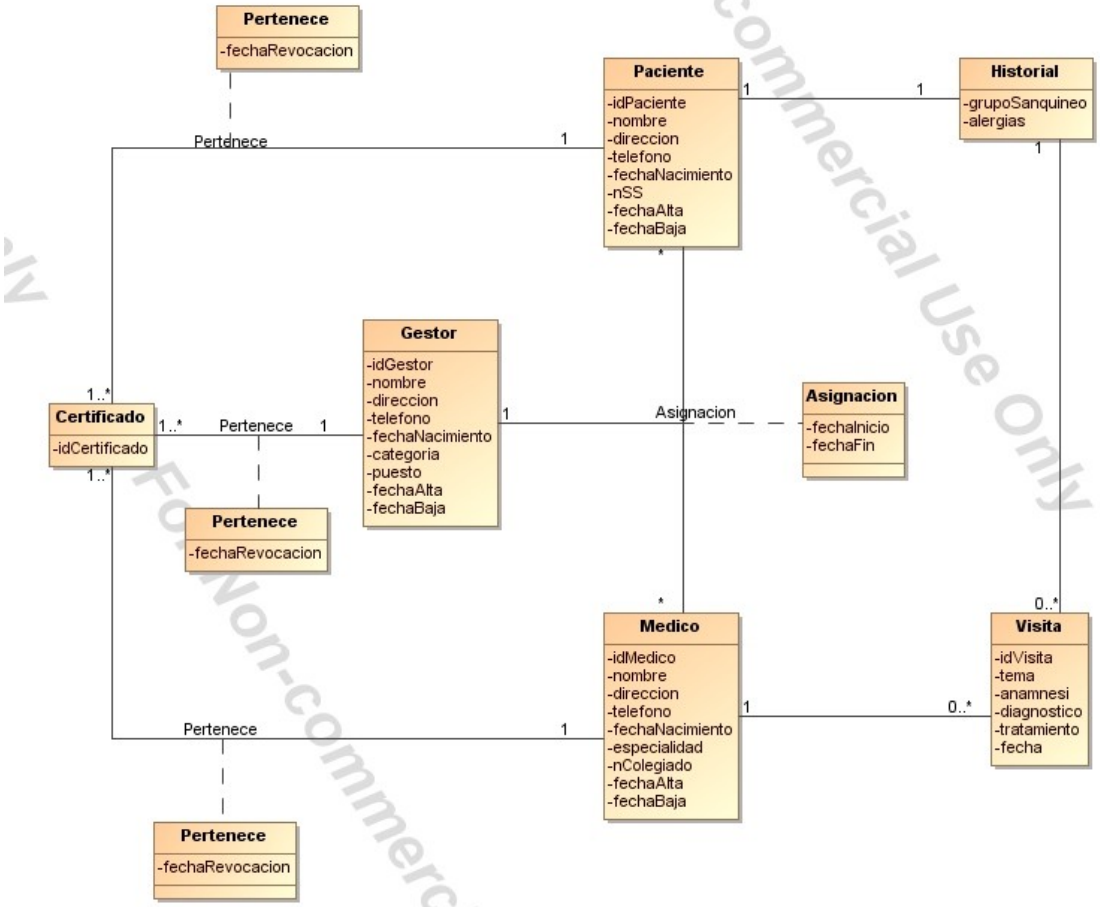


Ilustración 19: Diagrama clases para BD

Teniendo en cuenta el esquema de relaciones establecido en la ilustración anterior, el conjunto de tablas resultado es el siguiente:

Pacientes (idPaciente, nombre, direccion, telefono, fechaNacimiento, nSS, fechaAlta, fechaBaja).

Medicos (idMedico, nombre, direccion, telefono, fechaNacimiento, especialidad, nColegiado, fechaAlta, fechaBaja).

Gestores (idGestor, nombre, direccion, telefono, fechaNacimiento, categoria, puesto, fechaAlta, fechaBaja).

Historiales (idHistorial, grupoSanguineo).

Alergias (idHistorial, alergia).

Visitas (idVisita, idMedico, tema, anamnesi, diagnostico, tratamiento, fecha).

Certificados (idCertificado, certificado, idUsuario, fechaRevocacion).

VisitasHistorial (idPaciente, listaVisitasPaciente(*), firmaLista, claveSesion).

MedicosPaciente (idPaciente, listaMedicosPaciente(**), firmaLista, claveSesion).

PacientesMedico (idMedico, listaPacientesMedico(***), firmaLista, claveSesion).

(*) xml cifrado, compuesto por:

```
-<Historial>
        -<Paciente>idPaciente</Paciente>
        -<Visitas>
                -<Visita>idVisita</Visita>
                -...
        -</Visitas>
-<Historial>
```

(**) xml cifrado, compuesto por:

```
-<Paciente>
        -<idPaciente>idPaciente</idPaciente>
        -<Medicos>
                -<Medico>
                        -<Id>idMedico</Id>
                        -<FechaInicio/>
                        -<FechaFin/>
                        -<Asignador>idGestor</Asignador>
                -</Medico>
                -...
        -<Medicos>
-</Paciente>
```

(***) xml cifrado, compuesto por:

-<Medico>
-<idMedico>idMedico</idMedico>
-<Pacientes>
-<Paciente>
-<Id>idPaciente</Id>
-<FechaInicio/>
-<FechaFin/>
-<Asignador>idGestor</Asignador>
-</Paciente>
-...
-<Pacientes>
</Medico>

Como se puede apreciar en los xml que posteriormente son cifrados, se introduce en los mismos la clave principal de la tabla. Por ejemplo, en el listado de pacientes de un médico se incluye el idMedico, que corresponde con la clave de la entrada en la tabla.

Esto es así para evitar que un atacante que tenga acceso a la base de datos pueda visualizar uno de los listados cambiando simplemente el campo clave de la tabla. Así, si el médico con id XX es usuario de la aplicación y tiene acceso a la base de datos, si quisiese ver los pacientes del médico con id YY, sólo tendría que cambiar las entradas *listaPacientesMedico(***), firmaLista, claveSesion* correspondes a la clave YY en la tabla y situarlos con la clave propia XX. Este ataque se evita verificando, una vez validada la firma de la lista que el id de la clave del registro corresponde con la contenida en el xml.

Tablas resultantes y tipo de datos:

TABLA: Pacientes		
CAMPO	**TIPO DATO**	**DESCRIPCIÓN**
idPaciente	VARCHAR (10)	dni paciente
nombre	VARCHAR (30)	nombre y apellidos
direccion	VARCHAR (30)	dirección paciente
telefono	VARCHAR (13)	nº teléfono
fechaNacimiento	DATE	fecha de nacimiento
nSS	VARCHAR (18)	número de la Seguridad Social
fechaAlta	DATE	fecha de alta en el sistema
fechaBaja	DATE	fecha de baja en el sistema

TABLA: Medicos		
CAMPO	**TIPO DATO**	**DESCRIPCIÓN**
idPaciente	VARCHAR (10)	dni paciente
nombre	VARCHAR (30)	nombre y apellidos
direccion	VARCHAR (30)	dirección paciente
telefono	VARCHAR (13)	nº teléfono
fechaNacimiento	DATE	fecha de nacimiento
especialidad	VARCHAR (30)	especialidad del médico
nColegiado	VARCHAR (18)	nº de colegiado
fechaAlta	DATE	fecha de alta en el sistema
fechaBaja	DATE	fecha de baja en el sistema

TABLA: Gestores		
CAMPO	**TIPO DATO**	**DESCRIPCIÓN**
idPaciente	VARCHAR (10)	dni paciente
nombre	VARCHAR (30)	nombre y apellidos
direccion	VARCHAR (30)	dirección paciente
telefono	VARCHAR (13)	nº teléfono
fechaNacimiento	DATE	fecha de nacimiento
categoría	VARCHAR (30)	categoría del empleado
puesto	VARCHAR (30)	puesto de trabajo
fechaAlta	DATE	fecha de alta en el sistema
fechaBaja	DATE	fecha de baja en el sistema

TABLA: Historiales		
CAMPO	**TIPO DATO**	**DESCRIPCIÓN**
idHistorial	VARCHAR (10)	dni paciente
grupoSanguineo	CHAR (2)	grupo sanguíneo

TABLA: Alergias		
CAMPO	**TIPO DATO**	**DESCRIPCIÓN**
idHistorial	VARCHAR (10)	dni paciente
alergia	VARCHAR (30)	alergia

TABLA: Visitas		
CAMPO	**TIPO DATO**	**DESCRIPCIÓN**
idVisita	CHAR (28)	descriptor visita (hash)
idMedico	VARCHAR (10)	dni médico
tema	VARCHAR (50)	descripción visita
anamnesi	CLOB	anamnesi de la visita
diagnostico	CLOB	diagnostico de la visita
tratamiento	CLOB	tratamiento prescrito
fecha	DATE	fecha de la visita

TABLA: Certificados		
CAMPO	**TIPO DATO**	**DESCRIPCIÓN**
idCertificado	VARCHAR (20)	id del certificado (hash del certificado)
certificado	CLOB	certificado
idUsuario	VARCHAR (10)	id del usuario
fechaRevocacion	DATE	fecha de revocación

TABLA: VisitasHistorial		
CAMPO	**TIPO DATO**	**DESCRIPCIÓN**
idHistorial	VARCHAR (10)	id del Historial al que pertenecen las visitas
listaVisitasPaciente	BLOB	lista cifrada con clave sesión y firmada
firmaLista	BYTE (128)	firma de la lista
claveSesion	BYTE (128)	clave de sesión simétrica CIFRADA

TABLA: MedicosPaciente		
CAMPO	**TIPO DATO**	**DESCRIPCIÓN**
idPaciente	VARCHAR (10)	id del Paciente al que pertenecen las visitas
listaMedicosPaciente	BLOB	lista cifrada con clave sesión y firmada
firmaLista	BYTE (128)	firma de la lista
claveSesion	BYTE (128)	clave de sesión simétrica CIFRADA

TABLA: PacientesMedico		
CAMPO	**TIPO DATO**	**DESCRIPCIÓN**
idMedico	VARCHAR (10)	id del Historial al que pertenecen las visitas
listaPacientesMedico	BLOB	lista cifrada con clave sesión y firmada
firmaLista	BYTE (128)	firma de la lista
claveSesion	BYTE (128)	clave de sesión simétrica CIFRADA

7.3.-Diseño (Iteración 2).

Durante la segunda iteración se diseña la Base de Datos, según las tablas indicadas en el apartado anterior.

Respecto a la aplicación, se mantiene el diseño identificado en el apartado 6.2.

7.4.-Construcción y Pruebas (Iteración 2).

En esta fase se construye la base de datos, tal y como se especifica en el anexo III, en el que se encuentra el script de creación de la base de datos, usuario a utilizar y las correspondientes tablas.

En relación a la aplicación, en esta fase se construye y prueba la clase que implementa el interfaz IFachadaPacientes. Esta clase sustituirá a la clase construida en la iteración anterior que creaba y gestionaba objetos (pacientes, médicos, etc.) en memoria.

Además se crea una clase (BDConexion) que facilita la gestión de la base de datos, cargando el driver y facilitando la creación de conexiones y el cierre de las mismas.

En principio no se tiene pensado utilizar un DataSource que maneje las conexiones, por lo que cada consulta a la base de datos creará una nueva conexión. Si el número de usuarios fuese alto, sería conveniente manejar las conexiones con un DataSource.

El conjunto de casos de prueba para esta iteración es el mismo que los establecidos en el punto 6.3.

El resultado en la base de datos se puede observar con el programa MySQL Query Browser:

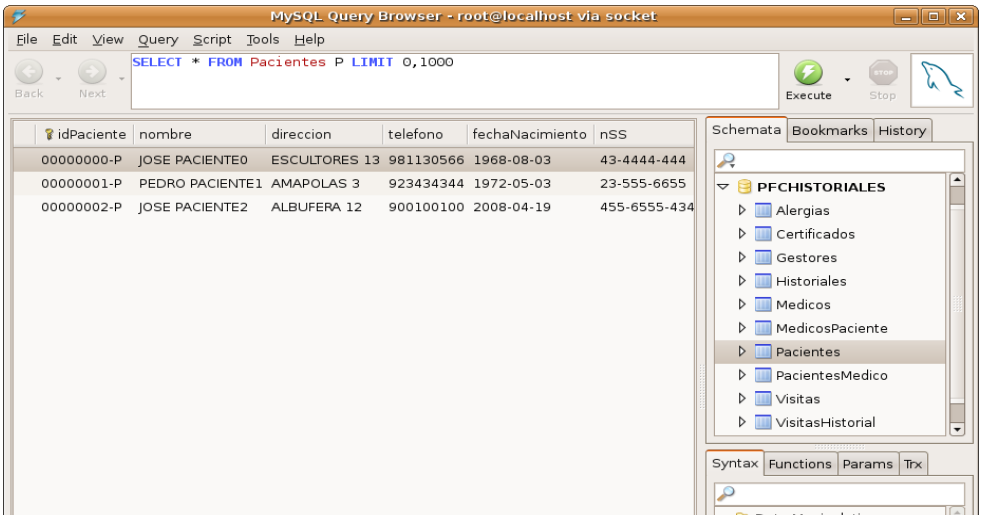


Ilustración 20: Resultados en la Base de Datos de las pruebas

En cuanto a los datos guardados en campos BLOB/CLOB vemos que en la base de datos quedan reflejados tal cual:

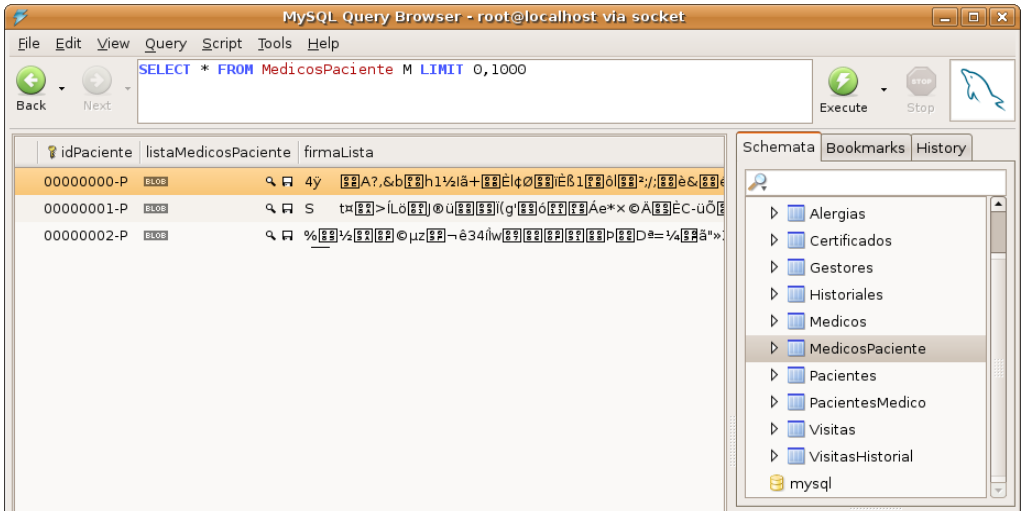


Ilustración 21: Resultados (2) de la pruebas sobre la BD

7.5.-Diseño (Iteración 3).

En esta fase se diseña el módulo correspondiente a la gestión que hacen los actores administrativos (gestores). Corresponde con los casos de uso de prioridad 2 definidos en el apartado 2.3.2.

A continuación se muestra la parte diseñada, en el marco de la aplicación completa:

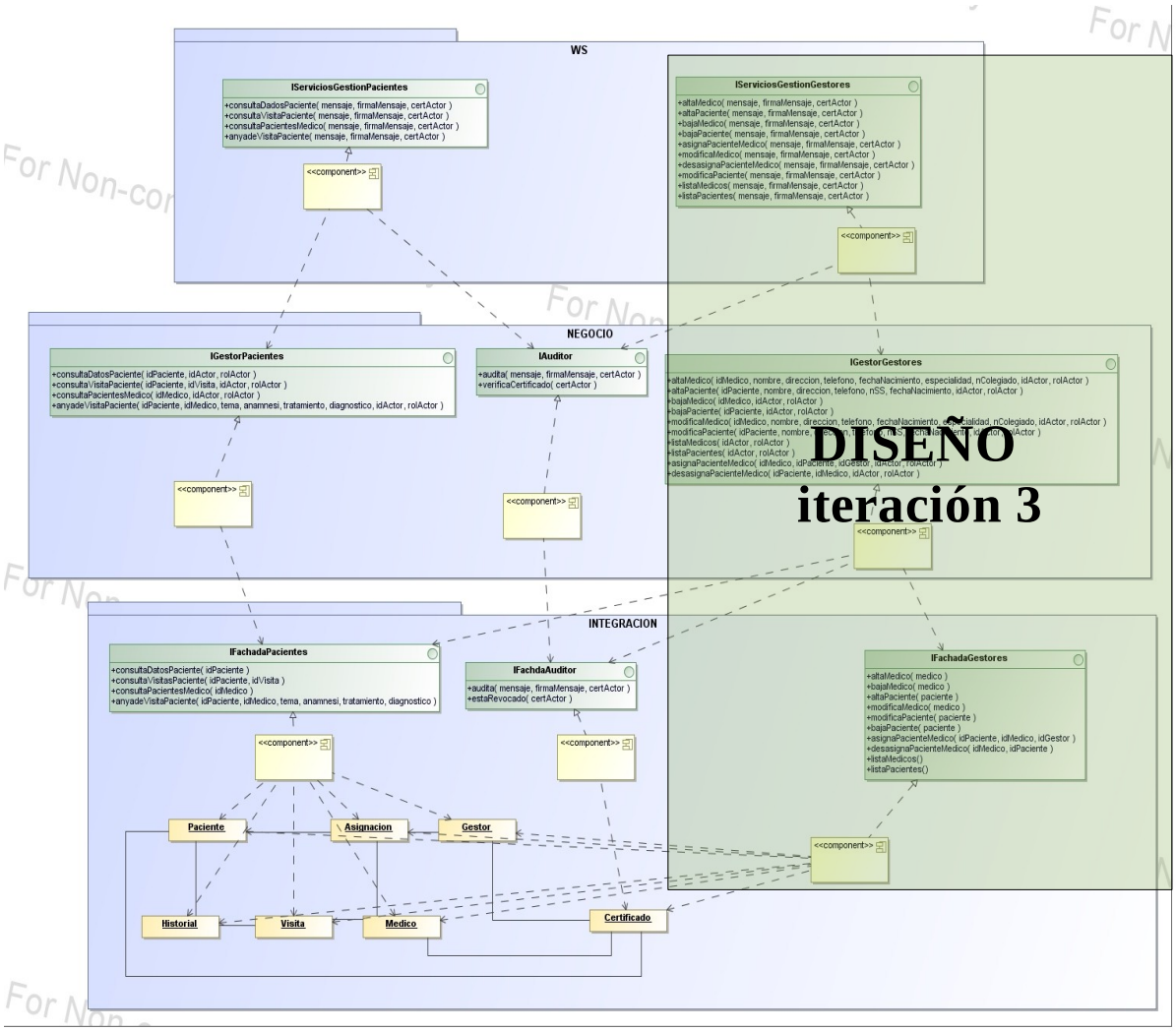


Ilustración 22: Diseño (Iteración 3)

En la página siguiente se muestra ampliado el esquema de las clases desarrolladas en esta iteración.

