

Instituto Tecnológico y de Estudios Superiores de Monterrey

Campus Monterrey

DIVISIÓN DE TECNOLOGÍAS DE LA INFORMACIÓN
Y ELECTRÓNICA



Desarrollo de un Sistema de Teleconsulta y Telediagnosís para Telemedicina

PROYECTO FINAL DE CARRERA
Presentado como requisito parcial para obtener el grado de
Ingeniero Superior en Telecomunicaciones
Especialidad Telemática

Jorge Martí Coronil
Monterrey (México) Junio 2007

Instituto Tecnológico y de Estudios Superiores de Monterrey

Campus Monterrey

**DIVISIÓN DE TECNOLOGÍAS DE LA INFORMACIÓN
Y ELECTRÓNICA**

**Los miembros del comité de evaluación del proyecto del alumno
Jorge Martí Coronil recomendamos que el presente sea aceptado como
Proyecto Final de Carrera, requisito parcial para la obtención del grado académico de**

INGENIERO SUPERIO DE TELECOMUNICACIÓN

Especialidad en Telemática

Comité de Tesis

Dr. Gerardo A. Castañón Ávila
Asesor
ITESM

Maestro Artemio Aguilar
Sinodal
ITESM

Dr. Gabriel Campuzano
Sinodal
ITESM

Prof. Antonio Arnau Vives
Subdirector de Relaciones Externas
UPV

Junio 2007

Abstract

La salud es uno de los aspectos más importantes en la vida de una persona, directamente relacionada con la felicidad de la misma y de sus allegados. El desarrollo de una sociedad en aspectos tecnológicos, económicos, sociales, etc. es absurdo si no va acompañado de un desarrollo paralelo en salud y bienestar. Para este objetivo, gobiernos y empresas destinan inmensas cantidades de dinero en investigación médica y farmacológica, en equipamientos, personal, etc., pero no siempre con los mejores resultados, ni tampoco a veces con las mejores intenciones. ¿Es posible intentar una mejora sustancial de la atención sanitaria visto desde otro prisma? ¿Por qué no intentar aplicar tecnologías ya existentes aparentemente no relacionadas directamente con el campo de la medicina clásica para mejorar esta última?

Éste es el objetivo principal de la Telemedicina: hacer uso de las hoy en día casi ubicuas tecnologías de la información y comunicación a distancia para potenciar y en cierta manera globalizar y mejorar la atención sanitaria. Con respecto a ese objetivo, este trabajo intenta aportar en los campos de *Teleconsulta* y *Telediagnosís*, relacionados ambos con la Telemedicina.

El proyecto desarrollado consiste en la realización de dos programas informáticos, uno pensado para el uso por parte de un doctor o especialista, y otro para ser utilizado por parte de un paciente (con la posible ayuda de un asistente o enfermero). Mediante el uso de ambos, el paciente puede recibir una consulta por parte del doctor sin encontrarse ambos en el mismo lugar, mediante la transmisión de una forma sencilla e intuitiva de voz, vídeo, imágenes, datos médicos, etc., para un mejor diagnóstico por parte del doctor.

Para darle un enfoque más cercano a la realidad, hemos querido ponerle un nombre al sistema, para que así el proyecto pudiese ser conocido por sí mismo. A continuación, les presentamos el proyecto **Telemédomai**.

Dedicatoria

Es ineludible mencionar el objetivo social de este proyecto, cuya contribución al acceso universal a la salud queremos no resulte cual clamor en el desierto, sino esperamos sirva para mejorar al menos un ápice la situación de aquellos a quienes el acceso a cualquier cosa que no sea la escasez es difícil, aquellos de quienes nadie se acuerda,

... los olvidados.

A ellos y a México, el país de cuya alma son los auténticos portadores, va dedicado este trabajo.[0]

Índice

Abstract	V
Dedicatoria	VII
Índice de figuras y tablas	XI
Capítulo 1. Introducción	1
1.1 Definición del problema	1
1.1.1 La Sanidad en México	1
1.1.2 Algunos datos	2
1.2 Telemedicina	3
1.2.1 Ventajas e inconvenientes de la telemedicina	3
1.3 Intento de aproximación al problema	6
1.3.1 Objetivos principales	7
1.4 Alcance de la solución propuesta	7
Capítulo 2. Sistema de Teleconsulta y Telediagnosis	9
2.1 Videoconferencia	9
2.2 Auscultaciones	10
2.2.1 Auscultación del corazón	10
2.2.2 Auscultación de los pulmones	11
2.2.3 Auscultación del abdomen	12
2.2.4 Otros tipos de auscultaciones	14
2.3 Diagnóstico mediante imágenes	14
2.4 Presión arterial	16
2.5 Otras medidas	17
Capítulo 3. Tecnología empleada	19
3.1 Java y Netbeans	19
3.2 Internet y sockets	22
3.2.1 Sockets TCP	24
3.3 Seguridad con SSL y certificados	26
3.4 Multimedia con JMF y RTP	29
Capítulo 4. Funcionamiento del sistema	33
4.1 Cliente y Servidor	33
4.2 Instalación	34
4.2.1 Instalación de cliente y servidor	35

4.2.2	Instalación de la máquina virtual Java y JMF	36
4.2.3	Creación e instalación de los certificados firmados	36
4.3	Realizar modificaciones del código de los programas	38
4.4	Funcionamiento parte cliente	39
4.4.1	Envío de datos del paciente	41
4.4.2	Inicio de conferencia vídeo y/o audio	41
4.4.3	Estetoscopia asistida	42
4.4.4	Toma de la presión arterial	44
4.4.5	Captura y envío de imágenes	45
4.4.6	Envío de cualquier tipo de medición	46
4.4.7	Recepción de diagnóstico y recomendaciones	47
4.4.8	Iniciar nueva consulta	48
4.5	Funcionamiento parte servidor	49
4.5.1	Log in del especialista	49
4.5.2	Inicio de conferencia vídeo y/o audio	51
4.5.3	Recepción de sonidos corporales	52
4.5.4	Control de la presión arterial	54
4.5.5	Análisis de imágenes	55
4.5.6	Otros niveles a analizar	56
4.5.7	Redacción y envío del diagnóstico	56
4.5.8	Otras funciones: nuevo paciente, ayuda,...	57
Capítulo 5.	'Destripando' el programa	59
5.1	Protocolo interno de comunicaciones	59
5.2	Jerarquía de clases	61
5.2.1	Programa cliente	61
5.2.2	Programa servidor	64
5.3	Almacenamiento de datos	66
Capítulo 6.	Conclusiones	69
6.1	Resultados	69
6.2	Posibles extensiones	70
Apéndice A.	Clases importantes	73
A.1	ProtocolClient.java	73
A.2	ProtocolServer.java	81
A.3	Ejemplo tipo de datos: Imagen.java	89
A.4	Otras clases auxiliares	91
Apéndice B.	Historial de versiones	95
Bibliografía	99
Epílogo	101

Índice de figuras y tablas

2.1. Estetoscopio analógico común	10
2.2. Focos de auscultación cardíaca	11
2.3. Puntos de auscultación pulmonar anterior y posterior	12
2.4. Patologías frecuentes del abdomen según localización	13
2.5. Ejemplo de una radiografía del tórax	14
2.6. Imagen de un ECG digitalizado	15
2.7. Colocación del esfigmomanómetro y gráfico de las medidas implicadas.....	16
2.8. Medidor de glucosa (glucómetro)	17
3.1. Netbeans funcionando en un entorno GNU/Linux Ubuntu 7.04	21
3.2. Pila de protocolos TCP/IP	23
3.3. Gráfico de protocolos	24
3.4. Esquema de creación de sockets	25
3.5. Pila de protocolos SSL	27
3.6. Esquema de intercambio de mensajes en fase “Handshake”	28
3.7. Etapas de la arquitectura JMF	30
3.8. Arquitectura de RTP	32
4.1. Ventana de información con logotipo del sistema	34
4.2. Programa de instalación del servidor	35
4.3. Ventana de abrir proyectos existentes en Netbeans	38
4.4. Menú contextual de un proyecto en Netbeans	39
4.5. Aspecto del programa cliente al iniciar	40
4.6. Iniciando videoconferencia	42
4.7. Captura de realización dos tipos de auscultaciones guiadas	43
4.8. Pestaña de toma de presión arterial	44
4.9. Imagen capturada, lista para ser transmitida	45
4.10. Pestaña de otras medidas	46
4.11. Recepción de diagnóstico y recomendaciones	47
4.12. Aspecto del programa servidor al iniciar	48

4.13. Ventana emergente de log in	50
4.14. Tras log in correcto, el programa muestra la información del especialista	51
4.15. Recepción de datos y videoconferencia desde paciente	52
4.16. Pestaña de estetoscopias, mostrando tres auscultaciones recibidas	53
4.17. Recepción de medidas de presión arterial	54
4.18. Pestaña de imágenes recibidas, mostrando un ejemplo de radiografía	55
4.19. Recepción de nivel de colesterol, con activación de la pestaña “Otros”	56
4.20. Ejemplo de informe generado con los datos de una consulta	57
4.21. Detalle de los menús principal (a) y de ayuda (b)	57
5.1. Esquema de mensajes del protocolo	61
5.2. Jerarquía de clases del programa cliente	62
5.3. Jerarquía de clases del programa servidor	65
5.4. Ejemplo de carpeta con los datos de la sesión de consulta	67

Capítulo 1

Introducción

Para este primer capítulo queremos antes que nada introducir al lector en la problemática de la atención sanitaria en países con grandes extensiones y multitud de pequeños núcleos habitados alejados de las grandes urbes.

Para potenciar esta introducción, analizaremos el ejemplo de un país con estas características como lo es México, intentando obtener una vista global del estado de la sanidad en este país, ofreciendo algunos datos para retratar de forma más fidedigna esta característica. Posteriormente y para entrar en situación intentaremos dar una definición global del concepto de la Telemedicina, y más en particular de la Teleconsulta y la Telediagnos, para lo que estudiaremos cuáles son las ventajas de utilizar esta serie de tecnologías.