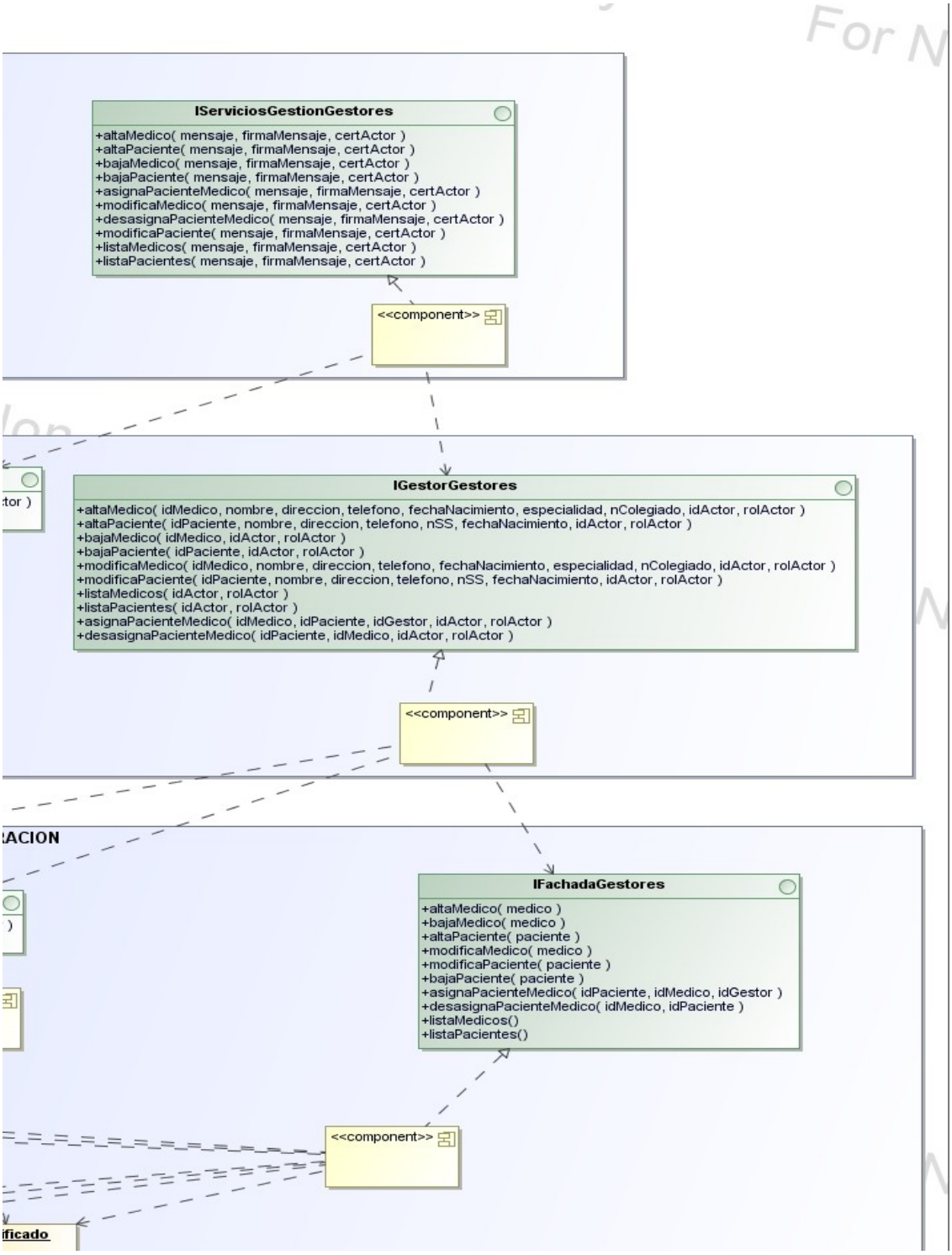


Ilustración 23: Diseño (Iteración 3). Ampliado

7.6.-Construcción y Pruebas (Iteración 3).

En esta etapa se construyen y prueban las clases especificadas en el apartado anterior.

Los casos de prueba seleccionados son:

CASO	VALOR ESPERADO	RESULTADO
alta paciente “0000000-P” actor: “0000000-G”; rol: “gestor”	mensaje ok	✔
alta medico “0000000-M” actor: “0000001-G”; rol: “gestor”	mensaje ok	✔
baja paciente “0000000-P” actor: “0000000-G”; rol: “gestor”	mensaje ok, se incluye fecha de baja al paciente	✔
baja medico “0000000-M” actor: “0000001-G”; rol: “gestor”	mensaje ok, se incluye fecha de baja al paciente	✔
lista médicos actor: “0000000-G”; rol: “gestor”	se obtienen los médicos	✔
lista pacientes actor: “0000000-P”; rol: “gestor”	se obtienen los pacientes	✔
lista médicos actor: “0000000-M”; rol: “medico”	mensaje de error	✔
modifica paciente “0000000-P” actor: “0000000-G”; rol: “gestor”	se modifica e paciente	✔
modifica medico “0000000-M” actor: “0000001-G”; rol: “gestor”	se modifica el médico	✔
asigna paciente “0000000-P” a médico “0000000-M” actor: “0000001-G”; rol: “gestor”	aparece asignado el paciente al médico	✔
asigna paciente “0000000-P” a médico “0000000-M” actor: “0000001-G”; rol: “gestor”	ya asignado, se produce mensaje de error	✔
desasigna paciente “0000000-P” a médico “0000000-M” actor: “0000001-G”; rol: “gestor”	se elimina la asignación, mensaje ok	✔
desasigna paciente “0000000-P” a médico “0000000-M” actor: “0000001-G”; rol: “gestor”	ya no asignado; mensaje de error	✔

Capítulo 8

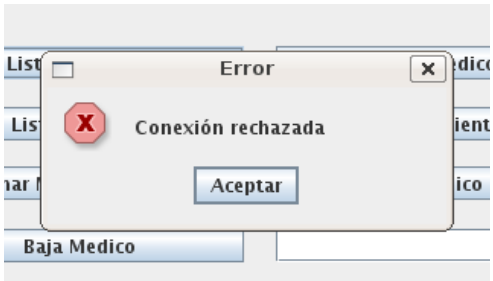
8.-Interfaces gráficas.

Para la construcción de las interfaces gráficas se utilizará la librería Swing de Java, en la versión 6 utilizada para construir el resto de la aplicación.

Las pantallas se diseñarán utilizando un plugin de eclipse que facilita esta tarea: Jigloo 4.0.5. Se trata de software libre para uso no comercial.

Aunque para este PFC es la parte menos relevante, se ha diseñado y construido el número mínimo de pantallas para dar suficiente funcionalidad a Paciente, Médico y Gestor.

Una cuestión a la que se le ha dado importancia es a la gestión de las excepciones, que propagan información relacionada con fallos en la aplicación hasta el cliente. El framework de Axis2 envía al cliente las excepciones ocurridas en el servidor en forma de objetos de la clase AxisFault, que pueden ser recuperados por el cliente e informarle sobre funcionamientos anómalos en sus acciones o en fallos del sistema. Ejemplos de mensajes son:



8.1.-Diseño (Iteración 4).

En esta fase se diseña la navegación de los diferentes actores sobre las pantallas a construir.

A continuación se definen las pantallas que compondrán la aplicación para cada uno de los usuarios y la navegación posible entre ellas.

Navegación paciente:

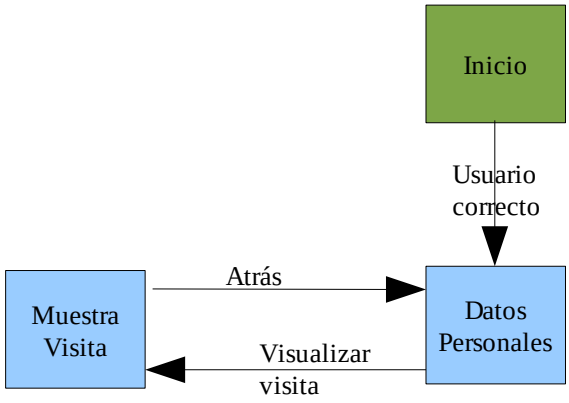


Ilustración 24: Navegación Paciente

Navegación Médico:

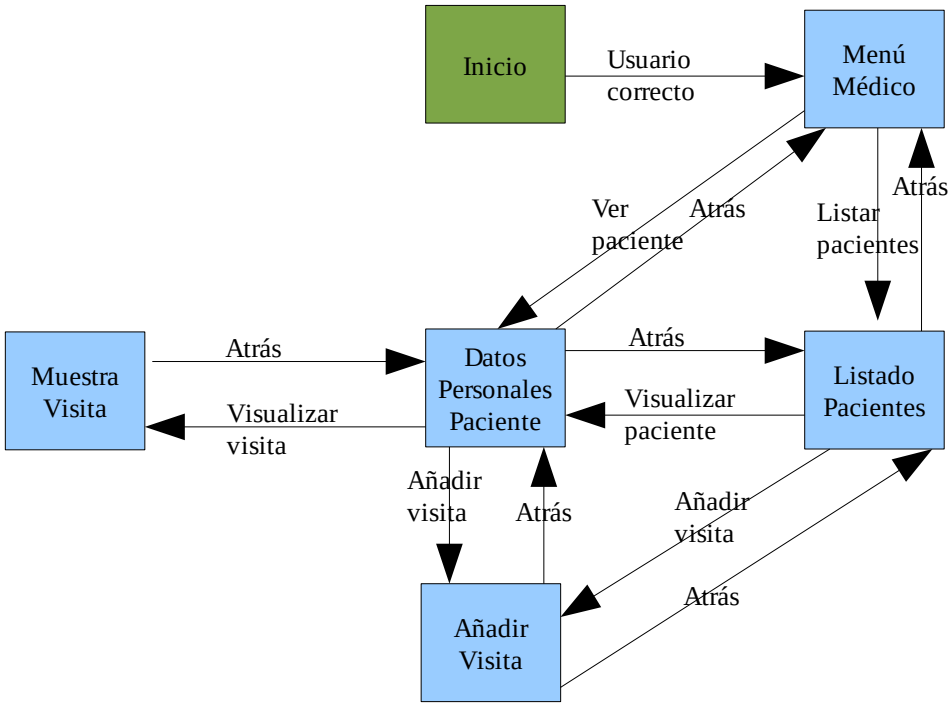


Ilustración 25: Navegación Médico

Navegación Gestor:

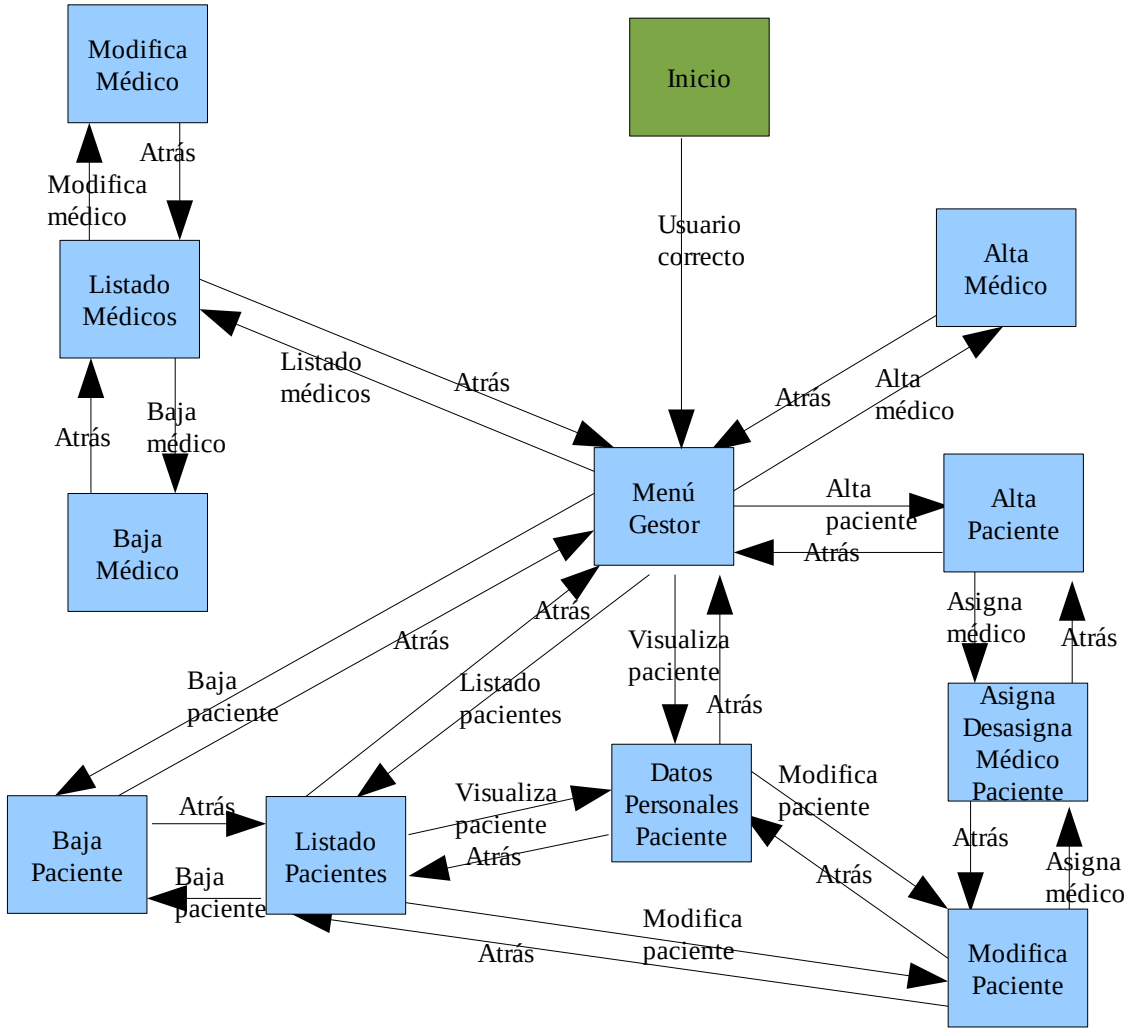


Ilustración 26: Navegación Gestor

La arquitectura de la parte cliente de la aplicación utilizará un JFrame como contenedor principal que servirá además de controlador, siguiendo el patrón MVC[28].

Los usuarios comenzarán autenticándose ante la aplicación; para ello será necesario que introduzcan su identificador (dni), la clave de protección de la clave privada contenida en el keystore y la localización del keystore. La clave de acceso al propio keystore está en un fichero de configuración en claro. Aunque para las pruebas ambas claves coinciden (la del keystore y la que protege la clave privada) en un caso real deberían ser diferentes. La clave introducida por el usuario es usada además por el framework de Axis2 para los procesos de firma digital.

La navegación hacia atrás se consigue utilizando una pila (Stack) de paneles, de tal manera que al pulsar el botón "Atrás" se recupera el JPanel anterior.

28 Modelo Vista Controlador

8.2.-Construcción y Pruebas (Iteración 4).

Cada una de las pantallas constituye un JPanel, en el que se muestran los controles (botones, etiquetas, desplegables de selección, etc.). Los Jpanel son controlados por un JFrame, que los contiene y gobierna la navegación entre las diferentes pantallas.

Es el JFrame el que instancia el cliente de Web Service, al que interroga respecto de las acciones que solicita el actor sobre cada uno de los JPanel.

Ejemplo de navegación por parte del Gestor es:

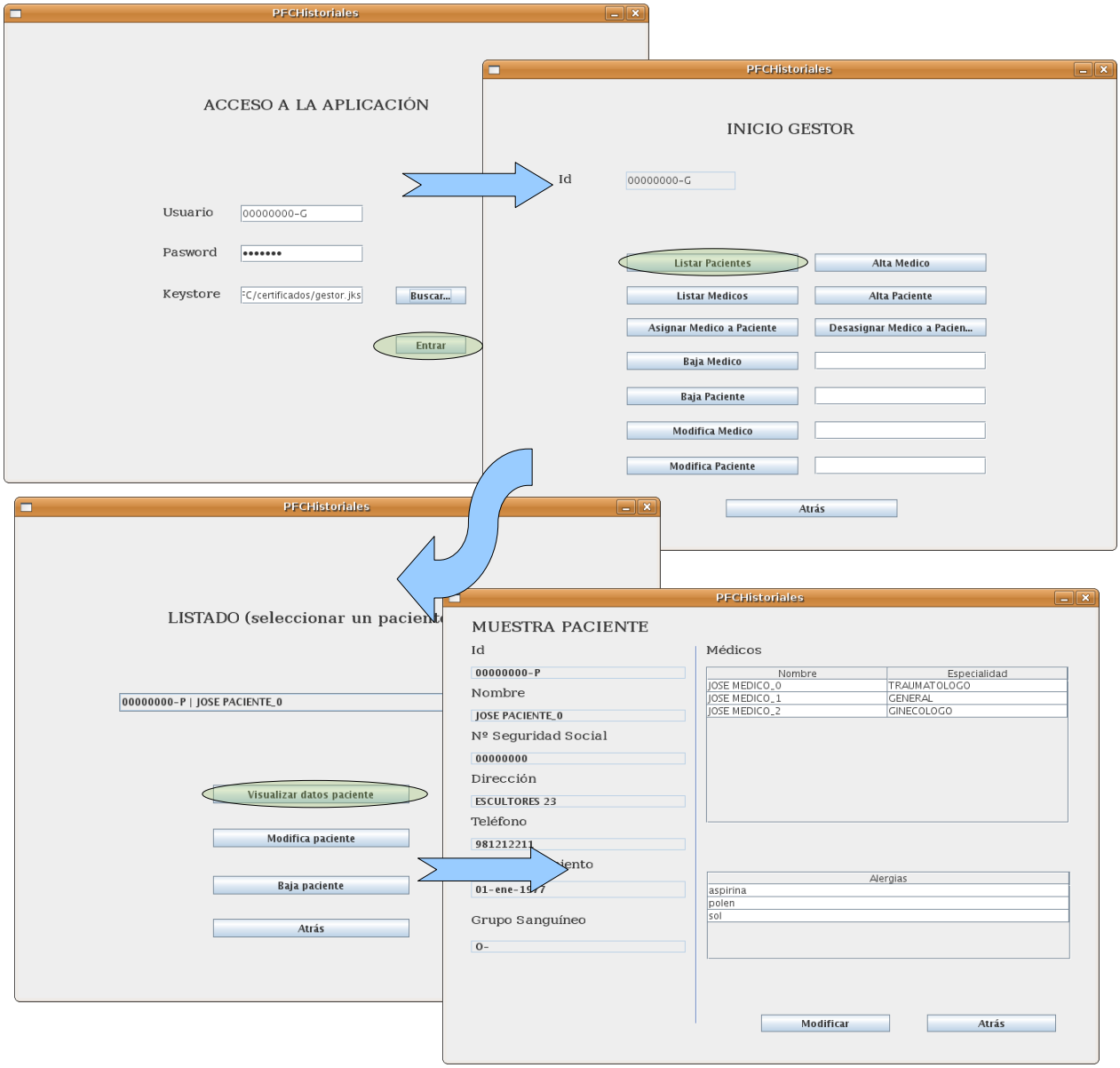


Ilustración 27: Ejemplo navegación Gestor

Veamos la navegación de un médico:

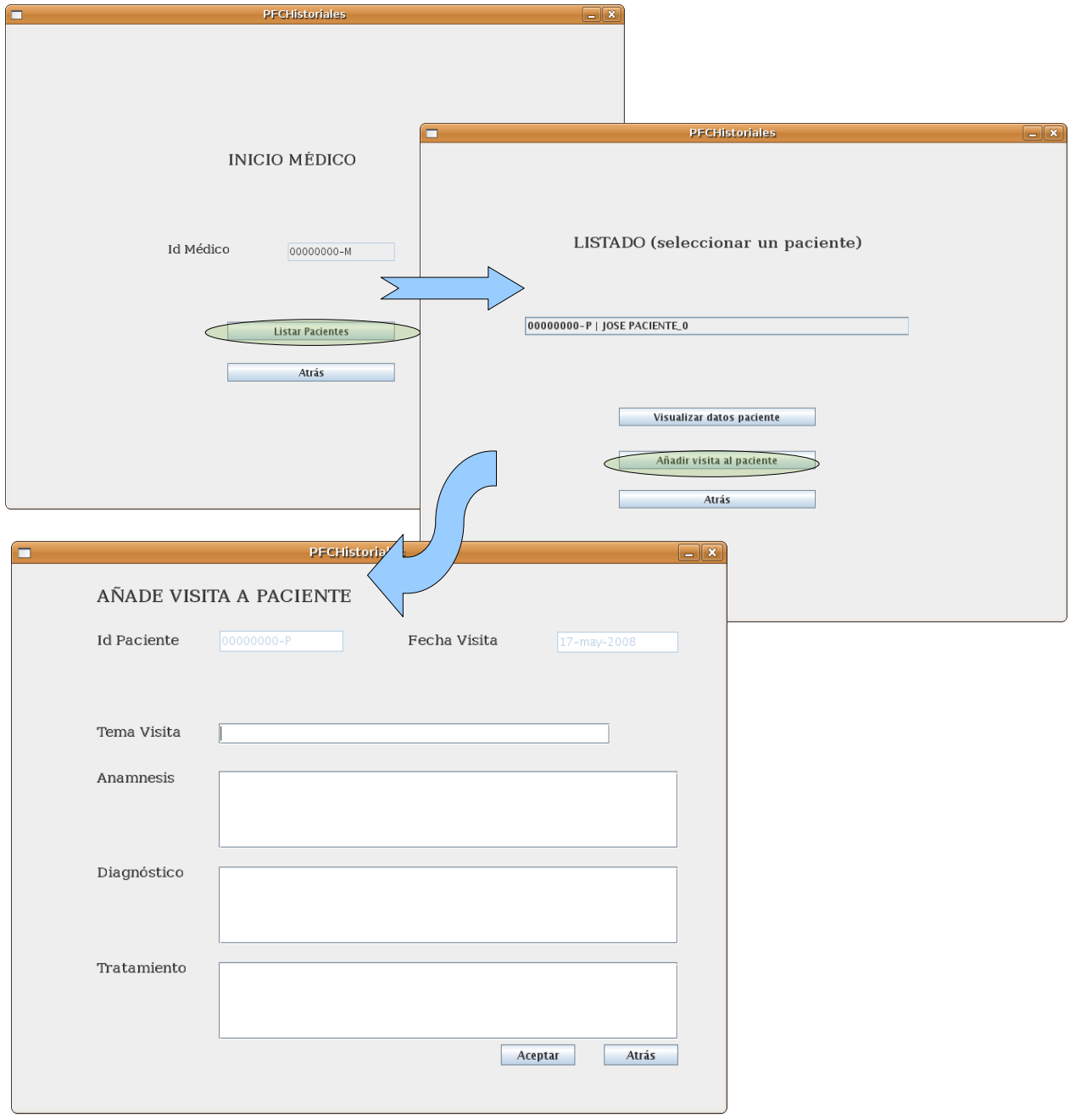


Ilustración 28: Ejemplo navegación Médico

Y finalmente pantallas utilizables por el Paciente:

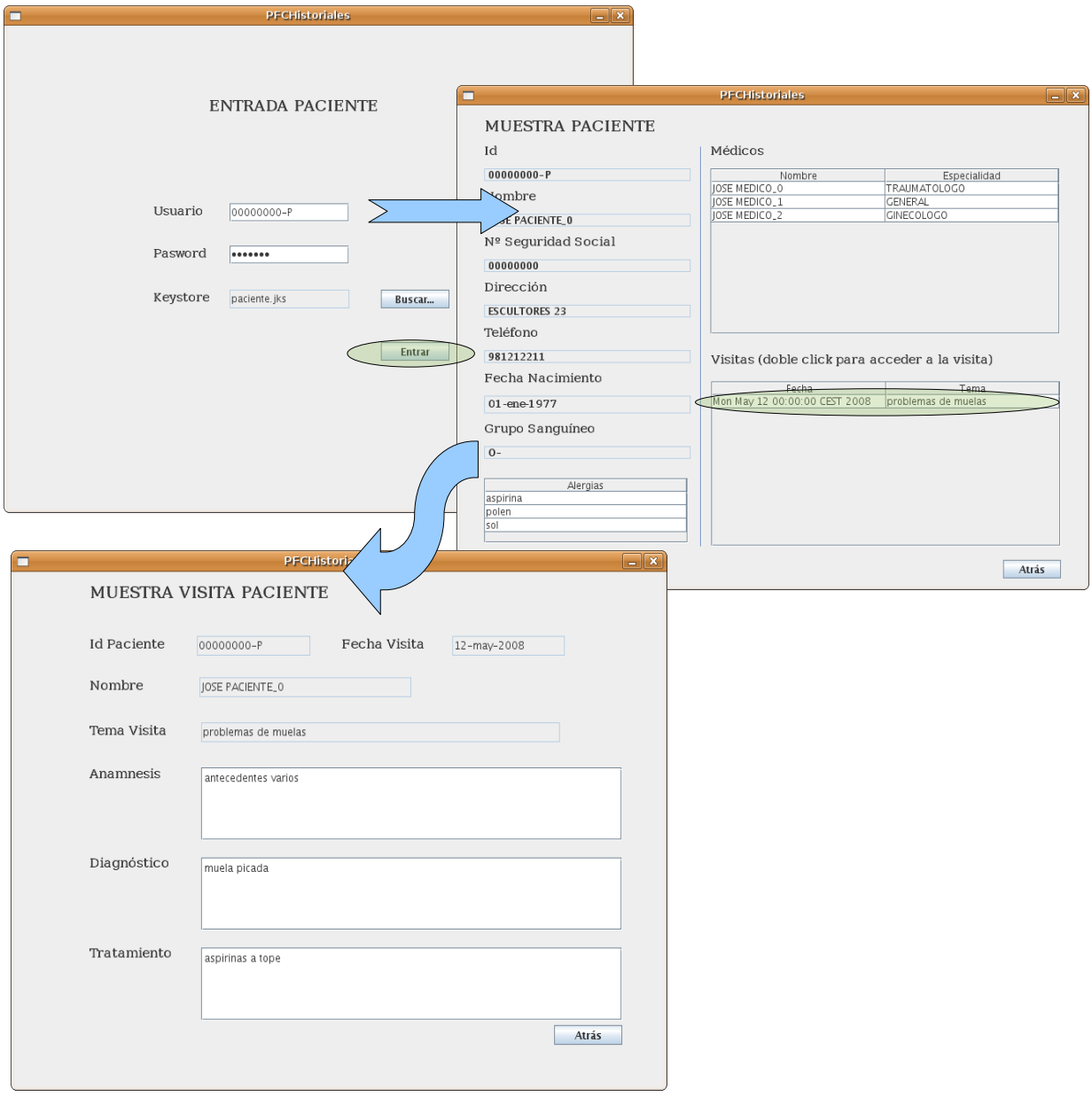


Ilustración 29: Ejemplo navegación paciente

Las pruebas en esta iteración consisten en:

- Usuario Gestor:
  - creación de pacientes.
  - creación de médicos.
  - asignación de médicos a pacientes.
  - modificación de pacientes.
  - modificación de médicos.
  - listados.
  - baja de médicos.
  - baja de pacientes.
- Usuario Médico:
  - listar pacientes.
  - añadir visita a un paciente.
  - consultar un paciente.
  - consultar visitas de un paciente.
- Usuario Paciente:
  - consultar sus datos.
  - consultar las visitas.

Estas pruebas se realizan directamente utilizando la interfaz gráfica creada.

8.3.-Diseño (Iteración 5).

En la última iteración se dejará preparado el sistema para realizar los procesos de auditoría, en los que se gestionan los siguientes procedimientos:

- Guardar en la Base de Datos las acciones relevantes de los actores a efectos de seguridad.
- Verificación de la revocación de certificados.
- Comprobación de las actuaciones de los actores.

En la base de datos se guardan, para auditoría, los siguientes datos:

- El id del Actor que realiza la acción.
- La fecha.
- El mensaje.
- Si se ha accedido a lo solicitado o no.

Conforme lo anterior, la tabla de auditoría se compone de:

Auditoria (idEntrada, idActor, fecha, mensaje, autorizado, firmaMensaje, claveCifrada).

El mensaje estará cifrado con una clave de sesión, firmado y con la clave de sesión cifrada con la clave privada del servidor.

Tablas resultante y tipo de datos:

TABLA: Auditoria		
CAMPO	**TIPO DATO**	**DESCRIPCIÓN**
idEntrada	INT (AUTO_INCREMENT)	clave de la tabla, auto incrementable
idActor	VARCHAR (10)	id del actor
fecha	DATE	fecha del acceso
mensaje	BLOB	mensaje cifrado
autorizado	BOOLEAN	si ha tenido éxito la operación
firmaMensaje	BYTE (128)	firma del mensaje
claveSesion	BYTE (128)	clave de sesión simétrica CIFRADA

La revocación de los certificados se comprueba consultando si el certificado en base de datos tiene fecha de revocación.

La comprobación de la autenticidad de un mensaje se realiza según se explica en el anexo IV (código relevante).

8.4.-Construcción y Pruebas (Iteración 5)

En esta fase se construyen las clases relacionadas con la Auditoría.

En la arquitectura general, en la parte servidora, afecta a:

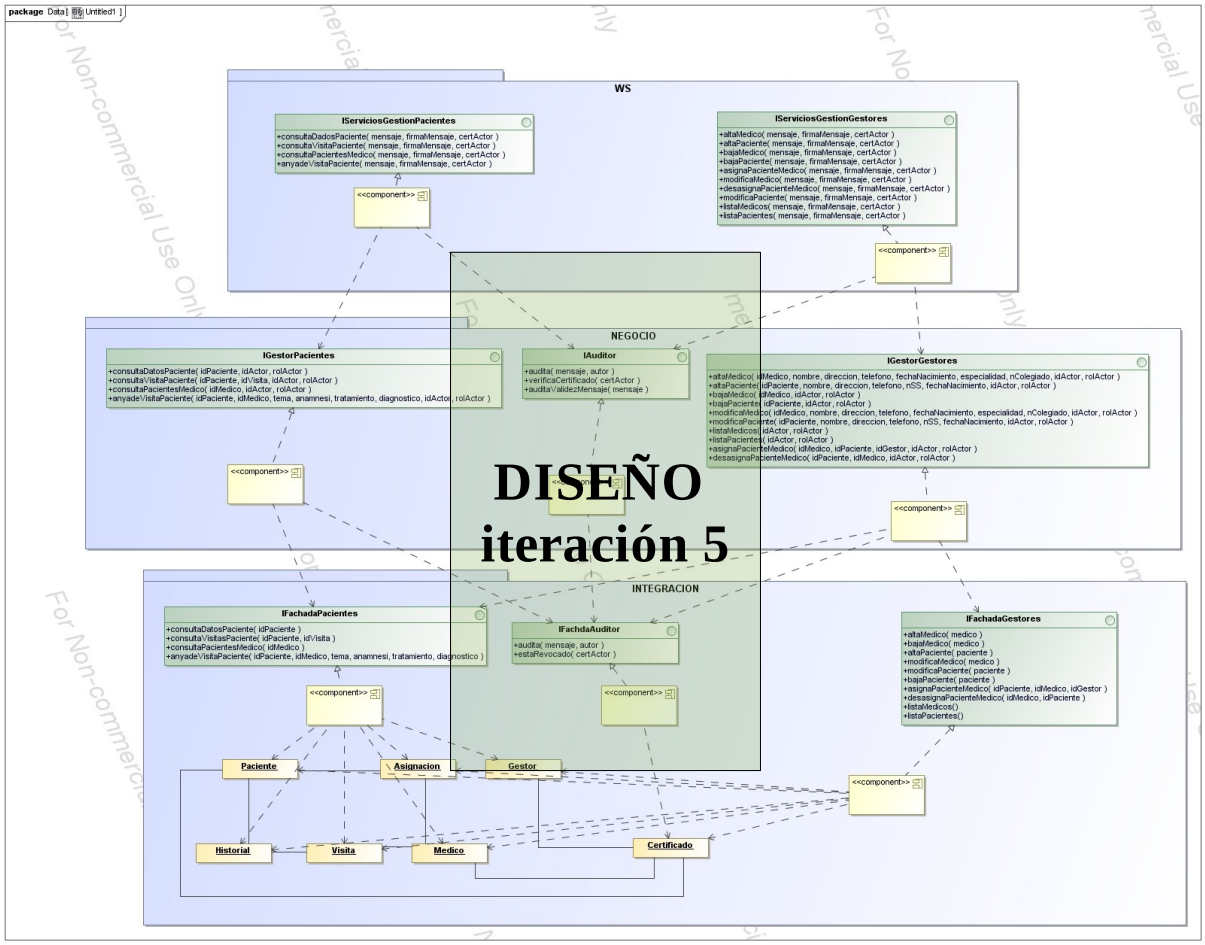


Ilustración 30: Arquitectura Servidor final

Las clases construidas han sido:

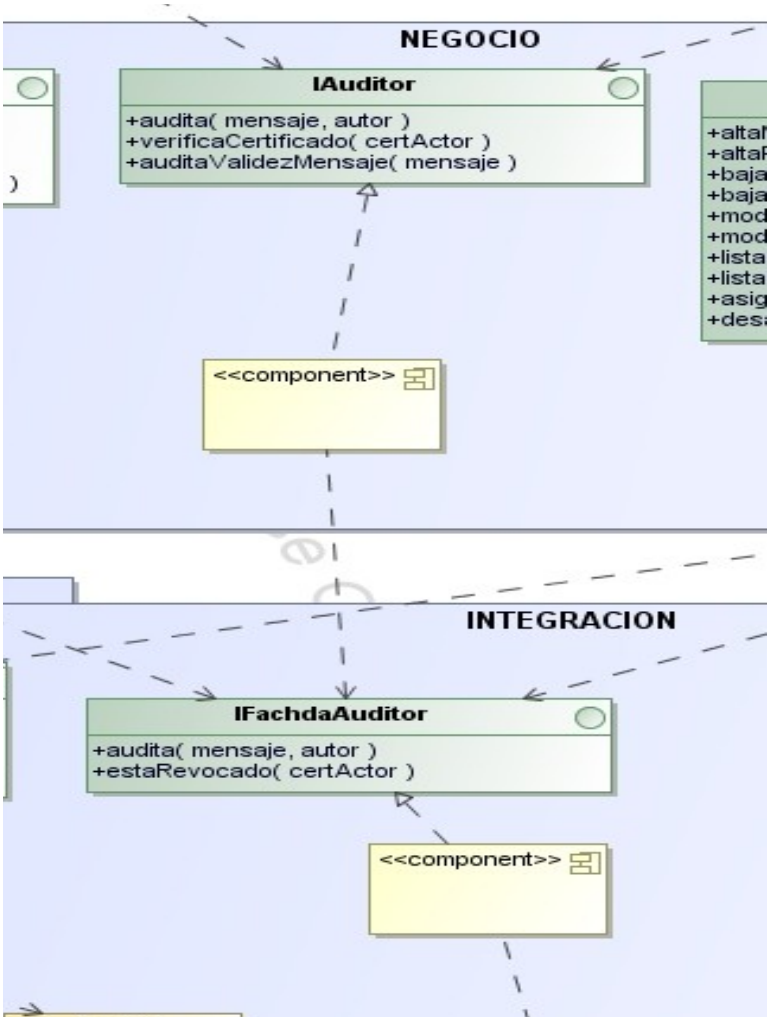


Ilustración 31: Interfaces y Clases (iteración 5)

En el paquete **Negocio** se define el interfaz IAuditor, con los métodos:

- audita (mensaje, autor): recibe un mensaje, el autor del mismo e invoca la clase correspondiente en la capa de Integración para su almacenamiento en la base de datos.
- verificaCertificado(certActor): recibe un certificado y realiza las siguientes comprobaciones:
  - comprueba que no está caducado.
  - verifica que está firmado por la CA de la PKI propia.
  - invoca a la capa de Integración para ver si está revocado.
- auditaValidezMensaje(mensaje): recibe un mensaje (SOAP, con la firma del cuerpo y timestamp en la cabecera) y procesa los componentes de seguridad del elemento Header.

En el paquete **Integración** se define e implementa el interfaz IfachadaAuditor, con los métodos:

- audita (mensaje, autor): guarda en base de datos, cifrado y firmado, el mensaje, indicando la fecha en la que se produjo y el autor del mismo.
- estaRevocado(certActor): verifica s un certificado está revocado.

Las pruebas en esta fase se realizan sobre la aplicación construida (GUI) y se verifican los efectos de las acciones sobre la base de datos.

Capítulo 9

9.-Conclusiones.

Ha sido, incluso antes de comenzar a trabajar en el propio proyecto, un trabajo de investigación interesante y absorbente, que ha conseguido que este autor se ponga medianamente al día en el apasionante mundo de la seguridad informática, los web services y sus estándares asociados.

El proyecto empezó su andadura allá por el mes de julio de 2.007, cuando me surgió la inquietud de buscar un tema de investigación, que fuese de interés personal y profesional, y que pudiese adaptarse a los temas propuestos por la UOC como proyectos de fin de carrera. La criptografía ha sido siempre un tema que me ha interesado, incluso antes de comenzar los estudios de ingeniería informática ya había leído algunos documentos que versaban sobre este tema. Por otra parte, la creciente importancia de las tecnologías asociadas con los WS, hacían esta otra materia también interesante para que el PFC se enfocase en esa dirección. No quedaba más que aunar Seguridad informática y WS para que el PFC tomase ambos campos de investigación.

A lo anterior había que sumar el reto tecnológico de buscar una plataforma de desarrollo y de soporte a la aplicación resultante que fuese exportable al mundo real y que no se quedase en un simple experimento, en un polígono de pruebas únicamente. Ha sido esta parte la que más quebraderos de cabeza me ha causado, pues no ha sido fácil que los planteamientos teóricos iniciales terminasen corriendo sobre la plataforma de ejecución elegida. Es de destacar que la mayor parte de escollos encontrados han sido salvados gracias a la estupenda red de redes, Internet, sin la cual este proyecto se hubiese quedado definitivamente en proyecto de PFC.

Considero que el PFC final ha conseguido sus objetivos; por un lado tenemos una aplicación de gestión de historiales médicos segura, que garantiza la confidencialidad, integridad, autenticación y autorización de los actores, y el no repudio de sus acciones. Tenemos además la información guardada en una base de datos de forma que los datos de especial relevancia quedan protegidos contra accesos no autorizados a la misma.

A pesar de la ingente cantidad de horas de dedicadas al proyecto (unas 320 horas aproximadamente) han quedado cuestiones sin implementar, aunque quedan, más abajo, planteadas y definido el modo de resolverlas.

Me parece que es el momento de analizar lo construido en el marco de la seguridad informática y ver qué potenciales vulnerabilidades quedan por resolver:

1. Validación de los campos de entrada de información, tanto en el cliente como en el servidor. Si tomamos un ejemplo del código que en la clase del lado servidor recibe peticiones del cliente, vemos que no se hace validación de la entrada:

```
idPaciente = element.getFirstChildWithName(new Qname("param0")).getText();
```

Por ejemplo, un atacante podría pasar, dentro del elemento <*param0*> una cadena de texto que, en vez de contener el id del paciente, aparezca código SQL (ataque SQL injection).

Esta vulnerabilidad se solventa introduciendo validaciones en cliente y servidor, por ejemplo, en el caso del idPaciente utilizado como ejemplo, definir un patrón y cotejar la entrada con este patrón. En el caso de la parte servidora, teniendo en cuenta que los parámetros entran en formato xml, se podría definir un DTD o un Schema y validar la entrada contra él.

De esta forma también impediríamos que el usuario pueda equivocarse al introducir los datos (por ejemplo, que el dni siga un patrón concreto, que el número de la seguridad social se ajuste a un patrón, etc.). El único dato que se valida de entrada en la parte cliente son las fechas, utilizándose objetos de la clase *JFormattedTextField.*

2. Ataques de texto en claro conocido en determinados mensajes, en los que el contenido del mismo es predecible, o en determinados datos guardados en la base de datos, por ser su contenido conocido, o con un número de posibles alternativas de su contenido limitadas.

   Aunque en los protocolos criptográficos ya se define la utilización de nonces en los mensajes, esta solución no está implementada por falta de tiempo.

   La solución evidente ante este problema consiste en añadir un nonce a los mensajes y datos cifrados susceptibles de ataque de texto claro conocido. Es cuestión de añadir un elemento más de carácter aleatorio a estos mensajes, incluyendo el correspondiente parámetro o elemento obtenido llamando a la clase Random de la librería estándar de Java:

   ```
   new Random().nextBytes(arg0)
   ```

   Donde arg0 es un array de longitud *n* bytes que guardará los bytes aleatorios generados.

3. Gestión de las claves de acceso, en el servidor, al keystore en el que se almacena la clave privada utilizada para cifrado y firma. Si un atacante accede al sistema de ficheros donde está instalado el servidor de aplicaciones Apache-Tomcat, tendrá acceso a la clave privada y al keystore. A partir de ahí podrá acceder a la base de datos y descifrar los datos más críticos contenidos en ella.

   ¿Solución a esta vulnerabilidad? Ciertamente la solución empleada, que consiste en incluir en el código de la clase *PWCBHandler* la palabra de acceso a la clave privada, no es la solución más segura, pues existen herramientas que facilitan descompilar las clases Java y obtener el valor de las variables. Existen en el mercado soluciones hardware para almacenar y custodiar las claves privadas en los servidores. Otra solución es la intervención del administrador de la aplicación cada vez que ésta tenga que ser puesta en marcha, introduciendo manualmente la clave.

4. Gestión de las claves del lado del cliente. La clave de acceso al keystore es conocida, no así la clave que custodia la parte privada de la clave RSA. El cliente debe mantener en secreto esta clave, pues el acceso a la misma permitiría suplantar su identidad ante el servidor.

No sólo quedan cuestiones relacionadas con la seguridad pendientes; otros aspectos que se podrían mejorar en la aplicación son:

1. PKI externa. Aunque en el planteamiento inicial se contempla la utilización de certificados de PKI diferente a la creada para el proyecto (por ejemplo, el dnie) no se ha podido implementar nada en la aplicación al respecto.
2. Uso de un keystore elegido por el usuario. El hecho de utilizar el framework de Axis2 también desde la parte cliente, trae una serie de limitaciones, como es el hecho de que la configuración se haya tenido que mantener en el fichero axis2.xml de forma estática. Como en este fichero se encuentra la ubicación, tipo de almacén y clave de acceso al keystore, no ha sido factible que dinámicamente el usuario decida la ubicación del keystore a usar.
3. UI para auditoría. No existe una interfaz gráfica que permita realizar las tareas de auditoría, como son:
   - listado de accesos de un usuario en unas fechas concretas.
   - validación de una acción de un usuario, verificando la firma de su mensaje.
4. UI para gestión de los certificados. Tampoco existe una interfaz que permita al gestor la creación de los certificados a petición de los usuarios, la revocación de los mismos, etc.
5. UI búsquedas. Falta una interfaz que facilite operaciones de búsqueda.
6. Bajas de pacientes/médicos. Los procesos de baja de pacientes y médicos no borran sus datos de la base de datos, simplemente añaden una fecha de baja. Sería necesario un proceso que pasase a otras tablas a pacientes/médicos de baja, de forma que no se pudiesen obtener sus datos como si fuesen actores en activo.
7. Certificados en BD. Se ha creado la tabla de certificados en la base de datos, pero no ha sido necesario rellenar sus campos, debido a que las validaciones de los certificados utilizados por los actores se realiza verificando sobre el propio certificado la caducidad, y con el de la CA su integridad. Se utiliza como identificador de un certificado un hash en base64 del mismo:

   ```
   idCert=Base64.encode(cs.digest(cert.toString().getBytes(), "SHA1"));
   ```

8. Script de instalación. La instalación de la parte servidora es compleja, debido a que no se ha generado un simple fichero .aar, que es la forma más fácil de crear e instalar un Web Service en Axis2, ya que simplemente copiado en la carpeta *services/* de Axis, se despliega y pone en marcha. La razón de la imposibilidad de crear un .aar está en la existencia del componente inicializador, que instancia una serie de componentes. Para que este último funcione, las clases compiladas deben estar en la carpeta *classes/* de Axis, lo que no es posible si se encuentran dentro de un .aar.
9. No se ha creado un Datasource para la gestión eficiente de las conexiones a la base de datos. Su configuración en Apache es compleja, y es por esto por lo que no se ha considerado, teniendo en cuenta la poca concurrencia que habrá de accesos a la aplicación durante su construcción y pruebas. Su puesta en productivo sí que requeriría una gestión más adecuada de las conexiones a la base de datos.

10. Cuando se planteó la utilización de clientes Java, se vio necesario servirse de la tecnología Java Web Start para mantener actualizados estos componentes. La aplicación final no contempla esta solución para distribuir los clientes, por lo que habría que implementarla en un futuro próximo.

Glosario

ACID: propiedad de una base de datos para realizar transacciones seguras.

AES: Advanced Encryption Standard, es un sistema de cifrado por bloques.

Anamnesis: información proporcionada por el propio paciente al profesional durante una entrevista clínica, con el fin de incorporar dicha información a la Historia clínica del paciente.

Apache-Tomcat: Servidor de Aplicaciones Web de código abierto, de uso ampliamente extendido.

Axis: motor Web Services, desarrollado por la fundación Apache. Existen dos versiones, Axis y Axis2.

Base64: Sistema de codificación en base 64, utilizado para la transmisión de datos binarios, siendo la mayor potencia de dos que puede ser representada usando únicamente los caracteres imprimibles de ASCII.

Certificado digital: documento digital mediante el cual un tercero confiable (la autoridad de certificación) garantiza la vinculación entre la identidad de un sujeto y su clave pública.

Criptosistema simétrico: Sistema criptográfico que utiliza la misma clave en los procesos de cifrado y descifrado.

Criptosistema asimétrico: Sistema criptográfico que utiliza una clave diferente en los procesos de cifrado y descifrado.

Clave Pública: Componente de la clave en un criptosistema asimétrico conocido por terceros, utilizado para remitir al poseedor de la clave privada documentos cifrados.

Clave Privada: Componente secreto de la clave en un criptosistema asimétrico, utilizado para firma y descifrado.

Datasource: Componente que gestiona un conjunto de conexiones, facilitando la reutilización de las mismas.

DTD: Definición de tipo de documento es una descripción de estructura y sintaxis de un documento XML.

Framework: Estructura de soporte definida en la cual otro proyecto de software puede ser organizado y desarrollado.

Hash: función de hash es una función para resumir o identificar probabilísticamente un gran conjunto de información, dando como resultado un conjunto imagen finito generalmente menor.

Historial médico: Recopilación de documentos que aportan información normalizada sobre aspectos personales y sobre las alteraciones o enfermedades que padece o ha padecido a lo largo de su vida.

Java: Lenguaje de programación de alto nivel, orientado a objetos.

Keystore: Almacén digital en el que se custodian claves y certificados.

RSA: Sistema criptográfico con clave pública basado en un algoritmo asimétrico cifrador de bloques.

Schema: Lenguaje utilizado para describir la estructura y las restricciones de los contenidos de los documentos XML

SGBD: Software dedicado a servir de interfaz entre la base de datos, el usuario y las aplicaciones que la utilizan. Se compone de un lenguaje de definición de datos, de un lenguaje de manipulación de datos y de un lenguaje de consulta.

SHA: Tipo de función hash que produce una salida resumen de 160 bits.

SOA: Arquitectura de software que define la utilización de servicios para dar soporte a los requerimientos de software del usuario

SQL: Es un lenguaje declarativo de acceso a bases de datos relacionales.

Swing: Biblioteca gráfica para Java que forma parte de las Java Foundation Classes (JFC).

UML: Lenguaje de modelado de sistemas de software más conocido y utilizado en la actualidad; está respaldado por el OMG (Object Management Group).

Web Services: Sistema de software diseñado para ofrecer interacción máquina a máquina sobre una red de ordenadores.

WS: Véase Web Services

WS-*: Conjunto de protocolos relacionados con los Web Services.

XMLSignature: Recomendación del W3C que define la sintaxis para la utilización de firma digital sobre documentos xml

XMLEncryption: Recomendación del W3C que define la sintaxis para la utilización de cifrado y descifrado sobre documentos xml

X509v3: Tipo de certificado digital.

Bibliografía

[1] *B. Hartman , D. J. Flinn, K. Beznosov , S. Kawamoto*: Mastering Web Services Security. Wiley 2003.

[2] *Mark O'Neill et al*: Web Services Security. McGraw-Hill/Osborne, 2003.

[3] *J. Rosenberg, D. L. Remy*: Securing Web Services with WS-Security. Sams, 2004.

[4] *M. Colan, J. Miller*: Understanding Web Services Security . IBM Corp, 2004.

[5] The Java Web Services Tutorial . Sun Microsystems, 2006.

[6] *F. J. Ceballos*: JAVA2 Interfaces gráficas y aplicaciones para Internet. RA-MA, 2004.

[7] Webgrafía (2.008):

Artículos sobre seguridad en ws:

http://webservices.xml.com/security/

http://www.devx.com/Java/Article/28816/1954?pf=true

Información sobre Axis2, guías, instalación, módulos, etc.:

http://ws.apache.org/axis2/index.html

http://today.java.net/pub/a/today/2006/12/13/invoking-web-services-using-apache-axis2.html

http://ws.apache.org/axis2/1_1/modules.html

http://wso2.org/library/174

http://wiki.apache.org/ws/FrontPage/WsFx/wss4jFAQ#usernme

http://www.developer.com/java/web/article.php/10935_3529321_1

http://www.developer.com/services/article.php/10928_3557741_1

http://www.developer.com/java/ent/article.php/10933_3613896_2

http://wso2.org/library/2060

http://www.javaranch.com/journal/200709/web-services-authentication-axis2.html

http://wso2.org/library/777

Artículos de Axis2 sobre eclipse:

http://wso2.org/library/1719

http://swik.net/axis2+eclipse

Arquitectura de Axis2:

http://www-128.ibm.com/developerworks/webservices/library/ws-apacheaxis2/

Sobre Rampart, uso e instalación:

http://wso2.org/library/240

http://blog.sweetxml.org/2007/12/rampart-basic-examples-how-you-add-ws.html

En relación al paquete WSS4J:

http://ws.apache.org/wss4j/package.html

http://weblogs.asp.net/jdanforth/archive/2005/01/16/354060.aspx

Información general de instalación:

https://wiki.internet2.edu/confluence/display/CPD/OSCARS+Installation

Sobre J2EE:

http://www.developer.com/java/ent/article.php/10933_3467801_2

http://java.sun.com/products/servlet/Filters.html

Otras plataformas, sólo exploradas (WSIT, Metro, Glassfishetc.):

http://www.netbeans.org/kb/60/websvc/wsit.html

http://webservices.xml.com/pub/a/ws/2003/11/25/jwss.html

http://docs.sun.com/app/docs/doc/820-1072

https://xwss.dev.java.net/Securing_JAVASE6_WebServices.html

http://www.sitepoint.com/article/getting-started-xml-security

http://wiki.apache.org/ws/StackComparison

JAVA, en general:

http://www.uv.es/sto/cursos/seguridad.java/html/sjava-40.html

http://java.sun.com/javase/6/docs/technotes/guides/security/index.html

http://java.sun.com/javase/6/docs/technotes/guides/security/crypto/CryptoSpec.html

http://www.devx.com/xml/Article/28701/0/page/3

Tomcat y Axis:

http://deigote.blogspot.com/2006/06/servicios-web-1-instalando-tomcat-y.html

MySQL:

http://www.guia-ubuntu.org/index.php?title=MySql

http://www.herongyang.com/jdbc/MySQL-CLOB-Large-Object.html

http://dev.mysql.com/doc/refman/5.0/en/connector-j-reference-type-conversions.html

http://www.developer.com/java/data/article.php/3417381

Anexos

Anexo I. Plataforma de desarrollo.

PC: AMD X2 64, 4Gb

S.O.: Linux 2.6.22-14-generic (distribución Ubuntu 7.10)

IDE:

-Netbeans 6.0
-Eclipse 3.3.2

Servidores:

-Glassfishv2
-Tomcat 6.0.16
-Jboss 4.2.2 GA

Librerías adicionales instaladas:

-JDK 1.6.0_03
-Axis2 1.3 (proyecto Apache)
-Rampart 1.3 (proyecto Apache)
    -wss4j-1.5.3.jar
    -xmlsec-1.4.0.jar
-El servidor Glassfishv2 incorpora el paquete:
    -xwss 3.0. XML y WebServices Security (Projecto Metro).
-MySQL connector (versión 5.1.6).

Gestión de claves/certificados:

-OpenSSL 0.9.8e.
-keytool (JDK6).

Creación de gráficos UML:

-MagicDraw 15.0 (Versión Community)

Base de datos:

-MySQL 5.0

Otro software:

-Tcpmon 1.0
-MySQL Query Browser
-MySQL Administrator

Proveedores librerías criptográficas:

En principio se pretende utilizar el que acompaña la distribución 6 de SUN, suficientemente completa en algoritmos respecto de las versiones anteriores. Al objeto de levantar las restricciones a determinadas longitudes de claves en los algoritmos utilizados, se sustituyen los ficheros local_policy.jar y US_export_policy.jar ubicados en el directorio <JAVA_HOME>/lib/security/ por los que proporciona SUN a tal efecto.

La librería Rampart 1.3 hace uso por defecto del paquete criptográfico Bouncy Castle (bcprov-jdk16-138.jar). Para utilizarla, cuando sea necesario, se instalará dinámicamente el proveedor, indicándose su uso con el código java siguiente:

```
BC = Class.forName( "org.bouncycastle.jce.provider.BouncyCastleProvider");
Security.insertProviderAt((Provider)BC.newInstance(), 2);
```

Y usando BC en las instancias de objetos criptográficos, por ejemplo:

```
Cipher aesCipher = Cipher.getInstance("AES","BC");
```

Por todo lo anterior, inicialmente no se utilizará la implementación criptográfica IAIK.

Si presentamos por curiosidad, los proveedores instalados por defecto, obtenemos:

```
-----------------------------------
jose@jose-desktop:~/Escritorio$ java InfoProveedores
-----------------------------------
Proveedores instalados en su sistema
-----------------------------------
Núm. proveedor : 1
Nombre         : SUN
Versión        : 1.6
Información    :
  SUN (DSA key/parameter generation; DSA signing; SHA-1, MD5 digests; SecureRandom; X.509
certificates; JKS keystore; PKIX CertPathValidator; PKIX CertPathBuilder; LDAP, Collection
CertStores, JavaPolicy Policy; JavaLoginConfig Configuration)
Propiedades    :
-----------------------------------
Núm. proveedor : 2
Nombre         : SunRsaSign
Versión        : 1.5
Información    :
  Sun RSA signature provider
Propiedades    :
-----------------------------------
Núm. proveedor : 3
Nombre         : SunJSSE
Versión        : 1.6
Información    :
  Sun JSSE provider(PKCS12, SunX509 key/trust factories, SSLv3, TLSv1)
Propiedades    :
-----------------------------------
Núm. proveedor : 4
Nombre         : SunJCE
Versión        : 1.6
Información    :
  SunJCE Provider (implements RSA, DES, Triple DES, AES, Blowfish, ARCFOUR, RC2, PBE, Diffie-
Hellman, HMAC)
Propiedades    :
-----------------------------------
```

```
Núm. proveedor : 5
Nombre         : SunJGSS
Versión        : 1.0
Información    :
  Sun (Kerberos v5, SPNEGO)
Propiedades    :
-------------------------------------
Núm. proveedor : 6
Nombre         : SunSASL
Versión        : 1.5
Información    :
  Sun SASL provider(implements client mechanisms for: DIGEST-MD5, GSSAPI, EXTERNAL, PLAIN,
CRAM-MD5; server mechanisms for: DIGEST-MD5, GSSAPI, CRAM-MD5)
Propiedades    :
-------------------------------------
Núm. proveedor : 7
Nombre         : XMLDSig
Versión        : 1.0
Información    :
  XMLDSig (DOM XMLSignatureFactory; DOM KeyInfoFactory)
Propiedades    :
-------------------------------------
Núm. proveedor : 8
Nombre         : SunPCSC
Versión        : 1.6
Información    :
  Sun PC/SC provider
Propiedades    :
-------------------------------------
```

Anexo II. Creación PKI.

Creación CA y Certificados necesarios con openssl y keytool:

```
#CREAMOS KEYS CA
openssl genrsa -des3 -out CA.key 2048
#CERTIFICADO AUTOFIRMADO CA
openssl req -new -sha1 -x509 -key CA.key -out CA.crt -days 360 -config openssl.cnf

#CREAMOS KEYS PACIENTE
keytool -genkey -alias paciente -keyalg RSA -dname "CN=Jose Paciente Paciente, C=ES,
L=Barcelona, ST=Barcelona, O=UOC, OU=Paciente, DNQ=00000000-P" -keystore paciente.jks
#SOLICITUD CERTIFICADO
keytool -certreq -keystore paciente.jks -storepass uoc0506 -alias paciente -file
paciente.cert.req
#CREACION DE CERTIFICADO POR CA
openssl x509 -req -in paciente.cert.req -days 180 -CA CA.crt -CAkey CA.key -CAcreateserial
-extfile openssl.cnf -extensions usr_cert -out paciente.crt
#INCORPORAMOS CERTIFICADO CA A KEYSTORE
keytool -import -file CA.crt -keystore paciente.jks -storepass uoc0506 -alias ca
#INCORPORAMOS CERTIFICADO PACIENTE A KEYSTORE
keytool -import -file paciente.crt -keystore paciente.jks -storepass uoc0506 -alias paciente
#COMPROBAMOS CONTENIDO KEYSTORE
keytool -list -v -keystore paciente.jks

############
##REPETIR PARA MEDICO, GESTOR Y SERVIDOR
############


#AÑADIMOS EL CERTIFICADO DEL SERVIDOR A LOS DEMÁS KEYSTORES
keytool -import -file servidor.crt -keystore paciente.jks -storepass uoc0506 -alias servidor
keytool -import -file servidor.crt -keystore gestor.jks -storepass uoc0506 -alias servidor
keytool -import -file servidor.crt -keystore medico.jks -storepass uoc0506 -alias servidor
```

Anexo III. Preparación plataforma de desarrollo.