Finalmente, relataremos de qué manera el proyecto realizado intenta ayudar a la implantación y uso de las ventajas explicadas acerca de la Telemedicina, qué funcionalidades ofrece y qué objetivos cumple en pos de su propósito principal de contribuir a una atención sanitaria global, accesible y de calidad.

1.1. Definición del problema

1.1.1. La Sanidad en México

La accesibilidad a una adecuada y oportuna atención de la salud es un derecho de todas las personas, sin embargo la falta de recursos y su mala distribución, obliga a priorizar las acciones

sobre todo en los grupos más vulnerables como lo son las comunidades del interior del país más alejadas de las grandes urbes por distancias o por situaciones geográficas como orografías dificultosas, desiertos, etc.

La Telemedicina, viene a traer una solución a esta problemática disminuyendo las distancias y unificando de esta manera la calidad de educación y salud que se ofrece a todos los ciudadanos del país. Pese a que la Telemedicina está aún en fase de proliferación, con la tecnología existente es posible brindar la mejor atención de salud a distancia.

México, como la mayoría de los países de Latinoamérica, comparten la mala distribución de recursos especializados, centralizados en las grandes urbes. Se suma a esto la gran extensión geográfica y diversidad de parajes, que dificulta las comunicaciones y traslados de profesionales y pacientes. La comunidad europea lo ha implementado ya hace años al ver que los costos en salud eran crecientes y con la posibilidad de reducirlos mediante la enseñanza y actualización médica a distancia, monitorización ambulatoria y domiciliaria de pacientes o promoción de atención primaria de la salud.[1].

La Telemedicina se plantea como una solución a este tipo de problemas y México en particular se encuentra entre los mas adecuados para este tipo de programas.

1.1.2. Algunos datos

En el censo efectuado por el INEGI (Instituto Nacional de Estadística, Geografía e Informática) [2] de 2005 había en México 187.939 localidades (o asentamientos), es decir, sitios de censo designados, las cuales pueden ser ya sea un pequeño pueblo, una ciudad grande o simplemente una sola vivienda en un área agrícola (rural) lejos o cerca de una área urbana. En este país, una ciudad se define como la localidad con más de 2.500 habitantes.

En el 2005 había 2.640 ciudades con una población entre los 2.500 y los 15.000 habitantes, 427 con una población entre 15.000 y 100.000 habitantes, 112 con una población entre 100.000 y un millón, y 11 ciudades con más de un millón de habitantes. Todas las ciudades se consideran "áreas urbanas" y albergan al 76,5% de la población nacional. Las localidades con menos de 2.500 habitantes se consideran "áreas rurales" (de hecho, 80.000 de estas localidades sólo tienen una o dos viviendas), y albergan al 23,5% de la población.

Resumiendo, podemos observar que sólo ese porcentaje del 23,5% de la población, considerada población rural, ocupa en cambio el 98,3% de los asentamientos, por lo que la asistencia sanitaria presencial en estas poblaciones, a todo este grupo de población, se hace increíblemente dificultosa, si no imposible. Es necesario por tanto utilizar otros medios que intenten garantizar la igualdad en cuanto a atención médica de forma igualitaria para todos los ciudadanos del país, o al menos intente maximizarla. Esta tesis intenta evaluar el uso de un sistema de Telemedicina para tal propósito. Intentemos entonces primero dar una definición precisa de este "novedoso" concepto.

1.2. Telemedicina

Si bien el uso de este concepto aparenta ser bastante reciente, la telemedicina en sí existe desde el primer momento que alguien utilizó algún sistema de comunicación para obtener una información útil para actuar en un determinado problema médico. Algo tan sencillo como una llamada telefónica de un especialista a otro colega solicitándole consejo médico ya entraría a formar parte de un acto dentro de la telemedicina. Pero mejor consultamos a las autoridades pertinentes para una definición más estricta de este concepto.

Según la Organización Mundial de la Salud (WHO en inglés), la telemedicina es definida como:

“ El suministro de servicios de atención sanitaria, en los que la distancia constituye un factor crítico, por profesionales que apelan a las tecnologías de la información y de la comunicación con objeto de intercambiar datos para hacer diagnósticos, recomendar tratamientos y prevenir enfermedades y accidentes, así como para la formación permanente de los profesionales de atención de salud y en actividades de investigación y evaluación, con el fin de mejorar la salud de las personas y de las comunidades en que viven. ”

Otra de las definiciones es la que propone la American Telemedicine Association (ATA), que considera la telemedicina como:

“ El intercambio de información médica de un lugar a otro, usando las vías de comunicación electrónicas, para la salud y educación del paciente o el proveedor de los servicios sanitarios y con el objetivo de mejorar la asistencia del paciente. ”

Podemos observar que la constante entre ambas definiciones es que la telemedicina siempre implica intercambio de información. De ahí la revolución de la telemedicina en la era de las comunicaciones, sobre todo con la aparición de Internet y sus herramientas. Hoy en día se hace posible e incluso sencillo la transmisión de grandes cantidades de datos de distintos formatos (texto, vídeo, sonido, imágenes,...) a casi cualquier parte del mundo. Esta es una ventaja que se ha de saber aprovechar, pues no siempre se saca el rendimiento adecuado a las tecnologías existentes. Para completar esta definición vamos a relatar las posibles ventajas e inconvenientes que a priori observamos acerca del uso de la telemedicina.

1.2.1. Ventajas e inconvenientes de la telemedicina

La telemedicina disminuye las diferencias entre los diferentes niveles asistenciales y aumenta la integración de la información de los pacientes y la cooperación entre los profesionales. También se debe considerar como un elemento de gestión que mejora la eficacia y eficiencia de la atención y la optimización de recursos. Podemos recopilar una lista de ventajas que podemos encontrar en el uso de la telemedicina, para cada uno de las tres entidades implicadas en el acto de la atención sanitaria.[3].

Para el usuario

- Facilita la accesibilidad de los recursos sanitarios en múltiples aspectos, desde una cita centralizada a los profesionales de atención primaria hasta facilitar las citas a atención especializada directamente desde su centro de salud.
- Consulta a distancia (Teleconsulta). Las posibilidades son importantes, ya que permiten llevar a cabo desde el domicilio del usuario múltiples intervenciones hasta ahora no posibles de realizar. El control domiciliario de pacientes crónicos, la transmisión de datos clínicos desde el domicilio a la historia clínica o a otro profesional para su valoración son realidades tecnológicas en el momento actual.
- Asistencia sanitaria global. Entendida en múltiples vertientes: por un lado, el desplazamiento físico del usuario (traslados del domicilio habitual o cambios de domicilio) no conlleva pérdida de información sanitaria. Por otro lado, cuando todos los profesionales sanitarios que intervienen en un mismo proceso de enfermar comparten y disponen de la totalidad de la información sobre un mismo usuario, se facilitan los procesos diagnósticos y se evita la duplicidad de intervenciones con el gasto humano y económico que esto produce.
- Intervención rápida en caso de urgencia o empeoramiento.
- Reducción de costes indirectos (desplazamientos, etc.), que de normal son asumidos por el paciente.

Para los profesionales

- Mejor accesibilidad a los datos del usuario. Con la posibilidad de una historia clínica informatizada e, independientemente de su ubicación y/o dispersión, podemos acceder a todos los datos sanitarios de un usuario concreto de manera global sin importar dónde se ha realizado cada una de las anotaciones o intervenciones concretas de la historia clínica.
- Posibilidad de uso de herramientas informáticas en la historia clínica. Con una historia clínica en papel la dispersión de los datos en un sistema poco flexible hace que exista el riesgo de una pérdida importante de información. La concreción y homogeneidad de los datos que exige una historia clínica informática tiene como ventaja la posibilidad de aplicar todas las herramientas informáticas a la misma.
- Posibilidad de investigación. Las posibilidades en la investigación son múltiples, ya que los datos en formato digital permiten un mayor acceso y un mejor manejo con menos pérdida de información.

- Docencia. La teleformación o teledocencia se puede considerar como curso temático a distancia o como sesión clínica impartida entre uno o varios ponentes interconectados entre sí donde las distancias físicas que separan a cada uno de los interlocutores son inexistentes a través de la red.
- Mejora las comunicaciones entre profesionales. Las comunicaciones entre diferentes profesionales sanitarios que intervienen en un proceso de enfermedad son escasas. Si intervienen diferentes niveles asistenciales, las dificultades de comunicación son mayores y en muchas ocasiones es el propio usuario quien realiza la labor de intermediario de la información con la pérdida que esto origina. A través de correo electrónico, con informes digitales o con teleconferencias, las comunicaciones entre profesionales son ágiles e inmediatas.

Para el sistema

- Comunicación interprofesional. Las ventajas para el sistema de la comunicación entre los diversos profesionales son múltiples. Por un lado agiliza y disminuye los tiempos de espera entre diferentes intervenciones, evita la duplicidad de intervenciones y, por lo tanto, aumenta la calidad en los servicios.
- Difusión de información. A través de sistemas de redes se puede difundir información multidireccional entre cada uno de los elementos del sistema. Esta información puede ser parcializada (de manera que no todos los integrantes de la red posean los mismos niveles de acceso) y segura.
- Facilita trámites burocráticos. Desde una accesibilidad al sistema por parte de los usuarios donde los factores distancia, tiempo o saturación de líneas no sean un inconveniente, hasta que todos los trámites internos puedan ser manejados de manera eficiente por el propio sistema sin que el usuario tenga que portar personalmente los volantes de citas.
- Oferta mayor calidad en la atención. El usuario se convierte en un sujeto activo de su proceso y el sistema resuelve todos los trámites e inconvenientes que se originan de forma paralela al proceso asistencial. Todo ello revierte en una mejora de la calidad de los servicios prestados.
- Mayor alcance de la cobertura de atención sanitaria del sistema y simplificación de la misma, consiguiendo a la vez un importante ahorro en infraestructuras que repercute en mejor servicio para toda la sociedad.

Pero, aparte de todas estas ventajas, se han de tener en cuenta ciertos aspectos también implicados en la práctica de la telemedicina relacionados con los riesgos potenciales propios del sistema, debidos a problemas físicos, financieros, éticos... Algunos de estos inconvenientes podrían ser los siguientes:[4]

- Falta de privacidad y confidencialidad. En el campo de las consultas en línea (teleconsultas), el principal problema con el uso de Internet tiene que ver con la privacidad y la confidencialidad. Es difícil separar espacios privados dentro de un escenario público por naturaleza, por lo que se han de tomar las medidas necesarias para garantizar el grado adecuado de privacidad (como por ejemplo utilizando conexiones seguras).
- Acceso desigual a la información. Al mismo tiempo que la sociedad de la información consigue reducir la brecha de conocimiento entre médicos y pacientes, genera un ensanchamiento de esta brecha entre quienes tienen acceso a las nuevas tecnologías y entre quienes han sido excluidos de ellas. Existe un círculo vicioso alrededor de la escasa educación, los bajos ingresos, los servicios deficientes de salud, y la inaccesibilidad a las tecnologías de la información, que no se puede romper a menos que no se deje actuar solamente a las fuerzas de los mercados, corporaciones, etc. sino por medio de políticas eficientes de **salud pública** que brinden el acceso universal a la información en salud. En dirección a este objetivo este proyecto intenta aportar un poco de ayuda para colaborar en paliar esta situación, que podemos observar claramente en países como México, donde **desigualdad** sería una adecuada palabra para definir al global de su sociedad.
- Calidad del servicio. Toda persona debe tener la posibilidad de consultar un médico, pero es necesario cerciorarse de la posesión del especialista de un título profesional obtenido con base a sus estudios y la correspondiente licencia para el ejercicio profesional. Debe entonces haber alguna regulación que asegure al “ciberpaciente” contra las consecuencias de la ignorancia e incapacidad de un farsante que podría atenderlo en línea. Una posible solución es el uso generalizado de certificados digitales que autenticuen a los especialistas, expedidos por la autoridad pertinente.

1.3. Intento de aproximación al problema

Como hemos podido observar en los puntos anteriores, la telemedicina es un campo no sólo extenso sino en continua expansión. Es por ello que a la hora de realizar un proyecto relacionado con esta materia se deba acotar los objetivos pretendidos, pues intentar proponerse la realización de un proyecto sobre telemedicina de forma global sería bastante pretencioso.

Por este motivo decidimos concretar a la hora de definir los que serían los objetivos para este proyecto, que serían los siguientes: realizar un *sistema de teleconsulta y telediagnóstico*, basado en el uso de dos programas, uno para el uso por parte del paciente, y otro para que el especialista pueda analizar y catalogar con facilidad la información recibida desde el paciente y proporcionarle un diagnóstico adecuado.

Con un despliegue extenso de este sistema, el paciente podría consultar desde su centro de atención primaria a diferentes especialistas en cualquier campo de la medicina directamente en el lugar que éstos se encuentren, recibiendo una atención inmediata, e intentando ésta sea de la mejor calidad, para lo que el programa hará de portador de toda la información que el especialista necesite en su labor de consulta y diagnóstico.

1.3.1. Objetivos principales

- Realizar un programa sencillo de manejar para permitir su uso y rápido aprendizaje por cualquier asistente o directamente por los pacientes, pues muchas de sus funciones serían auto explicativas y/o guiadas por el especialista al otro lado de la línea.
- Posibilidad de videoconferencia para un trato más cercano entre paciente y especialista.
- Fácil de distribuir e instalar.
- Flexible para su aplicación a diversas patologías.
- Poco exigente técnicamente: aprovechamiento máximo de los recursos disponibles en cada punto de utilización del sistema, desde centros con recursos mínimos hasta centros de salud bien dotados.
- Programación clara, modular y flexible para su fácil ampliación (extensible).

1.4. Alcance de la solución propuesta

Una vez concluida la fase de desarrollo tanto del programa cliente como del servidor, podemos finalmente observar cuál ha sido el alcance del sistema implementado, partiendo de los objetivos marcado en principio. Queremos puntualizar que, por encima de los anteriores objetivos, el propósito principal de este trabajo era ser un punto de partida en el desarrollo de un sistema mucho más completo, para lo que creemos se necesitaría la colaboración de un equipo multidisciplinario, incluyendo médicos, ingenieros de comunicaciones, electrónicos e informáticos.

Por este motivo, confiamos que la finalidad pionera de este programa se vea cumplida, y para ello ofrecemos la relación de las funcionalidades logradas para nuestro sistema:

- Videoconferencia bidireccional entre paciente y especialista, y posibilidad de conferencia sólo de voz si los recursos o la baja velocidad de la red no permiten vídeo.
- Captura guiada de sonidos corporales (auscultación) y comentarios, y transmisión hacia el programa servidor, para ser analizados y almacenados por el especialista.
- Captura de imágenes utilizando una cámara o lectura de imágenes capturadas mediante otros dispositivos para el diagnóstico ocular por parte del especialista (radiografías, imágenes superficiales, endoscopias, otoscopias...).
- Toma guiada de la presión arterial para el expediente médico por parte del doctor.
- Anotación y transmisión de otros tipos de medidas médicas, como niveles de colesterol, glucosa, peso u otro de cualquier tipo.

- Almacenamiento de TODOS los datos recibidos para una posible consulta “offline” de los datos, o para la retransmisión a otro especialista para una segunda opinión.
- Envío del informe final de diagnóstico, conteniendo además los datos del paciente y la consulta, recomendaciones y datos del especialista consultado.

Capítulo 2

Sistema de Teleconsulta y Telediagnos

En este segundo capítulo entraremos en más profundidad en la explicación de las funcionalidades del sistema, especialmente en los diferentes servicios que ofrece para la intercomunicación entre paciente y especialista. Empezaremos con la función de videoconferencia, una de las principales a la hora de realizar una consulta entre doctor y paciente que transmita cercanía y confianza. Continuaremos hablando del servicio de transferencia de sonidos, y en particular hablaremos sobre los diferentes tipos de auscultaciones que pueden realizarse, y de las dolencias que esperamos detectar mediante estas pruebas auditivas.

En el siguiente apartado nos adentraremos en la diagnosis mediante el uso de imágenes del alta resolución, básica en el análisis de radiografías, fotografías de heridas, infecciones externas o por ejemplo en otorrinolaringología (oído, nariz y garganta). Continuaremos explicando la utilidad de la función de toma de la presión arterial, y finalmente daremos una relación de otras posibles medidas y niveles empleados en medicina que también podemos transmitir mediante nuestro sistema.

2.1. Videoconferencia

La consulta a través de Internet, por videoconferencia, facilita y diversifica el acceso paciente-especialista. Evita desplazamientos, sin perder el contacto "cara a cara". No son las personas las que corren de un lado para otro, sino su imagen y su voz. Proporciona un contacto permanente con su médico, mejorando la relación, y sobre todo reduce su ansiedad y preocupación, pues el paciente siente que realmente hay una persona "al otro lado" y no una máquina.

Mediante la videoconferencia obtenemos una realimentación inmediata por lo que el resto de funciones del programa pueden ser más fácilmente guiadas por el especialista, quien es el primer beneficiado de su uso pues aumentan cualitativamente los análisis de que dispone para realizar un diagnóstico exacto.

Si bien no es tan completo como en el caso de vídeo, las conferencias de audio también son muy útiles para los mismos objetivos citados con anterioridad, pero en este caso quizá la relación paciente-especialista es un poco más fría, equivalente a una conversación telefónica, y evidentemente se pierde el aspecto visual del diagnóstico, por lo que debe recurrirse al paso de imágenes desde otro tipo de cámaras.

2.2. Auscultaciones

La auscultación es un procedimiento que consiste en escuchar atentamente sonidos provenientes del cuerpo, utilizando un estetoscopio (ver Figura 2.1), para determinar el funcionamiento de un órgano o para detectar una enfermedad. Para el correcto funcionamiento de nuestro sistema sería necesario utilizar o bien un estetoscopio digital (que podamos conectar directamente al ordenador), o bien realizar una adaptación de un estetoscopio analógico a un micrófono, y posteriormente éste a una de las entradas de audio del sistema [5]. Existen varios tipos de auscultaciones, según la parte del cuerpo que pretendamos escuchar. [6].



Figura 2.1: Estetoscopio analógico común

2.2.1. Auscultación del corazón

Para escuchar el corazón, el doctor coloca el estetoscopio en el pecho en cinco puntos determinados. Con el paciente sentado, recostado en una postura semi reclinada o yaciendo sobre su lado izquierdo, el doctor escucha intentando detectar alguna anomalía en la pulsación y el ritmo cardíacos, o algún tipo de sonido anormal que podría indicar algún problema cardíaco.

Existen cuatro ruidos cardíacos básicos denominados S1 a S4 que se corresponden al cierre de las cuatro válvulas cardíacas. Dado que todos los ruidos cardíacos son de frecuencia más bien baja y se mantienen en niveles difícilmente detectables por el oído humano, la auscultación se debe llevar a cabo en un ambiente lo más silencioso posible. Como los sonidos se transmiten en la dirección del flujo sanguíneo, los ruidos cardíacos se escuchan mejor sobre zonas adonde va la sangre una vez que ha traspasado una válvula.

Estos son los cinco focos de auscultación tradicionales (ver Figura 2.2): [7].

- Aórtico (zona de la válvula aórtica): segundo espacio intercostal derecho, en el borde esternal derecho.
- Pulmonar (zona de la válvula pulmonar): segundo espacio intercostal izquierdo, en el borde esternal izquierdo.
- Pulmonar secundario: tercer espacio intercostal izquierdo, en el borde esternal izquierdo.
- Tricúspide: cuarto espacio intercostal izquierdo, en la parte inferior del borde esternal izquierdo.
- Mitral o apical: en el ápex cardíaco, en el quinto espacio intercostal izquierdo, línea medioclavicular.