Apache-Tomcat:

- Descarga de la versión 6.0.16 de la web: http://tomcat.apache.org/
- Descomprimir el .zip en la carpeta de trabajo /home/jose/

Axis2:

- Descargar la versión 1.3 de la web: http://ws.apache.org/axis2/
- Descomprimir el .zip en la carpeta de trabajo /home/jose/
- Descargar el .war para su instalación en apache-tomcat, en http://ws.apache.org/axis2/download/1_3/download.cgi

Rampart:

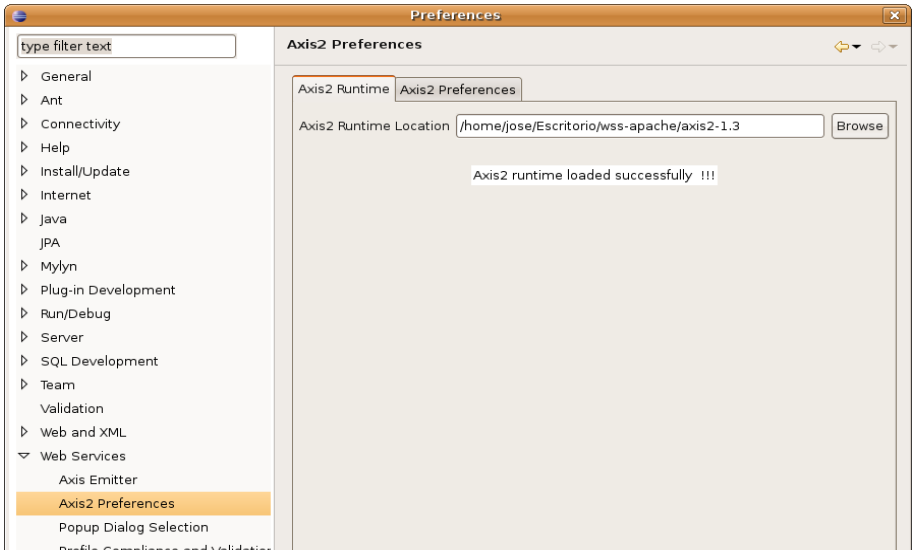
- Descargar la versión 1.3 de la web: http://ws.apache.org/axis2/modules/index.html
- Descomprimir el .zip en la carpeta de trabajo /home/jose/
- Copiar los .jar de la carpeta /lib en la carpeta de axis2: axis2-1.3/lib/
- Copiar los .mar (módulos) en la carpeta de axis2: axis2-1.3/repository/modules/
- Modificar el fichero axis2-1.3/repository/modules/modules.list añadiendo los módulos incorporados.

Tcpmon:

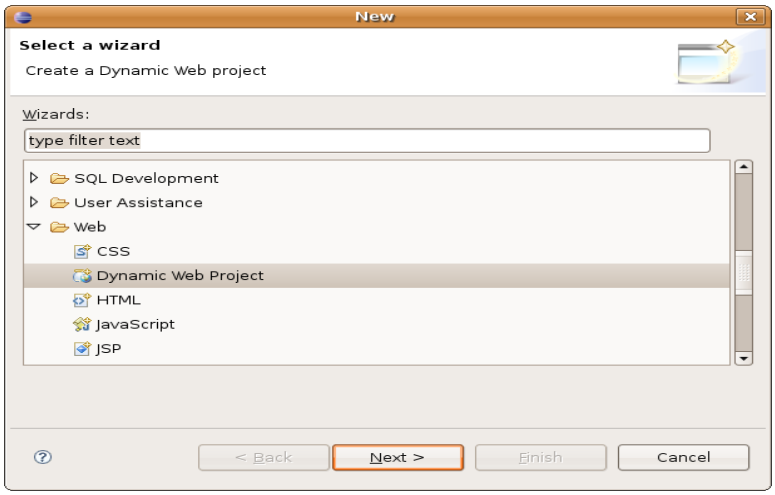
- Descargar la versión 1.0 de la web: http://ws.apache.org/commons/tcpmon/download.cgi
- Descomprimir el .zip en la carpeta de trabajo /home/jose/

Eclipse:

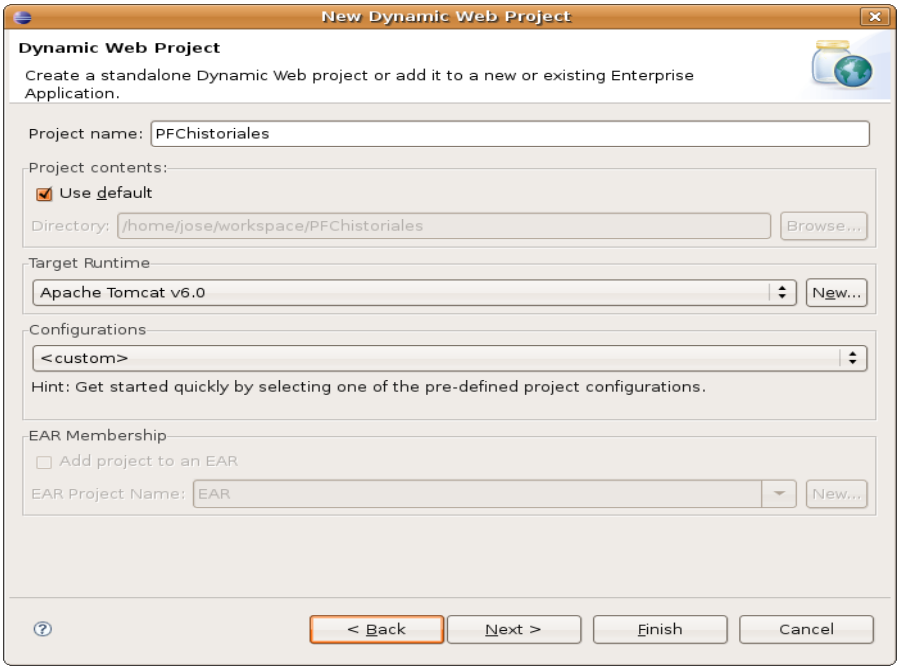
- Descargar la versión 3.3.2: **Eclipse IDE for Java EE Developers**
- Descomprimir el .gz en la carpeta de trabajo /home/jose/
- Configurar Eclipse para que trabaje con axis2 (Window/Preferences...):



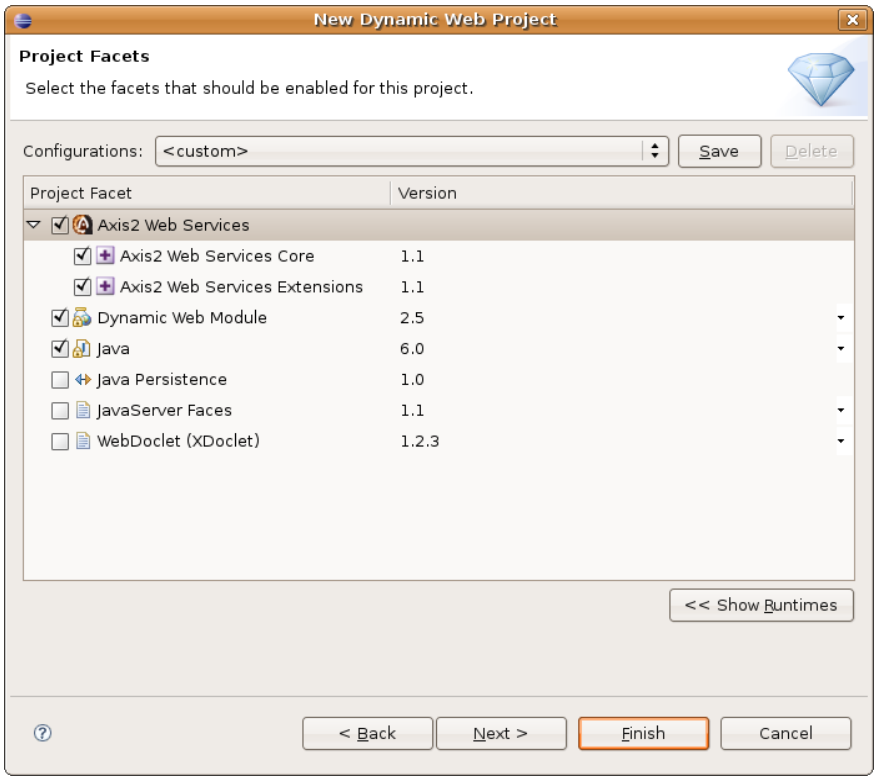
- Para la parte servidor, crear un nuevo proyecto PFChistoriales



- El proyecto utilizará el servidor Apache-Tomcat 6.0



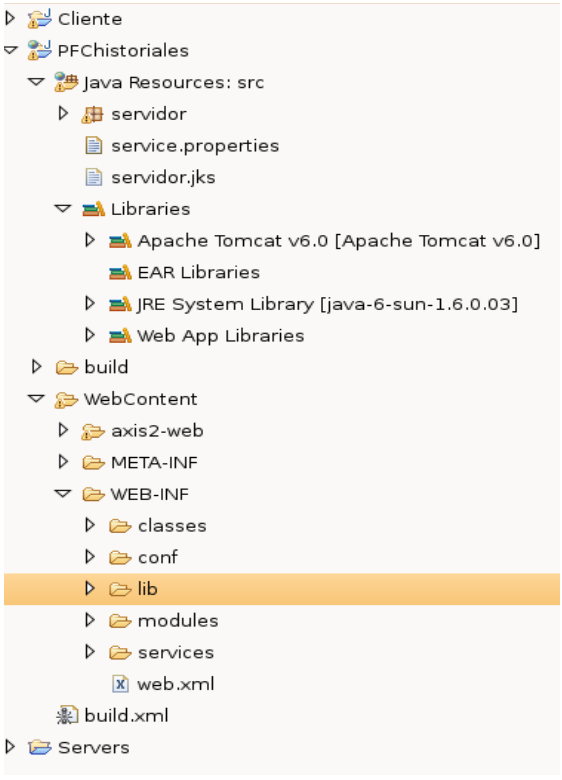
- Se incluyen las facet de Axis2:



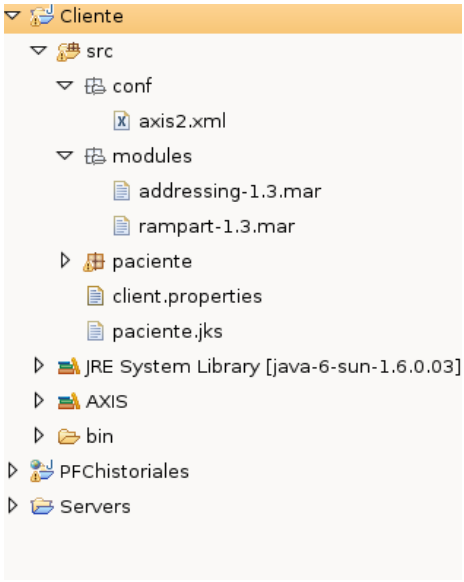
- Lo anterior no es suficiente; si se arranca el proyecto sobre el servidor, se produce un error al intentar cargar el módulo rampart.

```
INFO: Starting Servlet Engine: Apache Tomcat/6.0.16
[INFO] Deploying module: ping-1.3
[INFO] Deploying module: script-1.3
[INFO] Deploying module: addressing-1.3
[ERROR] The rampart-1.3.mar module, which is not valid, caused
org.apache.rampart.Rampart
...
```

Para solucionarlo, hay que importar los .jar de la librería de rampart a la carpeta /lib:

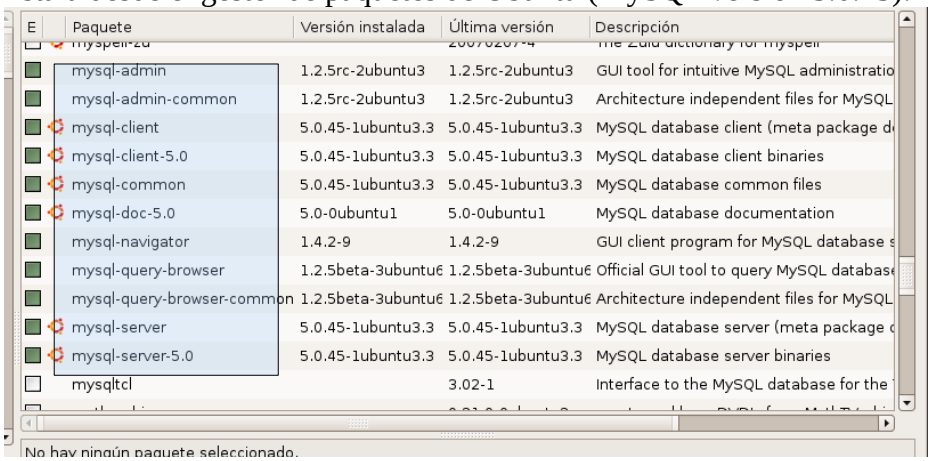


- Para la parte cliente, se crea un nuevo proyecto Java, Cliente , con la estructura de carpetas que se muestra a continuación:



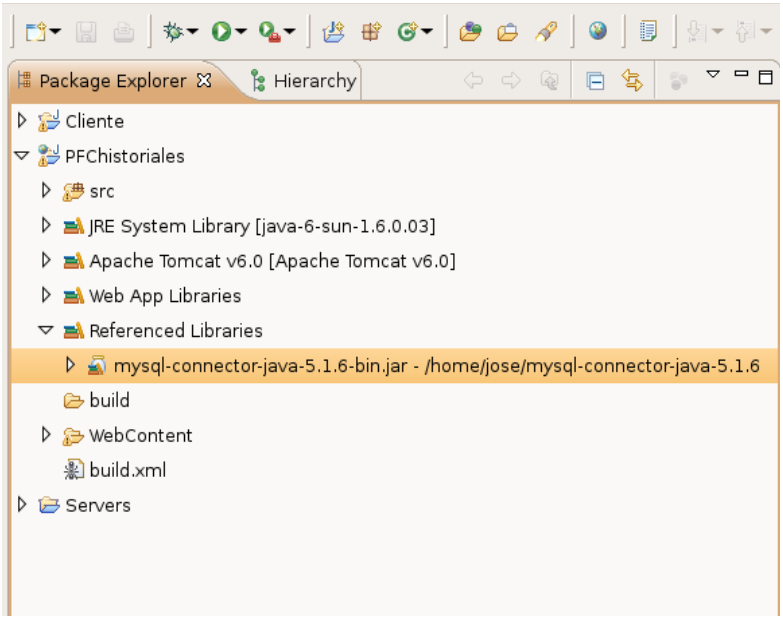
MySql:

- Se instala desde el gestor de paquetes de Ubuntu (MySQL versión 5.0.45):

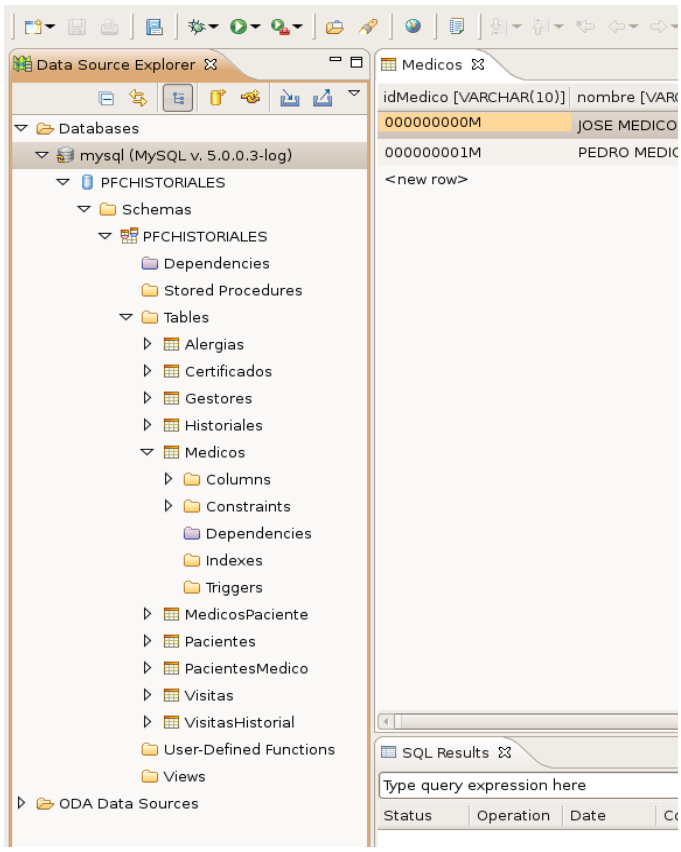


MySQL connector (versión 5.1.6):

- Descarga de la web: http://dev.mysql.com/downloads/connector/j/5.1.html
- Instalación en la carpeta de trabajo /home/jose/
- Preparación de Eclipse para utilizar la BD.
  - Añadir a las librerías utilizadas por el proyecto:



- Se puede utilizar el explorador incorporado a Eclipse para ver los datos de la base de datos:



- Copia del conector en la carpeta /lib del servidor de aplicaciones Tomcat

Jigloo (versión 4.0.5):

- Descarga desde la web (http://www.cloudgarden.com/jigloo/)
- Descomprimir y copiar en la carpeta /plugins de eclipse

Anexo IV. Creación Base de Datos y Tablas.

```
CREATE DATABASE IF NOT EXISTS PFCHISTORIALES;
GRANT SELECT,INSERT,UPDATE,DELETE,CREATE,DROP ON PFCHISTORIALES.* TO
'servidor'@'localhost' IDENTIFIED BY 'servidor';
USE PFCHISTORIALES;
CREATE TABLE Pacientes (
        idPaciente VARCHAR(10),
        nombre VARCHAR(30) NOT NULL,
        direccion VARCHAR(30) NOT NULL,
        telefono VARCHAR(13),
        fechaNacimiento DATE NOT NULL,
        nSS VARCHAR(18) NOT NULL,
        fechaAlta DATE NOT NULL,
        fechaBaja DATE,
        PRIMARY KEY (idPaciente)
        ) engine=InnoDB;
CREATE TABLE Medicos (
        idMedico VARCHAR(10),
        nombre VARCHAR(30) NOT NULL,
        direccion VARCHAR(30) NOT NULL,
        telefono VARCHAR(13),
        fechaNacimiento DATE NOT NULL,
        especialidad VARCHAR(30) NOT NULL,
        nColegiado VARCHAR(18) NOT NULL,
        fechaAlta DATE NOT NULL,
        fechaBaja DATE,
        PRIMARY KEY (idMedico)
        ) engine=InnoDB;
CREATE TABLE Gestores (
        idGestor VARCHAR(10),
        nombre VARCHAR(30) NOT NULL,
        direccion VARCHAR(30) NOT NULL,
        telefono VARCHAR(13),
        fechaNacimiento DATE NOT NULL,
        categoria VARCHAR(30) NOT NULL,
        puesto VARCHAR(30) NOT NULL,
```

```
        fechaAlta DATE NOT NULL,
        fechaBaja DATE,
        PRIMARY KEY (idGestor)
        ) engine=InnoDB;
CREATE TABLE Historiales (
        idHistorial VARCHAR(10),
        grupoSanguineo CHAR(2),
        PRIMARY KEY (idHistorial),
        FOREIGN KEY (idHistorial)
        REFERENCES Pacientes(idPaciente) ON DELETE RESTRICT
        ) engine=InnoDB;
CREATE TABLE Alergias (
        idHistorial VARCHAR(10) NOT NULL,
        alergia VARCHAR(30) NOT NULL,
        PRIMARY KEY (idHistorial,alergia)
        ) engine=InnoDB;
CREATE TABLE Visitas (
        idVisita CHAR(20),
        idMedico VARCHAR(10) NOT NULL,
        tema VARCHAR(50) NOT NULL,
        anamnesi TEXT,
        diagnostico TEXT,
        tratamiento TEXT,
        fecha DATE NOT NULL,
        PRIMARY KEY (idVisita),
        FOREIGN KEY (idMedico)
                REFERENCES Medicos(idMedico) ON DELETE RESTRICT
        ) engine=InnoDB;
CREATE TABLE Certificados (
        idCertificado VARCHAR (20),
        certificado TEXT NOT NULL,
        idUsuario VARCHAR (10) NOT NULL,
        fechaRevocacion DATE,
        PRIMARY KEY (idCertificado)
        ) engine=InnoDB;
CREATE TABLE VisitasHistorial (
        idHistorial VARCHAR (10),
        listaVisitasPaciente BLOB,
```

```
        firmaLista BINARY(128),
        claveSesion BINARY(128),
        PRIMARY KEY (idHistorial),
        FOREIGN KEY (idHistorial)
                REFERENCES Historiales(idHistorial) ON DELETE RESTRICT
        ) engine=InnoDB;
CREATE TABLE MedicosPaciente (
        idPaciente VARCHAR (10),
        listaMedicosPaciente BLOB,
        firmaLista BINARY(128),
        claveSesion BINARY(128),
        PRIMARY KEY (idPaciente),
        FOREIGN KEY (idPaciente)
                REFERENCES Pacientes(idPaciente) ON DELETE RESTRICT
        ) engine=InnoDB;
CREATE TABLE PacientesMedico (
        idMedico VARCHAR (10),
        listaPacientesMedico BLOB,
        firmaLista BINARY(128),
        claveSesion BINARY(128),
        PRIMARY KEY (idMedico),
        FOREIGN KEY (idMedico)
                REFERENCES Medicos(idMedico) ON DELETE RESTRICT
        ) engine=InnoDB;
CREATE TABLE Auditoria (
        idEntrada INT NOT NULL AUTO_INCREMENT,
        idActor VARCHAR(10) NOT NULL,
        fecha DATE NOT NULL,
        mensaje BLOB NOT NULL,
        autorizado BOOLEAN NOT NULL,
        firmaMensaje BINARY(128) NOT NULL,
        claveSesion BINARY(128) NOT NULL,
        PRIMARY KEY (idEntrada)
        ) engine=InnoDB;
```

Anexo V. Líneas de código relevantes.