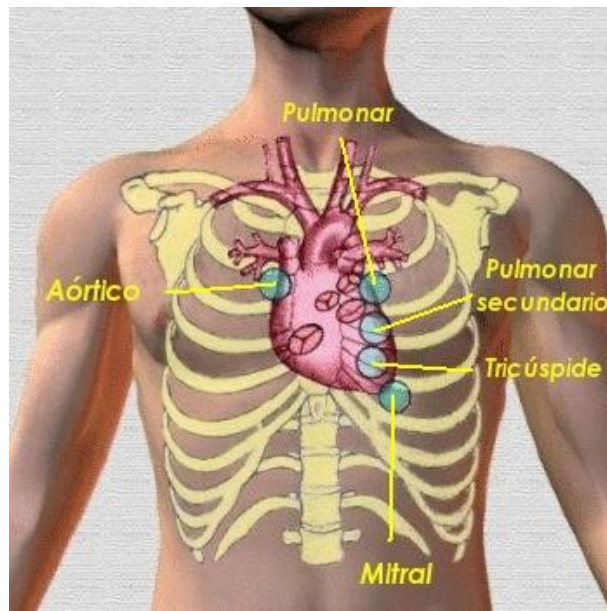


Figura 2.2: Focos de auscultación cardíaca

2.2.2. Auscultación de los pulmones

Para escuchar los sonidos provenientes de los pulmones, el doctor localiza el estetoscopio en numerosas áreas del pecho y la espalda. El paciente respira con normalidad, para posteriormente realizar largas inspiraciones, para así permitir al especialista comparar los sonidos de los lados izquierdo y derecho. Lo ideal es auscultar sucesivamente lugares homólogos de ambos

lados, lo que hace posible una comparación inmediata, útil para detectar lesiones unilaterales. La auscultación de la zona cubierta por la escápula y sus músculos es más difícil. La zona auscultable aumenta solicitando al paciente que cruce los brazos por delante del cuerpo, juntando los codos, lo que desplaza las escápulas. Sonidos anormales de respiración podrían indicar neumonía, bronquitis o neumotórax (presencia de aire en la cavidad pleural producto de una lesión en el parénquima pulmonar). [8].

Sonidos crujientes o burbujeantes, conocidos como crepitaciones, son causados por fluidos en los pulmones; sonidos como jadeantes pueden ser causados por contracciones en las vías respiratorias, normalmente como resultado de una inflamación de la pleura.

El doctor también puede realizar una prueba de resonancia vocal, pidiéndole en este caso al paciente que susurre algunas palabras. El sonido es más alto si existe pus en los pulmones por inflamación debido por ejemplo a una neumonía.

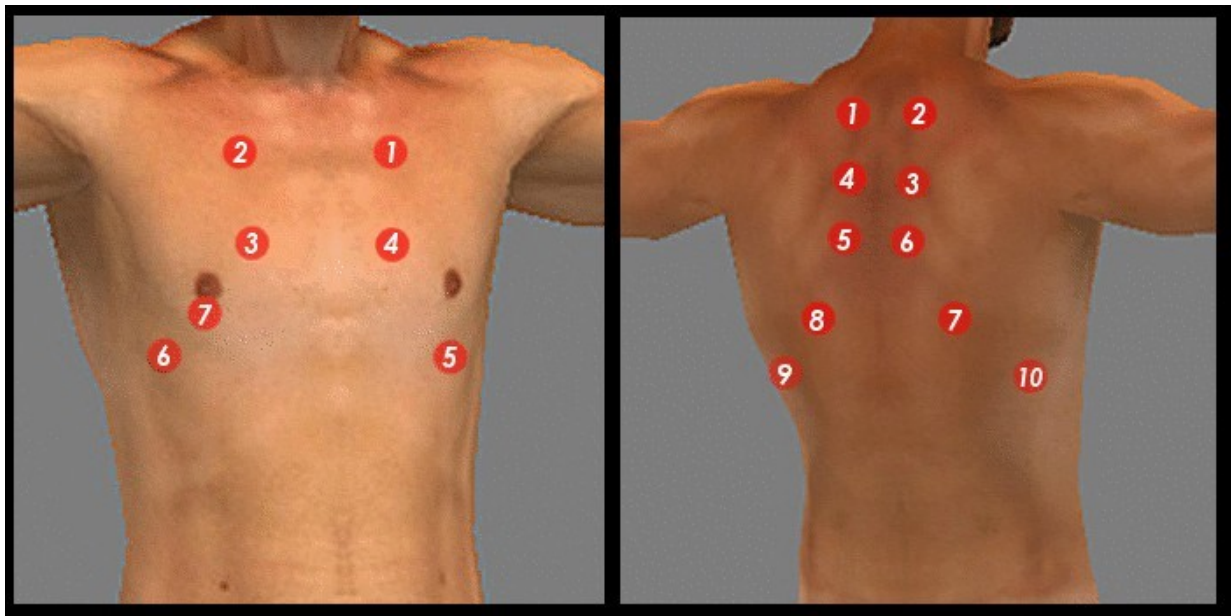


Figura 2.3: Puntos de auscultación pulmonar anterior y posterior

2.2.3. Auscultación del abdomen

El dolor abdominal es una síntoma inespecífico de multitud de procesos que si bien suele ser originado por causas intraabdominales, también puede ser provocado por procesos extraabdominales o por enfermedades sistémicas. El dolor abdominal puede tener diferentes desencadenantes y vías de propagación, así distinguimos:[9]

- **Dolor visceral:** Originado en las vísceras y el peritoneo visceral ; los estímulos dolorosos se transmiten por el sistema simpático hasta el ganglio raquídeo y de aquí al asta posterior medular por donde llegarán hasta el tálamo. Es un dolor de carácter sordo y de localización poco precisa , se puede acompañar de sintomatología vagal.

- **Dolor somático o parietal:** Originado en las estructuras de la pared abdominal y el peritoneo parietal; los estímulos se transmiten por los nervios periféricos correspondientes a los dermatomas, hasta el asta posterior medular y desde aquí a las fibras contralaterales del haz espinotalámico lateral. A nivel medular puede establecerse un reflejo autónomo a través de las vías eferentes simpáticas y también pueden transmitirse impulsos al asta anterior dando lugar a un componente motor (reflejo espinal, contractura muscular). Es un dolor agudo, intenso y bien localizado.
- **Dolor referido:** Se percibe en regiones anatómicas diferentes a la zona de estimulación y se produce por que esta zona de estimulación comparte segmento neuronal sensorial con el área dolorosa.

Las causas que desencadenan el dolor abdominal son tan diversas que hacen difícil el diagnóstico, es por ello que normalmente es necesario definir exactamente sus características, de esta manera podremos establecer un diagnóstico sindrómico que nos orientará para conducir su estudio y actitud terapéutica. Ante un dolor abdominal debemos considerar principalmente la edad del paciente (numerosas enfermedades se presentan principalmente en un determinada edad), intensidad del dolor (aunque este factor siempre es bastante relativo) y principalmente localización (ver Tabla 2.4) y cronología del dolor (dónde es originado y si se ha trasladado con el tiempo).

Patologías más frecuentes según la localización		
Cuadrante Superior Derecho	Epigastrio	Cuadrante Superior Izquierdo
Colecistitis aguda Úlcera duodenal perforada Pancreatitis aguda Hepatitis Hepatomegalia congestiva aguda Pielonefritis aguda Angina de pecho Apéndice retrocecal Neumonía con reacción pleural Cólico nefrítico	Úlcus péptico Esofagitis Perforación gástrica Infarto de miocardio Pancreatitis aguda Neumonía con reacción pleural	Rotura de Bazo Úlcera gástrica perforada Pancreatitis aguda Perforación de colon Neumonía con reacción pleural Pielonefritis aguda Infarto agudo de miocardio Cólico nefrítico
Periumbilical	Cuadrante Inferior Derecho	Cuadrante Inferior Izquierdo
Obstrucción intestinal Salpingitis aguda Pancreatitis aguda Trombosis mesentérica Hernia estrangulada Aneurisma aórtico complicado Diverticulitis aguda Uremia Cetoacidosis diabética Angor intestinal	Apendicitis Salpingitis aguda Rotura de folículo Embarazo ectópico roto Adenitis mesentérica Hernia inguinal estrangulada Ileitis regional Ciego perforado Cálculo uretral Epididimitis Pielonefritis Hidronefrosis Retención urinaria	Diverticulitis sigmoidea Salpingitis aguda Rotura de folículo Embarazo ectópico roto Hernia inguinal estrangulada Cálculo uretral Epididimitis Pielonefritis Hidronefrosis Colitis isquémica Retención urinaria

Tabla 2.4: Patologías frecuentes del abdomen según localización

2.2.4. Otros tipos de auscultaciones

La técnica de auscultación se puede utilizar para escuchar otros tipos de sonidos corporales, en pos de diagnosticar algún tipo de patología. Un ejemplo de esto sería la auscultación de vasos sanguíneos cerca de la superficie de la piel (normalmente en la arteria carótida en el cuello, la aorta abdominal o la arteria renal). El especialista puede escuchar atentamente en busca de sonidos causados por circulación sanguínea turbulenta o anormalmente rápida. Este factor puede ser producido por estrechamiento en algún vaso sanguíneo, o también por problemas en las válvulas del corazón.

2.3. Diagnóstico mediante imágenes

El diagnóstico visual por parte del especialista es vital en multitud de ramas de la medicina. Son miles los instrumentos médicos cuya única salida de respuesta es o bien un monitor o bien la impresión de un gráfico o imagen. Por citar algunos ejemplos importantes, el diagnóstico basado en imágenes es totalmente indispensable en los siguientes casos:

- **Radiología:** La tele radiología puede ser definida como la transmisión de imágenes radiológicas de una ubicación geográfica a otra, a través de redes de comunicaciones, para los propósitos de interpretación y consulta. [10]. Para ello es necesario convertir la radiografía convencional a formato digital para la transmisión sobre la red. Este propósito puede ser obtenido por varios mecanismos, desde los más tecnológicamente sofisticados como digitalizadores de imagen (digitalizadores láser y digitalizadores CCD, Charged Couple Devices) o hasta los más sencillos como un simple escaneo o captura de una radiografía tradicional (Figura 2.5). Del mismo modo es posible la transmisión de otras imágenes radiológicas como las TAC (tomografía axial computerizada) o las resonancias magnéticas.



Figura 2.5: Ejemplo de una radiografía del tórax

- **Ecografía:** es un procedimiento de imagenología que emplea los ecos de una emisión de ultrasonidos dirigida sobre un cuerpo u objeto como fuente de datos para formar una imagen de los órganos o masas internas con fines de diagnóstico. Un pequeño instrumento transductor emite ondas de ultrasonidos. Estas ondas sonoras de alta frecuencia se transmiten hacia el área del cuerpo bajo estudio, y se recibe su eco. El transductor recoge el eco de las ondas sonoras y una computadora convierte este eco en una imagen, que posteriormente se puede transmitir o almacenar. Esta técnica es utilizada tanto para visualizar embarazos como para detectar algunos tipos de tumores.
- **Imágenes superficiales:** Disponer de una imagen nítida y de alta resolución de la superficie es de gran ayuda a la hora de controlar heridas o roturas, o a la hora de diagnosticar cualquier tipo de problema epidérmico o reacción alérgica.
- **Electrocardiogramas:** El electrocardiograma es el gráfico que se obtiene con un instrumento llamado electrocardiógrafo para medir la actividad eléctrica del corazón en forma de cinta gráfica continua. Tiene una función relevante en el cribado y la diagnosis de las enfermedades cardiovasculares. Este gráfico es la visualización de las corrientes eléctricas que se producen en el músculo cardíaco durante cada latido. Estas corrientes se registran mediante el electrocardiógrafo, y posteriormente pueden ser capturadas y representadas como imágenes digitales (ejemplo en Figura 2.6). [11]

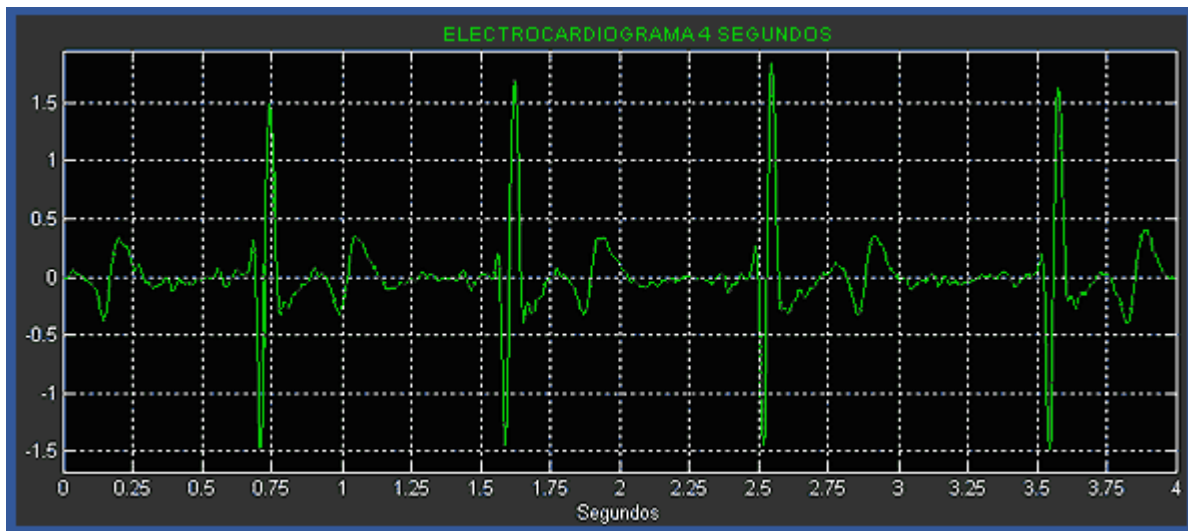


Figura 2.6: Imagen de un ECG digitalizado.

- **Otorrinolaringología:** La otorrinolaringología es una especialidad médico-quirúrgica donde se suman los conocimientos relativos al oído, nariz, faringo-laringe, patología del cuello y glándulas salivares. Gran parte del diagnóstico en esta especialidad se basa en la correcta visualización de las partes implicadas, por tanto es de vital importancia el uso de instrumentos que permitan obtener imágenes tanto de oídos (otoscopios), garganta (laringoscopios), etc.

2.4. Presión Arterial

La toma de presión o tensión arterial es utilizada para medir la fuerza o presión que ejerce la sangre en las paredes de las arterias. La sangre no fluye de forma continua, lo hace a borbotones y en oleadas que corresponden a cada latido del corazón.

La presión tiene dos medidas, la presión sistólica es la presión máxima y la diastólica que es la mínima, y se representa en números 120/80. La presión arterial puede presentar variaciones de persona a persona, así como, variaciones durante el día. Por lo general, cuando una persona duerme, la presión tiende a disminuir y cuando tiene actividad física la presión se eleva. El instrumento utilizado para la medición de la presión arterial se llama esfigmomanómetro o baumanómetro.

La presión alta puede desencadenar problemas cardíacos, renales, vasculares, etc. y es conocida como hipertensión arterial (HTA). Para este tipo de dolencias, es necesario una monitorización continua del paciente, que controle diariamente su presión. Es por ello que este tipo de técnica es especialmente indicada para su adaptación a la telemedicina. Son diversas las ventajas que se obtienen de la automedición de la presión arterial (PA):[12].

- Se evita la aparición del fenómeno de “bata blanca” y del efecto placebo. Esto permite una mejor aproximación al diagnóstico de la HTA
- Elimina el sesgo del observador.
- Mejor reproducibilidad que la toma de PA en la clínica.
- Más representativa del comportamiento de la PA en condiciones habituales.
- Permite conocer el perfil tensional diurno y facilita la implicación del paciente.
- Tiene buena correlación con la afectación de órganos diana.
- Contribuye a la mejor evaluación del efecto de la medicación antihipertensiva.
- Puede ser de utilidad en el diagnóstico de la HTA resistente.
- Puede lograr una reducción del coste farmacéutico y del número de visitas a los centros sanitarios.

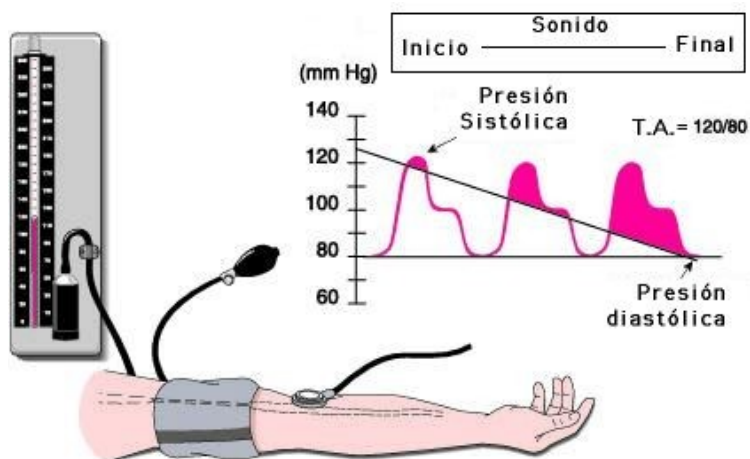


Figura 2.7: Colocación del esfigmomanómetro y gráfico de las medidas implicadas

2.5. Otras medidas

Aparte de las explicadas anteriormente, existen multitud de mediciones, niveles, constantes vitales... que también deben ser monitorizadas para un control más completo de la salud del paciente. Algunas de estas medidas que como ejemplo hemos añadido en nuestro sistema son:

- Peso del paciente.
- Nivel de colesterol.
- Triglicéridos (medidos como parte de una evaluación de factores de riesgo coronario).
- Nivel de glucosa (vital en el tratamiento y control de diabéticos, Figura 2.8).
- Ácido Láctico.
- Niveles hematológicos (hemoglobina, leucocitos, plaquetas, ...).



Figura 2.8: Medidor de glucosa (glucómetro)

Capítulo 3

Tecnología empleada

Una vez terminado con la introducción a los conceptos médicos que resultan la base de este trabajo y justo antes de pasar a relatar el funcionamiento del sistema objeto principal del proyecto, consideramos oportuno introducir una breve reseña de las tecnologías empleadas para la consecución del mismo, tanto para situar a lectores ya iniciados en aspectos tecnológicos relacionados con programación, comunicación utilizando la red Internet, seguridad informática o multimedia, como para aquellos lectores a quienes estos conceptos resulten relativamente desconocidos.

Evidentemente este trabajo no pretende ser un libro de aprendizaje ni consulta acerca de esas materias, para ello ya existe una extensísima bibliografía, si bien intentaremos introducir las nociones básicas para comprender todas estas piezas sin las cuales el motor, nuestro sistema, no sería posible.

3.1. Java y Netbeans

En este primer apartado vamos a hablar acerca de la plataforma de programación utilizada en el desarrollo de los dos programas que forman nuestro Sistema de Teleconsulta y Telediagnos.

En primer lugar vamos a hablar sobre la elección del lenguaje de programación. Para alcanzar con éxito los objetivos del programa a desarrollar necesitábamos utilizar un lenguaje que fuese a la vez potente, con gran soporte para aplicaciones de trabajo en red, con una buena interfaz gráfica y bien documentado. Además, y para ser coherente con los principios de todo el proyecto,

tanto el lenguaje como el entorno de desarrollo deberían ser libres y no atarnos a ningún tipo de plataforma ni compañía informática, es decir, que pudiese funcionar en cualquier tipo de ordenador siempre que los requisitos de hardware fuesen satisfechos.

Debido a estos motivos y a las satisfactorias experiencias anteriores del autor, y las consonantes recomendaciones del tutor, elegimos **Java** como lenguaje de programación.