Clase CriptoSigner (se muestra código perteneciente a su constructor). Es una clase que contiene métodos auxiliares de cifrado, firma y hash:

```
try {
        //cargamos el proveedor criptográfico
        BC = Class.forName( "org.bouncycastle.jce.provider.BouncyCastleProvider");
        Security.insertProviderAt((Provider)BC.newInstance(), 2);
        //preparamos el keystore
        KeyStore ks = KeyStore.getInstance(keystoreType);
        FileInputStream fis = new FileInputStream(keyStore);
        ks.load(fis, passwordKeyStore.toCharArray());
        //leemos la clave privada
        keyPrivadaRSA=(PrivateKey)
        ks.getKey(propioKeyAlias,propioKeyPass.toCharArray());
        //leemos nuestro certificado
        xcertPropio=(X509Certificate) ks.getCertificate(propioKeyAlias);
        //si se pide, el certificado del servidor
        if(ajenoCertAlias!=null) xcertAjeno=(X509Certificate)
                ks.getCertificate(ajenoCertAlias);
} catch (Exception e) {


}
```

Análisis de los resultados del procesamiento de la cabecera de los mensajes (clase ServiciosGestionGestores)

```
X509Certificate cert=null;
Timestamp ts=null;
byte [] firma=null;
//obtenemos el contexto del mensaje
MessageContext incomingContext = MessageContext.getCurrentMessageContext();
//vector con los resultados de procesado de la cabecera
Vector result =(Vector) incomingContext.getProperty(WSHandlerConstants.RECV_RESULTS);
for (int i = 0; i < result.size(); i++) {
        WSHandlerResult res = (WSHandlerResult) result.get(i);
        for (int j = 0; j < res.getResults().size(); j++){
                WSSecurityEngineResult secRes = (WSSecurityEngineResult)
                res.getResults().get(j);

                int action  =((Integer)
                secRes.get(WSSecurityEngineResult.TAG_ACTION)).intValue();

                // SIGNATURE
                if( ( action & WSConstants.SIGN ) > 0 ){
                        //certificado usado para firma
                        cert
                        =(X509Certificate)secRes.get(WSSecurityEngineResult.TAG_X509_CER
                        TIFICATE);
                        //firma
                        firma=(byte
                        [])secRes.get(WSSecurityEngineResult.TAG_SIGNATURE_VALUE);
                }
                //TIMESTAMP
                if( ( action & WSConstants.TS ) > 0 ){
                        //timestamp
                        ts=(Timestamp)secRes.get(WSSecurityEngineResult.TAG_TIMESTAMP);

                }
        }
}
```

Método para añadir una visita a un paciente (clase FachadaPacientesBD):

```
//Recibe como parámetros la visita y el paciente a incorporar la visita
public String anyadeVisitaPaciente(Visita visita, String idPaciente) throws Exception{

	Connection con = null;
	Statement s = null;
	PreparedStatement stmt=null;
	//obtenemos las visitas del paciente
	Visita [] visitas=consultaVisitasPaciente(idPaciente);

	String cadena=null;
	//si no tiene visitas, creamos de cero un Historial
	if (visitas==null || visitas.length==0){
		cadena="<Historial><Paciente>"+idPaciente+"</Paciente><Visitas>"+
			"<Visita>"+visita.getIdVisita()+"</Visita></Visitas></Historial>";
	}else{
		//añadimos la visita a las que ya existían
		StringBuffer sb=new StringBuffer();
		sb.append("<Historial><Paciente>"+idPaciente+"</Paciente><Visitas>");
		for(int i=0;i<visitas.length;i++){
			sb.append("<Visita>"+visitas[i].getIdVisita()+"</Visita>");
		}
		sb.append("<Visita>"+visita.getIdVisita()+"</Visita></Visitas></Historial>");
		cadena=sb.toString();
	}

	try{
		//cs es una instancia de la clase CriptoSigner
		//canonicalizamos la cadena
		byte[] s2=cs.c14n(cadena.getBytes());
		//obtenemos la firma de la cadena (lista de visitas) ya c14n
		byte [] firma=cs.firma(s2, "SHA1withRSA");
		//generamos una clave de sesión AES128
		SecretKey sk=cs.getSecretKeyAES(128);
		byte [] clave=sk.getEncoded();
		//ciframos la clave con el algo RSA
		byte [] claveC=cs.cifraRSA(clave, "RSA/ECB/PKCS1Padding");
		//ciframos la cadena inicial con la clave simétrica
		byte [] listaC=cs.cifra(s2, sk, "AES");
		//obtenemos una conexión
		con=bd.getConexion();
		//como habrá varias transacciones, se desactiva el autocommit
		con.setAutoCommit(false);
		//creamos un Blob
		Blob newBlob = con.createBlob();
		//se guarda en el Blob
		newBlob.setBytes(1,listaC);
		s=con.createStatement();
		//borramos las visitas antiguas
		s.executeUpdate("delete from VisitasHistorial where idHistorial='"+idPaciente
		+"'");
		//se prepara un Statement para la actualización
		stmt = con.prepareStatement("INSERT INTO VisitasHistorial
		(idHistorial,listaVisitasPaciente,firmaLista,claveSesion) VALUES(?,?,?,?)");
		//se situan los parámetros del Stetement
		stmt.setString(1,idPaciente);
		stmt.setBlob(2, newBlob);
		stmt.setBytes(3,firma );
		stmt.setBytes(4,claveC );
		//se ejecuta
		stmt.executeUpdate();
		stmt.close();
		//se obtienen Clobs para cadenas de texto largas
		Clob anamnesi=con.createClob();
		anamnesi.setString(1,visita.getAnamnesi());
		Clob diagnostico=con.createClob();
		diagnostico.setString(1, visita.getDiagnosis());
		Clob tratamiento=con.createClob();
		tratamiento.setString(1, visita.getTratamiento());
```

```
                //se prepara un nuevo Statement con la nueva inserción en la tabla de visitas
                stmt = con.prepareStatement("INSERT INTO Visitas
                (idVisita,idMedico,tema,anamnesi,diagnostico," +
                        "tratamiento,fecha) VALUES(?,?,?,?,?,?,?)");
                stmt.setString(1,visita.getIdVisita());
                stmt.setString(2, visita.getMedico().getId());
                stmt.setString(3,visita.getTema());
                stmt.setClob(4,anamnesi );
                stmt.setClob(5,diagnostico );
                stmt.setClob(6,tratamiento );
                stmt.setDate(7, new java.sql.Date(visita.getFecha().getTime()));
                stmt.executeUpdate();
                stmt.close();
                //hacemos commit final
                con.commit();
        }catch(Exception e){
                //Si hay error, y no es propio, lo relanzamos
                if (e instanceof MiExcepcion) throw e; else throw new Exception();
        }finally{
                //se cierra la conexión
                bd.closeConnection(con);
        }
        return "ok";
}
```

Verificación de la autenticidad de un mensaje auditado:

1. En la etapa de auditoría se guarda completo el mensaje SOAP, al que se le ha extraído el elemento xenc:EncryptedKey, que no será necesario a efectos de verificación de firma:

```
//obtenemos el mensaje SOAP
SOAPEnvelope se=incomingContext.getEnvelope();
//obtenemos una referencia al elemento a eliminar
OMElement om1=((OMElement)
        (se.getHeader().getChildElements().next())).getFirstChildWithName(new
        QName("http://www.w3.org/2001/04/xmlenc#","EncryptedKey","xenc"));
//desvinculamos el elemento del mensaje SOAP
om1.detach();
//convertimos el mensaje en una cadena
String mensaje=se.toString();

...
//auditamos el mensaje (y su autor)
if (audita)
        auditor.audita(mensaje,new
        X500Name(cert.getSubjectX500Principal().getEncoded()).getDNQualifier());
```

2. Para auditoría se recupera el mensaje, se convierte en un Documento y se procesa su cabecera:

```
//vector con los resultados del procesamiento
Vector v=null;
//motor que procesará la cabecera
WSSecurityEngine wsse=new WSSecurityEngine();
//archivo con propiedades de seguridad
Properties properties = new Properties();
try {
        //leemos propiedades
        properties.load(new FileInputStream("service.properties"));
        //procesamos la cabecera
        v=wsse.processSecurityHeader(getDOM(mensaje.getBytes()),null,new
        PWCBHandler(),CryptoFactory.getInstance(properties));
}catch (Exception e){
        //si fallo en la cabecera, devueve false
        return false;
}
//todo correcto
return true;
```

Constructor del cliente de WS (clase PacienteWSCliente):

```
        //fichero de propiedades de configuracion
        Properties properties = new Properties();
        try {
                properties.load(new FileInputStream("bin/entorno.properties"));
        } catch (IOException e) {
                throw new Exception ("ERROR: fallo en lectura variables entorno");
        }
        //URL en las que se encuentran los servicios web
        dirServGesPac=properties.getProperty("dirServ1");
        dirServGesGes=properties.getProperty("dirServ2");

        String dirConf="bin";
        try {
                //obtenemos la configuracion del cliente
                ConfigurationContext ctx =
                ConfigurationContextFactory.createConfigurationContextFromFileSystem(dirConf,
                dirConf + "/conf/axis2.xml");
                //creamos el cliente
                client = new ServiceClient(ctx, null);
        } catch (AxisFault e) {
                throw new Exception ("ERROR: fallo en creacion WS cliente");
        }
```

Invocación desde el cliente de operaciones en el WS (clase PacienteWSCliente):

```
//metodo que consulta los datos de un paciente
public Paciente consultaDatosPaciente(String idPaciente) throws Exception{
        //creamos las opciones
        Options options = new Options();
        //definimos la accion a invocar
        options.setAction("urn:consultaDatosPaciente");
        //fijamos la URL en la que se encuentra el servicio
        options.setTo(new EndpointReference(dirServGesPac));
        client.setOptions(options);
        //obtenemos la respuesta del servidor
        OMElement response =
        client.sendReceive(getPayload("consultaDatosPaciente",idPaciente));
        //convertimos la respuesta en un Paciente, con ayuda del OMElizador
        Paciente p=(Paciente)new OMElizador().desomeliza(Paciente.class, response);
        return p;

    }
```

Creación del payload a transportar por los mensajes de invocación (clase PacienteWSCliente):

```
//toma la acción + una serie de parámetros
private static OMElement getPayload(String accion,String[] value) {
        //factoría de elementos OMElement
        OMFactory factory = OMAbstractFactory.getOMFactory();
        //namespace propio
        OMNamespace ns = factory.createOMNamespace("http://PFC_seguridad.uoc.edu/xsd/","ns1");
        //elemento creado
        OMElement elem = factory.createOMElement(accion, ns);
        for (int i=0;i<value.length;i++){
                //parámetros creados
                OMElement childElem = factory.createOMElement("param"+i, null);
                childElem.setText(value[i]);
                elem.addChild(childElem);
         }
         return elem;
}
```

Clase PWCBHandler:

```
//clase que maneja la clave privada del servidor
public class PWCBHandler implements CallbackHandler {

        //clave que se fija cuando el usuario accede al keystore
        public static String clave=null;

        public void handle(Callback[] callbacks) throws IOException,
             UnsupportedCallbackException {
                for (int i = 0; i < callbacks.length; i++) {

                        WSPasswordCallback pwcb = (WSPasswordCallback)callbacks[i];
                        String id = pwcb.getIdentifer();
                        //fija la clave de acceso
                        if("gestor".equals(id) ||"medico".equals(id) || "paciente".equals(id)) {
                                pwcb.setPassword(PWCBHandler.clave);
                        } else if("servidor".equals(id)) {
                                pwcb.setPassword(PWCBHandler.clave);
                        }
                }
        }

}
```

Clase OMElizador:

1. Método que obtiene un OMElement de un objeto:

```
public OMElement omeliza(Object o){
        //obtiene un "reader" del objeto
        XMLStreamReader reader = BeanUtil.getPullParser(o);
        StreamWrapper parser = new StreamWrapper(reader);
        StAXOMBuilder stAXOMBuilder =
        OMXMLBuilderFactory.createStAXOMBuilder(
        OMAbstractFactory.getOMFactory(), parser);
        //genera el OMElement
        OMElement element = stAXOMBuilder.getDocumentElement();
        return element;
}
```

2. Método inverso:

```
public Object desomeliza(Class c,OMElement ome) throws AxisFault{
        return BeanUtil.deserialize(c, ome, new DefaultObjectSupplier(), null);
}
```

Clase InicializadorListener; se trata de un Listener que se ejecuta al arrancar Axis2, con lo que los objetos que se introducen en el contexto de axis2 no necesitan ser instanciados más veces, obteniéndose una importante ganancia de eficiencia:

```
public void contextInitialized(ServletContextEvent arg0) {

        //objetos situados en el contexto de la aplicación
        ctx=arg0.getServletContext();
        ctx.setAttribute("conexionBD", new BDConexion());
        ctx.setAttribute("criptoSigner",
                new CriptoSigner("JKS","servidor",null,"uoc0506","uoc0506",
                "./servidor.jks"));
        ctx.setAttribute("integracionGestores", new FachadaGestoresBD(ctx));
        ctx.setAttribute("integracionPacientes", new FachadaPacientesBD(ctx));
        ctx.setAttribute("omelizador", new OMElizador());
        ctx.setAttribute("gestorGestores",new GestorGestores(ctx));
        ctx.setAttribute("gestorPacientes",new GestorPacientes(ctx));
        ctx.setAttribute("auditor",new Auditor(ctx));
}
```

Auditar validez de un mensaje (clase Auditor):

```
//audita un mensaje; devuelve true si ok; false ni no correcto
public boolean auditaValidezMensaje(String mensaje) throws Exception {

	Vector v=null;
	//engine que procesará la cabecera
	WSSecurityEngine wsse=new WSSecurityEngine();
	Properties properties = new Properties();
	try {
		//carga las propiedades de la clase Crypto
		properties.load(new
		FileInputStream("/home/jose/workspace/PFChistoriales/src/service.properties"));
		//procesa la cabecera y guarda los resultados en el Vector
		v=wsse.processSecurityHeader(getDOM(mensaje.getBytes()),null,new PWCBHandler(),
				CryptoFactory.getInstance(properties));
	}catch (Exception e){
		return false;
	}
		return true;
	}
```

Creación identificador de visitas (clase GestorPacientes):

```
//obtenemos un array de 16 bytes
byte [] random=new byte[16];
//generamos un array de 16 bytes aleatorios
new Random().nextBytes(random);
//fijamos el id codificando en base 64 el array aleatorio
String idVisita=Base64.encode(random);
```

Anexo VI. Instalando y ejecutando la aplicación.

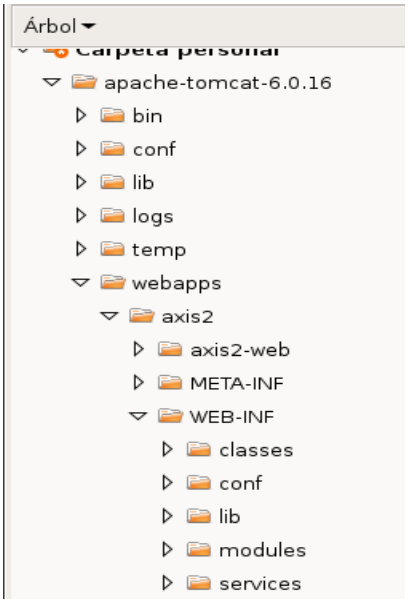
Partimos del entregable *PFC_VazquezAraujoJoseManuel.jar* descomprimido en una carpeta de trabajo (ejemplo: */home/jose/*).

Tenemos dos componentes:

- **Servidor:**
  - Preparamos la plataforma de ejecución:
    1. Punto de partida: apache-tomcat-6.0.16 desempaquetado (o instalado) en el directorio de trabajo (ejemplo /home/jose/)
    2. Tenemos axis2.war (descargado de http://ws.apache.org/axis2/0_94/installationguide.html#Download_Axis2).