Java se creó como parte de un proyecto de investigación para el desarrollo de software avanzado para una amplia variedad de dispositivos de red y sistemas embebidos o integrados (sistema informático de uso específico construido dentro de un dispositivo mayor). La meta era diseñar una plataforma operativa sencilla, fiable, portable, distribuida y de tiempo real. Cuando se inició el proyecto, C++ (la versión orientada a objetos del extendido lenguaje C) era el lenguaje del momento. Pero a lo largo del tiempo, las dificultades encontradas con C++ crecieron hasta el punto en que se pensó que los problemas podrían resolverse mejor creando una plataforma de lenguaje completamente nueva. [13].

Se extrajeron decisiones de diseño y arquitectura de una amplia variedad de lenguajes como Eiffel, SmallTalk u Objective C. El resultado es un lenguaje que se ha mostrado ideal para desarrollar aplicaciones de usuario final seguras, distribuidas y basadas en red en un amplio rango de entornos desde los dispositivos de red embebidos hasta los sistemas de sobremesa e Internet.

Java fue diseñado para ser:

- **Sencillo, orientado a objetos y familiar:** Sencillo, para que no requiera grandes esfuerzos de entrenamiento para los desarrolladores. Orientado a objetos, porque la tecnología de objetos se considera madura y es el enfoque más adecuado para las necesidades de los sistemas distribuidos y/o cliente/servidor. Familiar, porque aunque se rechazó C++, se mantuvo Java lo más parecido posible a C++, eliminando sus complejidades innecesarias, para facilitar la migración al nuevo lenguaje.
- **Robusto y seguro:** Robusto, simplificando la gestión de memoria y eliminando las complejidades de la gestión explícita de punteros y aritmética de punteros del C. Seguro para que pueda operar en un entorno de red.
- **Independiente de la arquitectura y portable:** Java está diseñado para soportar aplicaciones que serán instaladas en un entorno de red heterogéneo, con hardware y sistemas operativos diversos. Para hacer esto posible el compilador Java genera “bytecodes”, un formato de código independiente de la plataforma diseñado para transportar código eficientemente a través de múltiples plataformas de hardware y software.
- **Alto rendimiento:** A pesar de ser interpretado, Java tiene en cuenta el rendimiento, y particularmente en las últimas versiones dispone de diversas herramientas para su optimización. Cuando se necesitan capacidades de proceso intensivas, pueden usarse llamadas a código nativo.

- **Interpretado, multi-hilo y dinámico:** El intérprete Java puede ejecutar bytecodes en cualquier máquina que disponga de una Máquina Virtual Java (JVM). Además Java incorpora capacidades avanzadas de ejecución multi-hilo (ejecución simultánea de más de un flujo de programa) y proporciona mecanismos de carga dinámica de clases en tiempo de ejecución.

En cuanto a la plataforma de desarrollo, es decir, el “programa” desde el cual programamos, nos decidimos por la elección de un entorno que resultase sencillo, pero a la vez efectivo, y que estuviese enfocado a la programación de un modo gráfico. Por este último motivo consideramos que la mejor elección era realizar nuestro proyecto utilizando el entorno de desarrollo de código abierto **NetBeans** (Figura 3.1), desarrollado por una comunidad extensa de usuarios y cuyo principal patrocinador, Sun Microsystems, es la misma compañía creadora del lenguaje Java.

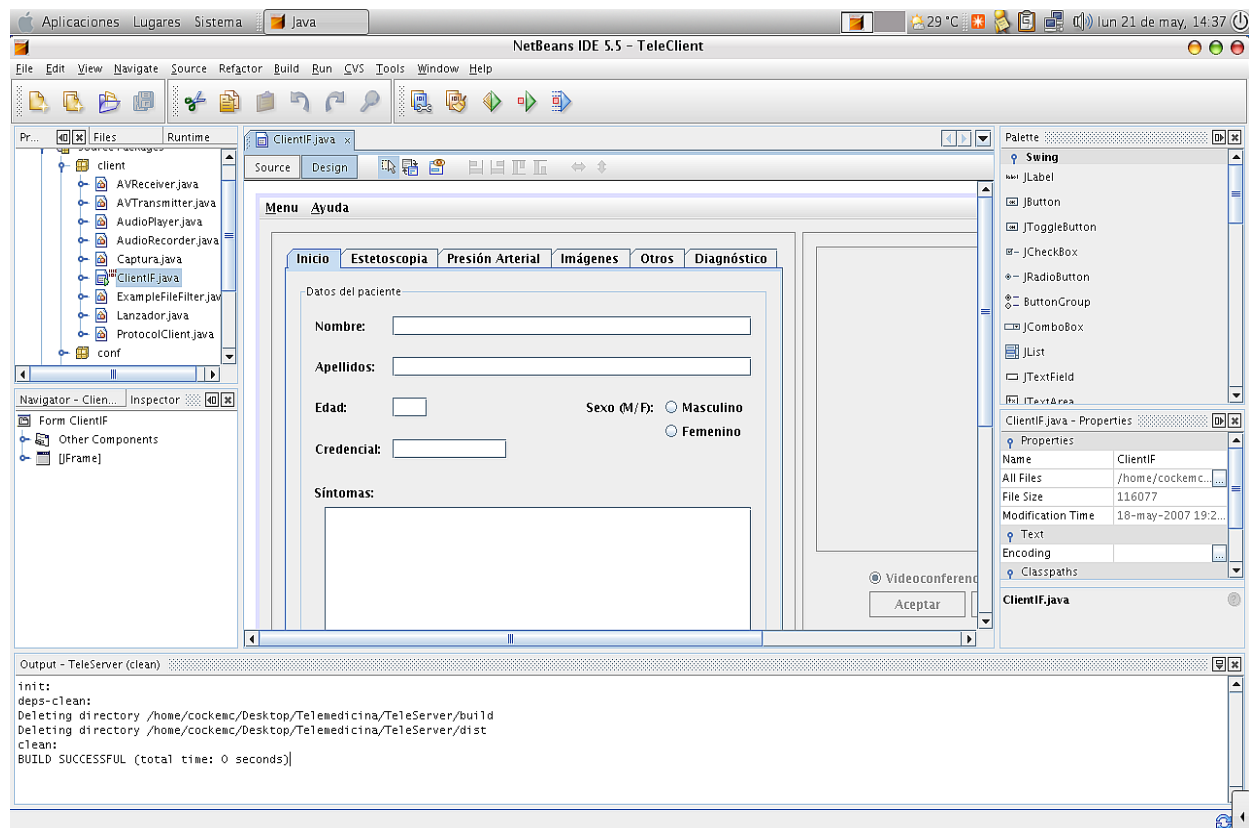


Figura 3.1: Netbeans funcionando en un entorno GNU/Linux Ubuntu 7.04

NetBeans es un entorno integrado de desarrollo (IDE) de código abierto cuya principal misión es hacer fácil el desarrollo de todo tipo de aplicaciones en la plataforma Java (JVM). [14] Con NetBeans es posible desarrollar tanto aplicaciones de escritorio, como aplicaciones empresariales en varias capas, o programas para todo tipo de dispositivos móviles. Es principalmente su sencillez para programación de aplicaciones gráficas de escritorio, con su característica de arrastrar y soltar los elementos gráficos observando en tiempo real cómo quedará finalmente la aplicación, la que motivó su elección para la programación de este proyecto.

NetBeans tiene una arquitectura modular que permite que con posterioridad se le añadan complementos (plug-ins), pero las funcionalidades que nos proporciona la instalación básica son tan numerosas, ricas y útiles que seguramente nunca tendremos necesidad de instalar uno de estos complementos adicionales.

Esta aplicación está escrito en el lenguaje Java, puro Java, por lo tanto se puede ejecutar en cualquier sistema en donde haya una máquina virtual de Java compatible, sin necesidad de depender de ninguna librería externa ni de enlace con el Sistema Operativo en que ejecutemos el programa.

3.2. Internet y Sockets

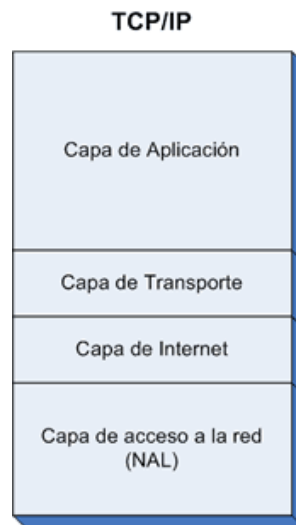
Una vez definidas las herramientas que se utilizaron para el diseño de nuestros programas, es necesario introducir la explicación de cómo éstos son capaces de comunicarse a distancia, transmitirse información de tipos muy diferentes y cómo logran crear la ilusión de encontrarse prácticamente en el mismo lugar, pese a que dos usuarios puedan encontrarse a miles de kilómetros.

Todo esto es posible gracias al uso que realiza nuestro sistema de Internet y la comunicación mediante sockets. Hoy en día es prácticamente imposible mantenerse ajeno y no tener como mínimo nociones acerca de lo que es y significa Internet para la sociedad actual. Tal es su importancia, que las sociedades modernas se atreven a considerar como los analfabetos del siglo XXI a aquellos que no sepan utilizar este medio. Pero, ¿en que consiste exactamente el concepto Internet?

Podemos definir Internet como un método de interconexión de computadoras, que garantiza que redes físicas heterogéneas funcionen como una red (lógica) única. De ahí que Internet se conozca comúnmente con el nombre de “red de redes”, pero es importante destacar que Internet no es un nuevo tipo de red física, sino un método de conexión. Internet es una red formada por la interconexión de otras redes menores. Aparece por primera vez en 1969, cuando la red ARPAnet, creada por el Departamento de “Defensa” de los EEUU establece su primera conexión entre tres universidades en California y una en Utah.

Está basada en la implementación de un paquete de protocolos denominado TCP/IP. Entendemos por protocolo al conjunto de reglas establecidas entre dos dispositivos para permitir la comunicación entre ambos. Se llama paquete o pila de protocolos TCP/IP (Figura 3.2) pues consiste en la combinación de una serie de estos protocolos, cada uno de los cuales se encarga de una función dentro de la tarea de comunicar dos entidades. [15]

- Los protocolos del nivel de acceso a la red incluyen la **capa física**, que encargan de definir las características físicas de la comunicación, como las convenciones sobre la naturaleza del medio usado para la comunicación, y todo lo relativo a los detalles como los conectores, código de canales y modulación, sincronización, etc. y también la capa de **enlace de datos**, que define cómo son transportados los paquetes sobre el nivel físico, en forma de tramas, incluido los delimitadores.



*Figura 3.2: Pila de protocolos
TCP/IP*

- En la capa de **Internet** los protocolos se encargan de realizar las tareas básicas para conseguir transportar datos desde un origen a un destino, como encapsular la información a enviar en paquetes y realizar el enrutamiento de estos paquetes hasta alcanzar su destino. El propósito de la capa de Internet es enviar paquetes origen desde cualquier red en la red y que estos paquetes lleguen a su destino independientemente de la ruta y de las redes que recorrieron para llegar hasta allí. El protocolo específico que rige esta capa se denomina Protocolo Internet (IP). En esta capa se produce la determinación de la mejor ruta y la conmutación de paquetes. Esto se puede comparar con el sistema postal. Cuando enviamos una carta por correo, no sabemos cómo llega a destino (existen varias rutas posibles); lo que nos interesa es que la carta llegue. En Internet las direcciones de las “casas”, que en este caso serían los nodos de la red, y los hosts origen y destino, se llaman direcciones IP, y en la versión 4 de este protocolo están compuestas por 4 números de 8 bits (de 0 a 255) separados por un punto: Ej. 192.168.1.34
- La capa de **transporte** se refiere a los aspectos de calidad del servicio con respecto a la fiabilidad, el control de flujo y la corrección de errores. Uno de sus protocolos, el protocolo para el control de la transmisión (TCP), ofrece maneras flexibles y de alta calidad para crear comunicaciones de red fiables, sin problemas de flujo y con un nivel de error bajo. TCP es un protocolo orientado a la conexión. Mantiene un diálogo entre el origen y el destino mientras empaqueta la información de la capa de aplicación en unidades denominadas segmentos.
- Capa de **aplicación**: Los diseñadores de TCP/IP pensaron que los protocolos de nivel superior deberían incluir los detalles de las capas que en el modelo de referencia internacional OSI se llaman de sesión y presentación. Simplemente crearon una capa de aplicación que maneja protocolos de alto nivel, aspectos de representación, codificación y control de diálogo. El modelo TCP/IP combina todos los aspectos relacionados con las aplicaciones en una sola capa y garantiza que estos datos estén correctamente empaquetados para la siguiente capa.

Para aclarar mejor todos estos conceptos y poner ejemplos de protocolos para cada nivel, ofrecemos este gráfico de protocolo. Este gráfico ilustra algunos de los protocolos comunes especificados por el modelo de referencia TCP/IP. También se puede observar claramente una de las máximas de la arquitectura TCP/IP, que sería “todo sobre IP, IP sobre todo”, y significa que IP es el cuello de botella donde convergen todos los protocolos: los de alto nivel lo utilizan para mandar sus mensajes, y los de bajo nivel deben encapsular finalmente los paquetes IP cada uno en su forma determinada:

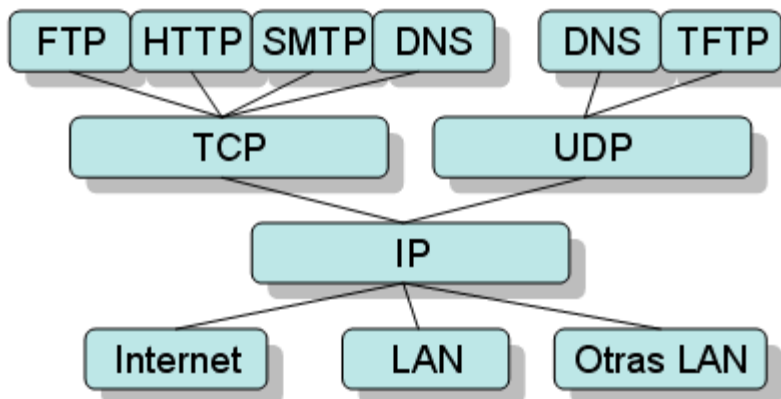


Figura 3.3: Gráfico de protocolos

3.2.1. Sockets TCP

Necesitamos de un mecanismo que permita la comunicación entre dos programas de Java a través de la red, y nos sirva para transmitir datos de uno a otro. La gestión de esta comunicación se realiza a través de los llamados **sockets** de comunicación y de streams de envío y recepción de datos generados sobre dichos sockets. Cada socket tiene un stream de salida (OutputStream) y un stream de entrada (InputStream) que permite la comunicación bidireccional entre dos terminales conectadas a una red.[16].

El establecimiento de sockets de comunicación, la creación de streams asociados a ellos y el envío de los datos a través de estos elementos previamente generados, siguen en Java un protocolo concreto.

El protocolo a seguir exige que una de las partes implicadas en la comunicación, que haría la función de servidor, se encuentre “escuchando” en uno de los puertos TCP de su computadora. Estos puertos harían la función de los números de apartamento dentro de un edificio. La dirección del edificio equivaldría entonces a la dirección IP, pero sólo con ella no basta, necesitamos otro parámetro que nos indique dentro de ese edificio a cuál de los habitantes va dirigido un paquete. Esa es la función de los puertos de comunicación.

Entonces, el programa cliente no sólo debe conocer la IP del servidor, sino también saber en qué puerto está escuchando éste. Así, se establece que el ordenador configurado para “escuchar” ha de lanzar un servidor a un puerto concreto. De este modo, el servidor quedará a la espera de una llamada de algún terminal que desee comunicarse con él.

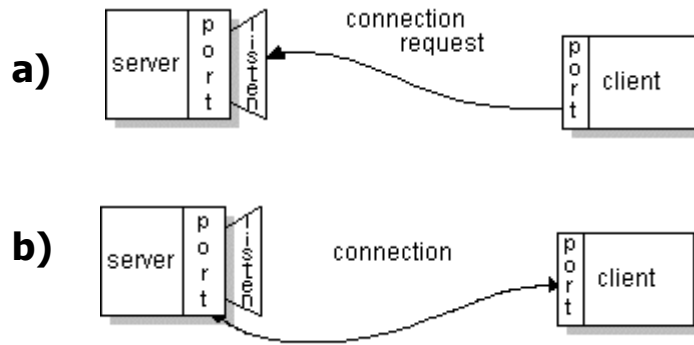


Figura 3.4: Esquema de creación de sockets:
a) Llamada del cliente al puerto de escucha del servidor.
b) Respuesta de aceptación del servidor

Para lanzar la creación de los sockets, el terminal que “llama”, deberá mandar una petición a la dirección IP y al puerto correspondiente del servidor de sockets del otro terminal (ver Figura 3.5-a), el cual, al recibir esta llamada, responderá generando el socket propio y devolviendo el permiso y los datos que hagan posible la generación del socket en el otro extremo de la comunicación (Figura 3.5-b). Un socket queda definido por una dirección IP, un protocolo y un número de puerto.

Una vez generados los dos sockets que enlazan los dos puntos de acceso, sólo restará asociar a cada uno de ellos los streams (de entrada y salida de datos) en cada uno de los terminales entre los cuales deseamos establecer la comunicación, que serán los que nos permitan el envío y recepción de datos entre el programa y la red.

Una vez establecidos los sockets y generados los streams, ya se podrá proceder al envío y recepción de datos entre los dos terminales. En Java, el envío de datos se realiza cargando la información sobre el OutputStream alojado en el socket local. Al recibir la información, el sistema la guarda en un buffer (o cola) en el que se almacenan los datos que se encuentran a la espera de ser enviados por el socket a través del sistema de comunicación.

Conforme el socket va enviando los datos, esta cola va avanzando hasta que el paquete de datos termina siendo enviado por la red, empleando una conexión basada en el protocolo TCP/IP, hasta el socket del programa destino. Éste tiene, a su vez, un InputStream en el que se van almacenando los datos recibidos en espera de que sean leídos o extraídos por el programa principal que corre en el punto de destino. Al realizar la lectura sobre el socket destino, el InputStream se va vaciando, hasta que el paquete de datos enviado llega a su destinatario final, cerrando así la comunicación.

3.3. Seguridad con SSL y certificados

De cara a ofrecer un producto con las máximas garantías de privacidad y autenticidad es necesario hacer uso de mecanismos que nos permitan lograr esos objetivos, pues al estar trabajando con programas orientados al trabajo en red nos estamos exponiendo al uso indebido de la información que transmitimos, o a otros tipos de fraudes informáticos como la suplantación de identidad.

Lógicamente, nadie quiere que la información que intercambia con un especialista sea leída por otras personas, no importa el uso que esas terceras personas quisieran hacer con esos datos. Es por ello que un programa de estas características debe hacer uso de mecanismos de **encriptación** de los datos transmitidos, para ofrecer al usuario la confidencialidad deseada.

Por otro lado, si bien los datos pueden estar muy bien protegidos, con sistemas de encriptación potentes y contraseñas irrompibles, no es suficiente para completar un sistema de comunicación totalmente seguro, pues podría darse la situación de no estar seguros de quién es la persona que se encuentra al otro lado. Diariamente encontramos en las páginas especializadas en tecnología centenares de casos de fraudes por suplantación de identidad, en bancos, etc. En el caso de nuestro sistema, el paciente quiere cerciorarse que la persona que le va a atender es ciertamente un especialista titulado y con todos los permisos para ejercer su profesión. Para conseguir esta meta, debemos proporcionar a nuestros programas de un sistema de **autenticación** que nos atestigüe fielmente quién es quién en este juego.