       Se descomprime (es necesario descomprimirlo para tocar varios de sus componentes) en la carpeta de Apache:

       apache-tomcat-6.0.16/webapps/
    3. Quedando apache así:



    4. Arrancamos Apache con Axis2 instalado:

       jose@jose-desktop:~/apache-tomcat-6.0.16/bin$ ./startup.sh

       Using CATALINA_BASE: /home/jose/apache-tomcat-6.0.16

       Using CATALINA_HOME: /home/jose/apache-tomcat-6.0.16

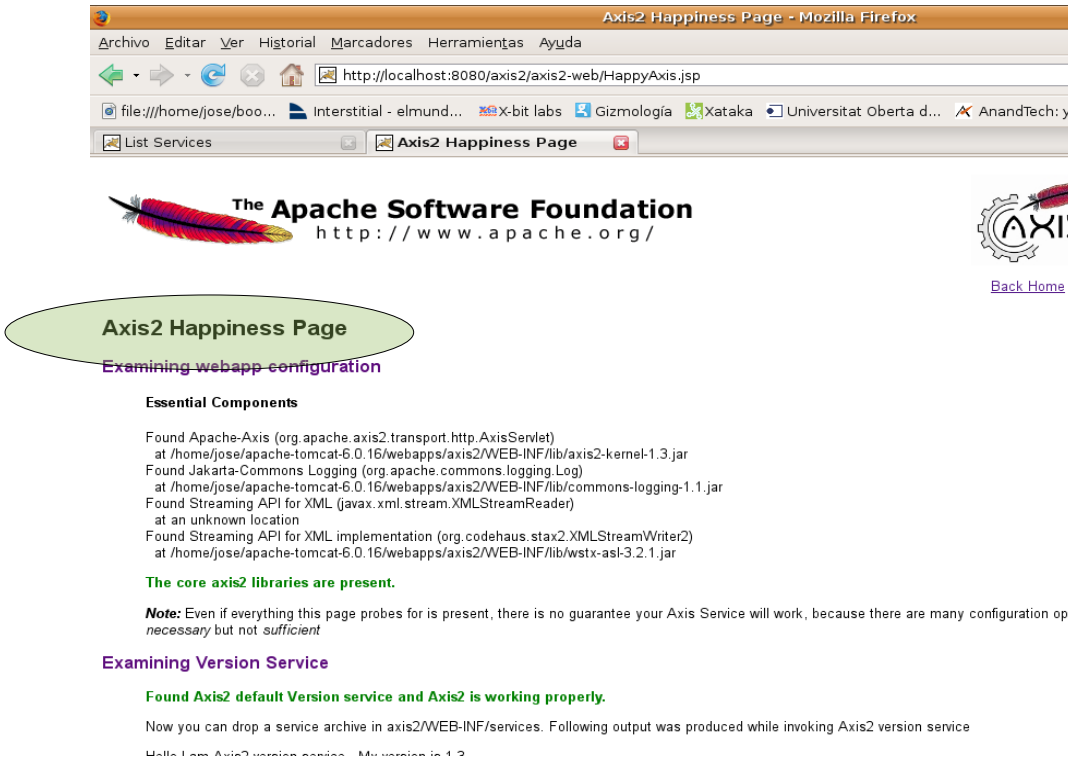
       Using CATALINA_TMPDIR: /home/jose/apache-tomcat-6.0.16/temp

       Using JRE_HOME: /home/jose/jdk1.6.0_03

jose@jose-desktop:~/apache-tomcat-6.0.16/bin$

5. Podemos verificar con el navegador que axis2 está felizmente arrancado, en la dirección:

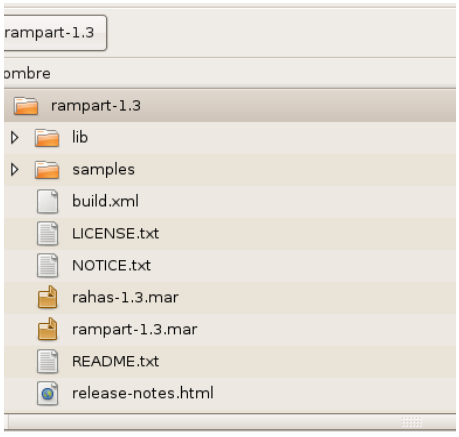
   http://localhost:8080/axis2/axis2-web/HappyAxis.jsp



6. El .war de Axis2 no contiene el módulo de seguridad Rampart-1.3, que hay que descargar de la dirección:

   http://ws.apache.org/axis2/modules/index.html

7. Descomprimimos el fichero descargado rampart-1.3:



8. Las librerías .jar contenidas en la carpeta /lib de *rampart-1.3/* han de ser copiadas en

   apache-tomcat-6.0.16/webapps/axis2/WEB-INF/lib/

9. Los módulos (ficheros con extensión .mar) contenidos en *rampart-1.3/* han de ser copiados en:

   apache-tomcat-6.0.16/webapps/axis2/WEB-INF/modules/

10. Instalamos el driver de MySQL (descargado en http://dev.mysql.com/downloads/connector/j/5.1.html) en apache-tomcat; copiamos el archivo mysql-connector-java-5.1.6-bin.jar en la carpeta:

    apache-tomcat-6.0.16/lib/

Con esto ya está Apache-Tomcat preparado para trabajar con Axis2 y Rampart!

- Instalamos los componentes de la aplicación construida, que se encuentran en la carpeta del entregable, *ejecutables/ServiciosPFCHistoriales/* en Apache-Tomcat:

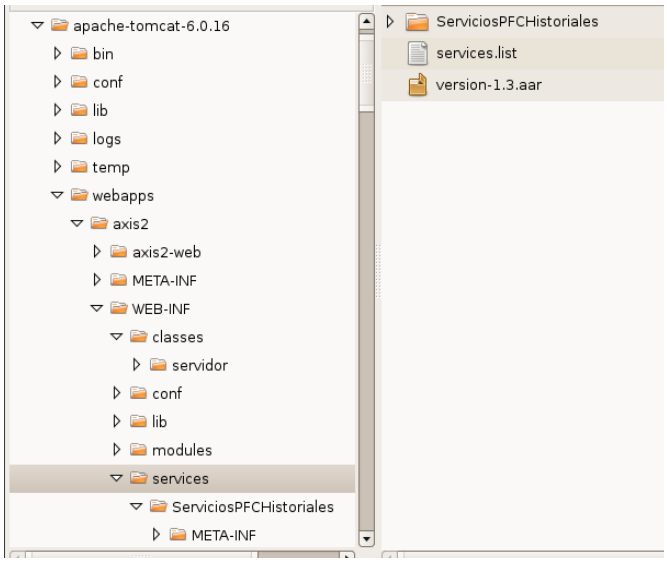
  1. Hay que copiar la carpeta ServiciosPFC*Historiales/* en:

     apache-tomcat-6.0.16/webapps/axis2/WEB-INF/services/

  2. De la carpeta anterior, *ServiciosPFCHistoriales/* nos llevamos la carpeta *servidor/* (contiene las clases compiladas) a la carpeta

     apache-tomcat-6.0.16/webapps/axis2/WEB-INF/classes/

     Quedando así:



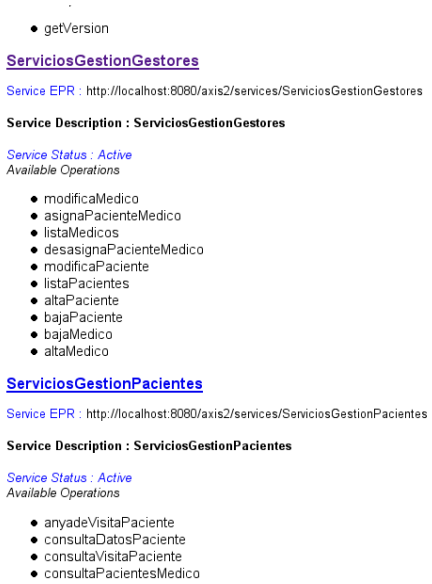
3. Modificamos el fichero web.xml de axis2 (situado en: *apache-tomcat-6.0.16/webapps/axis2/WEB-INF/web.xml*) de axis2 para incorporar, colgado del elemento `<web-app>` el elemento:

```
<listener>
     <listener-class>servidor.ws.IniciadorListener</listener-class>
</listener>
```

   Se trata del componente de la aplicación que inicializa otros componentes.

4. Reiniciamos Apache-Tomcat y comprobamos que los servicios nuevos están instalados correctamente en:

   http://localhost:8080/axis2/services/listServices



- getVersion

ServiciosGestionGestores

Service EPR : http://localhost:8080/axis2/services/ServiciosGestionGestores

Service Description : ServiciosGestionGestores

Service Status : Active
Available Operations

- modificaMedico
- asignaPacienteMedico
- listaMedicos
- desasignaPacienteMedico
- modificaPaciente
- listaPacientes
- altaPaciente
- bajaPaciente
- bajaMedico
- altaMedico

ServiciosGestionPacientes

Service EPR : http://localhost:8080/axis2/services/ServiciosGestionPacientes

Service Description : ServiciosGestionPacientes

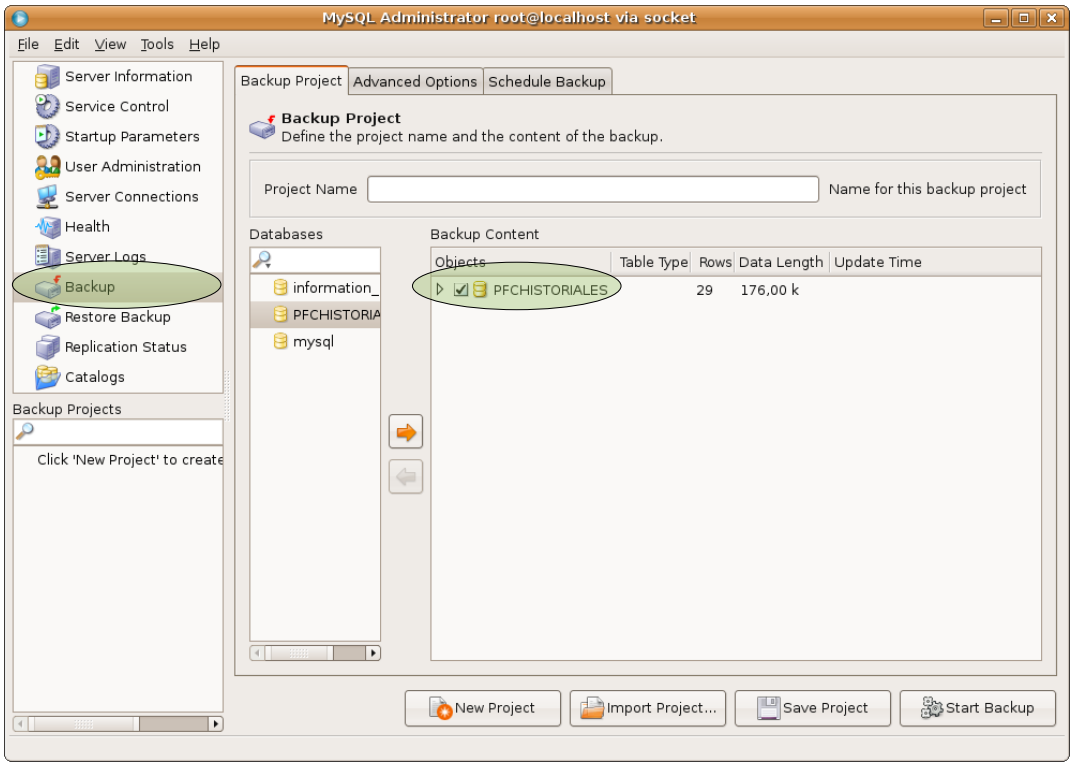
Service Status : Active
Available Operations

- anyadeVisitaPaciente
- consultaDatosPaciente
- consultaVisitaPaciente
- consultaPacientesMedico

Con esto ya está nuestra aplicación instalada y funcionando (a falta de la BD)!

- Preparamos la base de datos:

1. La mejor opción es partir de un back_up de la base de datos, que se adjunta en el entregable del proyecto, con el nombre _20080516_2011.sql en la carpeta *baseDatos/*. En realidad se trata de un dump de la base de datos, por lo que es fácil crearlo, con la herramienta MySQL Administrator:



Y recuperar el back-up adjunto desde la misma herramienta.

2. El archivo que configura la conexión de la aplicación de la BD está en:

apache-tomcat-6.0.16/webapps/axis2/WEB-INF/services/propiedades.properties

Se ha definido la url y el usuario de acceso siguiente (que habrá que cambiar si modificamos la URL o el usuario de la BD):

dirBD=jdbc:mysql://localhost:3306/PFCHISTORIALES

usuarioBD=servidor

claveBD=servidor

Con esto ya está nuestra aplicación instalada y funcionando, incluida la BD!

Segundo componente:

- **Cliente:**
  1. Entrar en la carpeta *PFCHistorialesCliente/*
  2. Copiar en la carpeta *PFCHistorialesCliente/libs/* las librerías de axis2 y de rampart (las tenemos ya en *apache-tomcat-6.0.16/webapps/axis2/WEB-INF/lib/*)
  3. La configuración del cliente, en cuanto a la localización de los servicios se encuentra en el fichero *ejecutables/entorno.properties*. Si se quiere utilizar tcpmon para capturar los mensajes, habrá que modificar el puerto por el configurado en el programa.

     #VARIABLES DE ENTORNO CLIENTE

     dirServ1=http://localhost:8080/axis2/services/ServiciosGestionPacientes

     dirServ2=http://localhost:8080/axis2/services/ServiciosGestionGestores

Con esto ya está nuestra aplicación cliente instalada!

- **Para ejecutar el cliente:**

  Entrar en la carpeta *PFCHistorialesCliente/* y en línea de comandos ejecutar:

```
jose@jose-desktop:~/Escritorio/PFCHistorialesCliente$ java -jar PFCHistoriales.jar Gestor
```

Tenemos un usuario de prueba con su keystore con el rol de Gestor:

usuario: 00000000-G

clave: uoc0506

keystore: (en la propia carpeta, no elegible) gestor.jks

```
jose@jose-desktop:~/Escritorio/PFCHistorialesCliente$ java -jar PFCHistoriales.jar Medico
```

Tenemos un usuario con su keystore con el rol de Medico:

usuario: 00000000-M

clave: uoc0506

keystore: (en la propia carpeta, no elegible) medico.jks

```
jose@jose-desktop:~/Escritorio/PFCHistorialesCliente$ java -jar PFCHistoriales.jar Paciente
```

Tenemos un usuario con su keystore con el rol de Gestor:

usuario: 00000000-P

clave: uoc0506

keystore: (en la propia carpeta, no elegible) paciente.jks

Anexo VII. XML; sintaxis de los protocolos y ejemplos.

Sintaxis SOAP:

```
<soap:Envelope >
  <soap:Header>?
  <soap:Body>
    <soap:Fault>?
  </soap:Body>
</soap:Envelope>
```

Sintaxis XMLSignature:

```
<Signature ID?>
  <SignedInfo>
    <CanonicalizationMethod/>, ej. CanonicalXML
    <SignatureMethod/>, ej. DSA-SHA1  PCKS1
   (<Reference URI?>
     (<Transforms/>)?, ej. XSLT and XPath
      <DigestMethod/>, ej. SHA1
      <DigestValue/>
    </Reference>)+
  </SignedInfo>
  <SignatureValue/>
 (<KeyInfo/>)?
 (<Object ID?>)*
</Signature>
```

Sintaxis XMLEncryption:

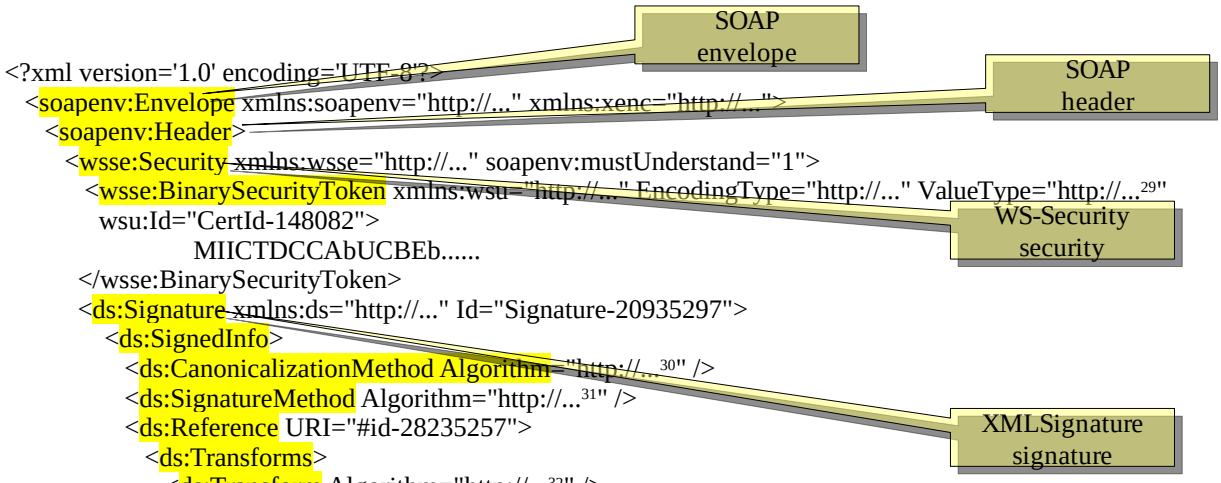
```
<EncryptedData Id? Type? MimeType? Encoding?>
  <EncryptionMethod/>?, ej. TRIPLEDES, AES-128, RSA
  <ds:KeyInfo>
    <EncryptedKey>?
    <AgreementMethod>?, ej. Diffie-Hellman
    <ds:KeyName>?
    <ds:RetrievalMethod>?
    <ds:*>?
  </ds:KeyInfo>?
  <CipherData>
    <CipherValue>?
    <CipherReference URI?>?
  </CipherData>
  <EncryptionProperties>?
</EncryptedData>
```

Sintaxis WS-Security:

```
<SOAP:Envelope>
 <SOAP:Header>
  <wsse:Security actor=      mustUnderstand=     >
   <wsse:BinarySecurityToken>?
   <ds:Signature>
    <ds:SignedInfo>
     (<ds:Reference>)+
    </ds:SingedInfo>
    <ds:SignatureValue>
    <ds:KeyInfo>?
   </ds:Signature>
   <xenc:EncryptedKey>
    <ds:KeyInfo>?
   </xenc:EncryptedKey>?
   <wsu:TimeStamp>?
  <wsse:Security>
 </SOAP:Header>
 <SOAP:Body>
  <xenc:EncryptedData>?
 </SOAP:Body>
</SOAP:Envelope>
```

Ejemplo de mensaje tipo utilizado, con los protocolos anteriores:

A continuación se muestra un mensaje utilizado por la aplicación, especificándose los estándares utilizados:




```
<?xml version='1.0' encoding='UTF-8'?>
  <soapenv:Envelope xmlns:soapenv="http://..." xmlns:xenc="http://...">
    <soapenv:Header>
      <wsse:Security xmlns:wsse="http://..." soapenv:mustUnderstand="1">
        <wsse:BinarySecurityToken xmlns:wsu="http://..." EncodingType="http://..." ValueType="http://...[29]"
          wsu:Id="CertId-148082">
                MIICTDCCAbUCBEb.....
        </wsse:BinarySecurityToken>
        <ds:Signature xmlns:ds="http://..." Id="Signature-20935297">
          <ds:SignedInfo>
            <ds:CanonicalizationMethod Algorithm="http://...[30]" />
            <ds:SignatureMethod Algorithm="http://...[31]" />
            <ds:Reference URI="#id-28235257">
              <ds:Transforms>
                <ds:Transform Algorithm="http://...[32]" />
              </ds:Transforms>
              <ds:DigestMethod Algorithm="http://...[33]" />
              <ds:DigestValue>
                  874VXIwSMw6JNkJecW28jh0Nks0=
              </ds:DigestValue>
            </ds:Reference>
```

29 http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509v3
30 http://www.w3.org/2001/10/xml-exc-c14n#
31 http://www.w3.org/2000/09/xmldsig#rsa-sha1
32 http://www.w3.org/2001/10/xml-exc-c14n#
33 http://www.w3.org/2000/09/xmldsig#sha1

```
        </ds:SignedInfo>
        <ds:SignatureValue>
                Tc2lPMGwwR....
        </ds:SignatureValue>
        <ds:KeyInfo Id="KeyId-14732323">
          <wsse:SecurityTokenReference xmlns:wsu="http://..." wsu:Id="STRId-8000886">
            <wsse:Reference URI="#CertId-148082" ValueType="http://..." />
          </wsse:SecurityTokenReference>
        </ds:KeyInfo>
      </ds:Signature>
      <xenc:EncryptedKey Id="EncKeyId-33320514">
        <xenc:EncryptionMethod Algorithm="http://...[34]" />
        <ds:KeyInfo xmlns:ds="http://...">
          <wsse:SecurityTokenReference>
            <wsse:KeyIdentifier EncodingType="http://..." ValueType="http://...">
                ZVwxpuQACcaFPU+WcbSYqhltNWg=
            </wsse:KeyIdentifier>
          </wsse:SecurityTokenReference>
        </ds:KeyInfo>
        <xenc:CipherData>
          <xenc:CipherValue>
                eWP9Bjag....
          </xenc:CipherValue>
        </xenc:CipherData>
        <xenc:ReferenceList>
          <xenc:DataReference URI="#EncDataId-28235257" />
        </xenc:ReferenceList>
      </xenc:EncryptedKey>
      <wsu:Timestamp xmlns:wsu="http://..." wsu:Id="Timestamp-32012057">
        <wsu:Created>
                2008-03-19T09:18:40.865Z
        </wsu:Created>
        <wsu:Expires>
                2008-03-19T09:23:40.865Z
        </wsu:Expires>
      </wsu:Timestamp>
    </wsse:Security>
  </soapenv:Header>
  <soapenv:Body xmlns:wsu="http://..." wsu:Id="id-28235257">
    <xenc:EncryptedData Id="EncDat337aId-28235257" Type="http://...">
      <xenc:EncryptionMethod Algorithm="http://...[35]" />
      <ds:KeyInfo xmlns:ds="http://...">
        <wsse:SecurityTokenReference xmlns:wsse="http://...  >
          <wsse:Reference URI="#EncKeyId-33320514" />
        </wsse:SecurityTokenReference>
      </ds:KeyInfo>
      <xenc:CipherData>
        <xenc:CipherValue>
                5DlHfD...
        </xenc:CipherValue>
      </xenc:CipherData>
    </xenc:EncryptedData>
  </soapenv:Body>
</soapenv:Envelope>
```

XMLEncryption encryptedKey

SOAP body

XMLEncryption encryptedData

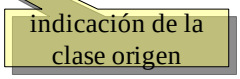
34 http://www.w3.org/2001/04/xmlenc#rsa-1_5

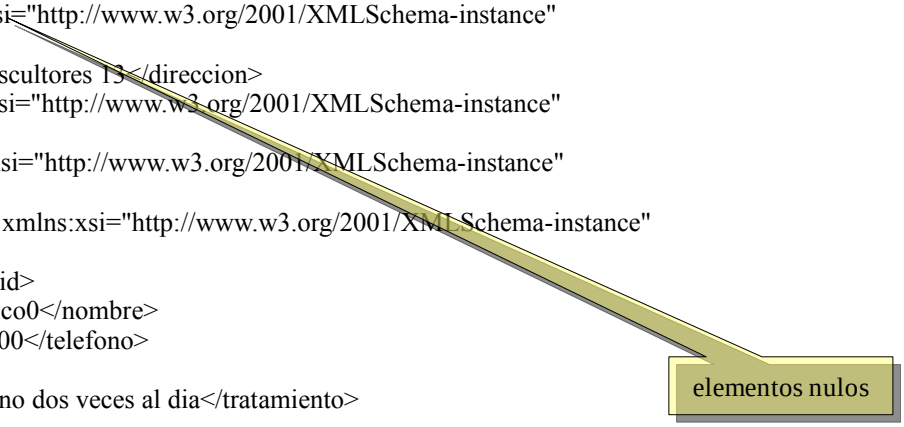
35 http://www.w3.org/2001/04/xmlenc#aes256-cbc

Anexo VIII. ADB; procesamiento de JavaBean.

A continuación se muestran ejemplos de cada uno de los JavaBean una vez transformados por ADB en XML:

```
<Paciente type="servidor.modelo.Paciente">
  <historial type="servidor.modelo.Historial">
    <alergias>aspirina</alergias>
    <alergias>polen</alergias>
    <grupoSanguineo>O+</grupoSanguineo>
    <visitas type="servidor.modelo.Visita">
      <anamnesi>enfermedades varias</anamnesi>
      <diagnosis>malito</diagnosis>
      <fecha>2008-04-06T14:55:20.108Z</fecha>
      <idVisita>v1</idVisita>
      <medico type="servidor.modelo.Medico">
        <especialidad>TRAUMATOLOGO</especialidad>
        <pacientes xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:nil="true" />
        <direccion>sector escultores 13</direccion>
        <fechaAlta xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:nil="true" />
        <fechaBaja xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:nil="true" />
        <fechaNaciemiento xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:nil="true" />
        <id>00000000-M</id>
        <nombre>jose medico0</nombre>
        <telefono>900000000</telefono>
      </medico>
      <tratamiento>ibuprofeno dos veces al dia</tratamiento>
    </visitas>
    <visitas type="servidor.modelo.Visita">
      <anamnesi>enfermo poc antes</anamnesi>
      <diagnosis>malito</diagnosis>
      <fecha>2008-04-06T14:55:20.108Z</fecha>
      <idVisita>v2</idVisita>
      <medico type="servidor.modelo.Medico">
        <especialidad>TRAUMATOLOGO</especialidad>
        <pacientes xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:nil="true" />
        <direccion>sector escultores 13</direccion>
        <fechaAlta xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:nil="true" />
        <fechaBaja xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:nil="true" />
        <fechaNaciemiento xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xsi:nil="true" />
        <id>00000000-M</id>
        <nombre>jose medico0</nombre>
        <telefono>900000000</telefono>
      </medico>
      <tratamiento>aspirina</tratamiento>
    </visitas>
  </historial>
```





```
    <medicos type="servidor.modelo.Asignacion">
      <asignador type="servidor.modelo.Gestor">
        <categoria>jefe</categoria>
        <puestoTrabajo>puesto1</puestoTrabajo>
        <direccion xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xsi:nil="true" />
        <fechaAlta xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xsi:nil="true" />
        <fechaBaja xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xsi:nil="true" />
        <fechaNaciemiento xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xsi:nil="true" />
        <id>00000000-G</id>
        <nombre>jose gestor0</nombre>
        <telefono>900000000</telefono>
      </asignador>
      <fin>2046-07-12T05:50:40.302Z</fin>
      <inicio>2008-04-06T14:55:20.151Z</inicio>
      <persona type="servidor.modelo.Medico">
        <especialidad>TRAUMATOLOGO</especialidad>
        <pacientes xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:nil="true" />
        <direccion>ninguna</direccion>
        <fechaAlta xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xsi:nil="true" />
        <fechaBaja xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xsi:nil="true" />
        <fechaNaciemiento xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xsi:nil="true" />
        <id>00000000-M</id>
        <nombre>jose medico0</nombre>
        <telefono>900000000</telefono>
      </persona>
    </medicos>
    <direccion>ninguna</direccion>
    <fechaAlta>2008-04-06T14:55:20.108Z</fechaAlta>
    <fechaBaja>2008-04-06T14:55:20.108Z</fechaBaja>
    <fechaNaciemiento>2008-04-06T14:55:20.108Z</fechaNaciemiento>
    <id>00000000-P</id>
    <nombre>jose paciente0</nombre>
    <telefono>900000000</telefono>
</Paciente>

<Envoltorio type="servidor.modelo.Envoltorio">
  <objetos type="servidor.modelo.Asignacion">
    <asignador type="servidor.modelo.Gestor">
      <categoria>jefe</categoria>
      <puestoTrabajo>puesto1</puestoTrabajo>
      <direccion>ninguna</direccion>
      <fechaAlta>2008-04-06T14:55:20.150Z</fechaAlta>
      <fechaBaja>2008-04-06T14:55:20.150Z</fechaBaja>
      <fechaNaciemiento>2008-04-06T14:55:20.150Z</fechaNaciemiento>
      <id>00000000-G</id>
      <nombre>jose gestor0</nombre>
      <telefono>900000000</telefono>
    </asignador>
    <fin>2046-07-12T05:50:40.302Z</fin>
    <inicio>2008-04-06T14:55:20.151Z</inicio>
```

```
    <persona type="servidor.modelo.Paciente">
      <historial xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:nil="true" />
      <medicos xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:nil="true" />
      <direccion>ninguna</direccion>
      <fechaAlta>2008-04-06T14:55:20.108Z</fechaAlta>
      <fechaBaja>2008-04-06T14:55:20.108Z</fechaBaja>
      <fechaNaciemiento>2008-04-06T14:55:20.108Z</fechaNaciemiento>
      <id>00000000-P</id>
      <nombre>jose paciente0</nombre>
      <telefono>900000000</telefono>
    </persona>
  </objetos>
  <objetos type="servidor.modelo.Asignacion">
    <asignador type="servidor.modelo.Gestor">
      <categoria>jefe</categoria>
      <puestoTrabajo>puesto1</puestoTrabajo>
      <direccion>ninguna</direccion>
      <fechaAlta>2008-04-06T14:55:20.150Z</fechaAlta>
      <fechaBaja>2008-04-06T14:55:20.150Z</fechaBaja>
      <fechaNaciemiento>2008-04-06T14:55:20.150Z</fechaNaciemiento>
      <id>00000000-G</id>
      <nombre>jose gestor0</nombre>
      <telefono>900000000</telefono>
    </asignador>
    <fin>2046-07-12T05:50:40.302Z</fin>
    <inicio>2008-04-06T14:55:20.151Z</inicio>
    <persona type="servidor.modelo.Paciente">
      <historial xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:nil="true" />
      <medicos xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:nil="true" />
      <direccion>ninguna</direccion>
      <fechaAlta>2008-04-06T14:55:20.108Z</fechaAlta>
      <fechaBaja>2008-04-06T14:55:20.108Z</fechaBaja>
      <fechaNaciemiento>2008-04-06T14:55:20.108Z</fechaNaciemiento>
      <id>00000002-P</id>
      <nombre>jose paciente2</nombre>
      <telefono>900000000</telefono>
    </persona>
  </objetos>
</Envoltorio>


<Visita type="servidor.modelo.Visita">
  <anamnesi>enfermo</anamnesi>
  <diagnosis>malito</diagnosis>
  <fecha>2008-04-06T14:55:20.108Z</fecha>
  <idVisita>v1</idVisita>
  <medico type="servidor.modelo.Medico">
    <especialidad>TRAUMA</especialidad>
    <pacientes xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:nil="true" />
    <direccion>ninguna</direccion>
    <fechaAlta>2008-04-06T14:55:20.149Z</fechaAlta>
    <fechaBaja>2008-04-06T14:55:20.149Z</fechaBaja>
    <fechaNaciemiento>2008-04-06T14:55:20.149Z</fechaNaciemiento>
```

```
    <id>00000000-M</id>
    <nombre>jose medico0</nombre>
    <telefono>900000000</telefono>
  </medico>
  <tratamiento>aspirina</tratamiento>
</Visita>
```

Las clases (Javabean) a las que pertenecen los objetos anteriores, que Axi2 procesa y transforma en xml son:

```
public class Persona{

        private String id;
        private String nombre;
        private String telefono;
        private String direccion;
        private Date fechaNaciemiento;
        private Date fechaAlta;
        private Date fechaBaja;

        public Persona() {
                super();
        }
        public Persona(String id, String nombre, String telefono, String direccion,
                        Date fechaNaciemiento, Date fechaAlta, Date fechaBaja) {
                super();
                this.id = id;
                this.nombre = nombre;
                this.telefono = telefono;
                this.direccion = direccion;
                this.fechaNaciemiento = fechaNaciemiento;
                this.fechaAlta = fechaAlta;
                this.fechaBaja = fechaBaja;
        }

        ...métodos get/set



public class Paciente extends Persona{

        private String nSS;
        private Historial historial;
        private Asignacion [] medicos;

        public Paciente() {
                super();
        }
        public Paciente(String id, String nombre, String telefono,
                        String direccion, Date fechaNaciemiento, Date fechaAlta,
                        Date fechaBaja,String nss, Historial historial, Asignacion [] medicos) {
                super(id, nombre, telefono, direccion, fechaNaciemiento, fechaAlta, fechaBaja);
                nSS = nss;
                this.historial = historial;
                this.medicos = medicos;
        }

        ...métodos get/set
```

```
public class Medico extends Persona{

	private String nColegiado;
	private String especialidad;
	private Asignacion [] pacientes;

	public Medico() {
		super();
	}
	public Medico(String id, String nombre, String telefono, String direccion,
			Date fechaNaciemiento, Date fechaAlta, Date fechaBaja,
			String colegiado, String especialidad,
			Asignacion [] pacientes) {
		super(id, nombre, telefono, direccion, fechaNaciemiento, fechaAlta, fechaBaja);
		nColegiado = colegiado;
		this.especialidad = especialidad;
		this.pacientes = pacientes;
	}

	...métodos get/set


public class Gestor extends Persona{

	private String categoria;
	private String puestoTrabajo;


	public Gestor() {
		super();
	}
	public Gestor(String id, String nombre, String telefono, String direccion,
			Date fechaNaciemiento, Date fechaAlta, Date fechaBaja,
			String categoria, String puestoTrabajo) {
		super(id, nombre, telefono, direccion, fechaNaciemiento, fechaAlta, fechaBaja);
		this.categoria = categoria;
		this.puestoTrabajo = puestoTrabajo;
	}

	...métodos get/set



public class Historial  {

	private Visita [] visitas;
	private String grupoSanguineo;
	private String [] alergias;

	public Historial(Visita [] visitas,
			String grupoSanguineo, String [] alergias) {
		super();
		this.visitas = visitas;
		this.grupoSanguineo = grupoSanguineo;
		this.alergias = alergias;
	}
	public Historial() {
		super();
	}

	...métodos get/set
```

```
public class Visita {

	private String idVisita;
	private String tema;
	private String anamnesi;
	private Medico medico;
	private Date fecha;
	private String tratamiento;
	private String diagnosis;

	public Visita(String idVisita,String tema, String anamnesi, Medico medico,
			Date fecha, String tratamiento, String diagnosis) {
		super();
		this.idVisita=idVisita;
		this.tema=tema;
		this.anamnesi = anamnesi;
		this.medico = medico;
		this.fecha = fecha;
		this.tratamiento = tratamiento;
		this.diagnosis = diagnosis;
	}
	public Visita() {
		super();
	}

	...métodos get/set


public class Envoltorio {

	private Object[] objetos;

	...métodos get/set
```

Anexo IX. Configuración de Axis2.

services.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<service>
    <operation name="consultaDatosPaciente">
        <messageReceiver class="org.apache.axis2.receivers.RawXMLINOutMessageReceiver"/>
    </operation>
    <operation name="consultaVisitaPaciente">
        <messageReceiver class="org.apache.axis2.receivers.RawXMLINOutMessageReceiver"/>
    </operation>
    <operation name="consultaPacientesMedico">
        <messageReceiver class="org.apache.axis2.receivers.RawXMLINOutMessageReceiver"/>
    </operation>
    <operation name="anyadeVisitaPaciente">
        <messageReceiver class="org.apache.axis2.receivers.RawXMLINOutMessageReceiver"/>
    </operation>
    <operation name="asignaPacienteMedico">
        <messageReceiver class="org.apache.axis2.receivers.RawXMLINOutMessageReceiver"/>
    </operation>
    <operation name="desasignaPacienteMedico">
        <messageReceiver class="org.apache.axis2.receivers.RawXMLINOutMessageReceiver"/>
    </operation>
    <parameter name="ServiceClass" locked="false">servidor.SimpleService</parameter>
    <module ref="rampart" />
    <parameter name="InflowSecurity">
        <action>
            <items>Timestamp Signature Encrypt</items>
            <passwordCallbackClass>servidor.PWCBHandler</passwordCallbackClass>
            <signaturePropFile>service.properties</signaturePropFile>
        </action>
    </parameter>
    <parameter name="OutflowSecurity">
        <action>
            <items>Timestamp Signature Encrypt</items>
            <user>servidor</user>
            <encryptionKeyTransportAlgorithm>
                http://www.w3.org/2001/04/xmlenc#rsa-1_5
            </encryptionKeyTransportAlgorithm>
            <encryptionSymAlgorithm>
                http://www.w3.org/2001/04/xmlenc#aes256-cbc
            </encryptionSymAlgorithm>
            <passwordCallbackClass>servidor.PWCBHandler</passwordCallbackClass>
            <signaturePropFile>service.properties</signaturePropFile>
            <signatureKeyIdentifier>DirectReference</signatureKeyIdentifier>
            <encryptionKeyIdentifier>SKIKeyIdentifier</encryptionKeyIdentifier>
            <signatureParts>
                {Element}{http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
                utility-1.0.xsd}Timestamp;
                {Element}{http://schemas.xmlsoap.org/soap/envelope/}Body
            </signatureParts>
            <encryptionUser>useReqSigCert</encryptionUser>
        </action>
    </parameter>
    <schema schemaNamespace="http://PFC_seguridad.uoc.edu/xsd/"/>
</service>
```

operaciones ofrecidas por el WS

clase que implementa los servicios

acciones flujo entrada

clase que proporciona claves

acciones flujo salida

usuario

algoritmo transporte clave simétrica

algoritmo simétrico

partes firmadas: Timestamp, Body

espacio de nombres propio

De igual modo, el cliente utiliza un fichero xml de configuración, en el que se indican las acciones especulares con relación a las utilizadas por el servidor.

axis2.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<axisconfig name="AxisJava2.0">
   <module ref="rampart" />
   <parameter name="OutflowSecurity">
      <action>
         <items>Timestamp Signature Encrypt</items>
         <user>paciente</user>
         <encryptionKeyTransportAlgorithm>
            http://www.w3.org/2001/04/xmlenc#rsa-1_5
         </encryptionKeyTransportAlgorithm>
         <encryptionSymAlgorithm>
           http://www.w3.org/2001/04/xmlenc#aes256-cbc
         </encryptionSymAlgorithm>
         <passwordCallbackClass>paciente.PWCBHandler</passwordCallbackClass>
         <signaturePropFile>client.properties</signaturePropFile>
         <signatureKeyIdentifier>DirectReference</signatureKeyIdentifier>
         <signatureParts>
           {Element}{http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
           utility-1.0.xsd}Timestamp;
           {Element}{http://schemas.xmlsoap.org/soap/envelope/}Body
         </signatureParts>
         <encryptionKeyIdentifier>SKIKeyIdentifier</encryptionKeyIdentifier>
         <encryptionUser>servidor</encryptionUser>
      </action>
   </parameter>
   <parameter name="InflowSecurity">
      <action>
         <items>Timestamp Signature Encrypt</items>
         <passwordCallbackClass>paciente.PWCBHandler</passwordCallbackClass>
         <signaturePropFile>client.properties</signaturePropFile>
     </action>
   </parameter>


   <!-- RESTO CONFIGURACION AXIS2 -->
   ...


</axisconfig>
```