Para la consecución en nuestro sistema de estos dos factores, privacidad y autenticidad, hemos elegido la utilización por una parte de sockets seguros **SSL**, y por otra de **certificados** de autenticación.[17]

SSL es un acrónimo de Secure Socket Layer (capa de sockets seguros), y es básicamente un protocolo desarrollado por Netscape Communications Corporation (la empresa que desarrolló el primer navegador comercial, cuyo núcleo es utilizado por navegadores actuales como Mozilla Firefox). Fue pensado para transmitir documentos privados vía Internet, proveyéndole confiabilidad y autenticidad. SSL funciona como una capa añadida entre los protocolos de Internet y las aplicaciones, es decir, el protocolo es independiente de la aplicación, haciendo posible su utilización conjuntamente con FTP, telnet o cualquier otra aplicación como HTTP.

SSL opera sobre la capa de transporte (TCP), por lo que puede ofrecer seguridad a cualquier protocolo que utilice comunicaciones orientadas a conexión fiables. Es utilizado conjuntamente con HTTP para formar HTTPS, el cual es empleado para asegurar páginas web por ejemplo de comercio electrónico, por medio de certificados de clave pública para verificar la identidad del sitio. Cabe aclarar que un **certificado de autenticación** es un documento digital mediante el cual un tercero confiable (una autoridad de certificación, como un banco importante o una empresa como VeriSign) garantiza la vinculación entre la identidad de un sujeto o entidad y su clave pública.

Protocolos SSL

SSL es en realidad no sólo un único protocolo, sino un conjunto de protocolos que pueden adicionalmente ser divididos en dos capas:

- Protocolos para garantizar la seguridad e integridad de los datos (SSL Record Protocol).
- Protocolos diseñados para establecer una conexión SSL. Se utilizan tres protocolos en esta capa: SSL Handshake Protocol, SSL ChangeCipher Protocol y el SSL Alert Protocol.

La pila de protocolos SSL es la siguiente (Figura 3.5):

SSL handshake protocol	SSL cipher change protocol	SSL alert protocol	Application Protocol (eg. HTTP)
SSL Record Protocol			
TCP			
IP			

Figura 3.5: Pila de protocolos SSL

Fases del protocolo

SSL Request:

Para configurar SSL hay que realizar una petición. Normalmente un cliente solicita una URL a un servidor que soporte SSL. Una vez la petición se ha hecho, cliente y servidor empiezan la negociación de los parámetros de la conexión SSL mediante el protocolo SSL Handshake.

SSL Handshake protocol

Durante la fase del protocolo SSL Handshake, el cliente y el servidor intercambian una serie de mensajes para negociar el contrato de seguridad. El protocolo incluye los siguientes seis pasos:

- Fase “Hello”, utilizada para acordar el conjunto de algoritmos para la preservación de la privacidad y la autenticación
- La fase de intercambio de claves, al final de la cual ambas partes comparten el mismo secreto.
- Fase de producción de la clave de sesión, que será la utilizada para cifrar los datos intercambiados.
- Fase de verificación del servidor.
- Fase de autenticación del cliente, donde el servidor pide a éste un certificado de tipo X.509.

- Fase de finalización, que indica que la sesión segura ya puede comenzar.

En la siguiente figura (Figura 3.6) podemos observar gráficamente el intercambio de mensajes entre estas dos entidades:



Figura 3.6: Esquema de intercambio de mensajes en fase "Handshake"

SSL Record Protocol

El protocolo SSL Record incumbe el utilizar SSL de un modo seguro y con integridad de mensajes asegurada. Es utilizado por los protocolos SSL superiores. Especifica el modo de encapsular los datos transmitidos y recibidos. El "record data" construido está formado por los siguientes elementos:

- MAC-DATA, el "message authentication code"
- ACTUAL-DATA, los datos de aplicación a ser transmitidos
- PADDING-DATA, los datos requeridos de relleno cuando se utilice algún tipo de cifrado en bloque.

Transferencia de datos

A partir de este momento, un canal de transmisión segura ha sido establecido, por lo que el intercambio de datos es ya posible. Cuando el cliente o el servidor quieren mandar un mensaje, se crea una especie de “huella dactilar” utilizando el algoritmo de “hash” o resumen, que es encriptada con el mensaje. Posteriormente, el receptor descriptará los datos y comprobará que, aplicando la misma función que el emisor, obtiene la misma huella dactilar que se le envió, por tanto tiene la garantía de que los datos no han sido modificados de forma alguna.

Fin de sesión SSL

Suele ser representado por el cierre del programa que utilizaba SSL (como en el caso de nuestro proyecto), o por ejemplo por la advertencia por parte del navegador de que ya no se navega por una página segura.

3.4. Multimedia con JMF y RTP

Como ya comentamos en el capítulo 2, de cara a realizar un sistema de Teleconsulta y Telediagnóstico considerábamos de vital importancia la existencia de un contacto visual entre paciente y especialista, con objeto de conseguir una consulta que transmita cercanía y confianza. Es por ello que uno de nuestros objetivos para nuestro proyecto era que contara con un sistema de videoconferencia, sencillo de utilizar pero con efectividad. Como objetivo secundario también nos propusimos que, en caso de problemas de falta de hardware o lentitud de la red, la opción de realizar conferencia sólo de voz también fuese posible.

Para conseguir estos objetivos, debíamos comenzar a manejar conceptos y datos de tipo multimedia, para lo cual no era suficiente con la extensa librería de clases de java, sino que había que añadir una biblioteca externa que nos permitiera un manejo sencillo de este tipo de datos en tiempo real, es por ello que hicimos uso de **Java Media Framework**, el API de Sun Microsystems diseñado para manejar este tipo de datos.

Una vez capturados estos datos “en directo”, utilizaremos el protocolo RTP (Real-time Transport Protocol) para mandar esta información a través de la red, concepto conocido como “streaming”.

Java Media Framework proporciona a los applets y aplicaciones Java la capacidad de reproducir, capturar y transmitir/recibir en tiempo real audio vídeo y otros contenidos multimedia. Provee de una serie de codificadores y decodificadores para los formatos multimedia más relevantes siendo capaz además, de realizar transcodificación entre dichos formatos.[18].

Una aplicación multimedia es aquella que produce, reproduce, procesa o maneja uno o varios contenidos multimedia. A su vez un contenido multimedia es aquel que está compuesto de diversos “medios”, como pueden ser audio, vídeo, texto, etc. Decimos que un contenido multimedia está basado en el tiempo en tanto que cada uno de sus medios cambia

significativamente con él. Esta característica hace que un contenido multimedia requiera ser proporcionado y procesado en unas condiciones temporales estrictas. Por ejemplo cuando se reproduce un vídeo, si los datos multimedia no pueden ser proporcionados lo suficientemente rápido pueden producirse pausas y retardos en la reproducción; por otro lado si los datos no pueden ser recibidos y procesados lo suficientemente rápido el vídeo se reproduce a saltos en tanto que se desechan cuadros como medio para mantener la tasa de reproducción.

Cada uno de los medios de los que se compone un contenido multimedia se denomina pista. Por ejemplo un contenido multimedia correspondiente a una videoconferencia puede contener una pista de audio y otra de vídeo. Se dice que las pistas que componen un contenido multimedia están multiplexadas, al proceso de extracción de las distintas pistas que componen un contenido multimedia se le denomina demultiplexación. Existen distintos tipos de pistas en función del tipo de datos que contienen, como audio y/o vídeo; a su vez cada pista posee un formato que define como están estructurados los datos que forman parte de ella.

En Java Media Framework los datos multimedia pueden proceder de diversas fuentes, como archivos locales o remotos y vídeo y audio en tiempo real o bajo demanda. Una fuente de datos multimedia se modela mediante un objeto `DataSource`. Podemos crear una **`DataSource`** directamente a través de una URL (Universal Resource Locator) o bien mediante un objeto de tipo `MediaLocator`. Por otra parte, los datos para ser exportados (bien sea para ser almacenado en un archivo local o para su posterior retransmisión vía Internet) son modelados mediante un objeto **`DataSink`**.

El API de JMF nos proporciona métodos sencillos para, desde cualquier aplicación Java, acceder y controlar los dispositivos de captura que existan en nuestro sistema, como pueden ser micrófonos o cámaras web. Estos son los dos dispositivos que vamos a utilizar en nuestro sistema, aunque como explicamos con anterioridad, tanto el programa cliente como el servidor están preparados para realizar conferencias con o sin vídeo (sólo audio).

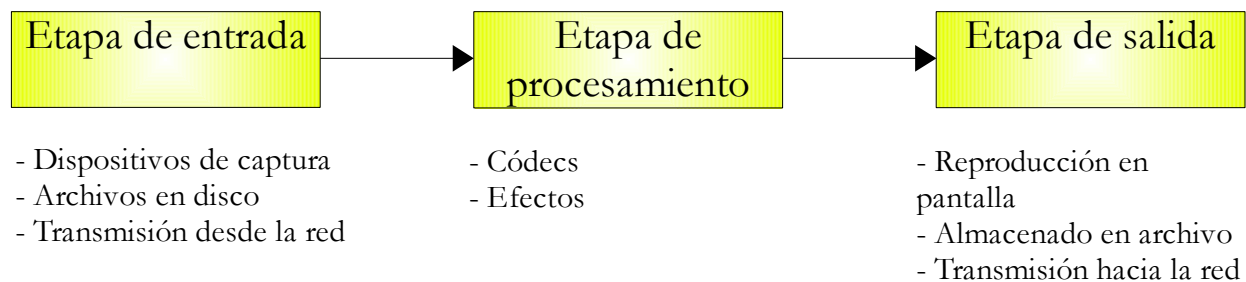


Figura 3.7: Etapas de la arquitectura JMF

Arquitectura de JMF

En la etapa de entrada (Figura 3.7), se leen los datos desde una fuente y son pasados mediante buffers a la etapa de procesado. JMF está preparado tanto para leer datos desde un dispositivo de captura (como comentábamos en el punto anterior), como desde un archivo local o bien desde una transmisión recibida de la red (que también utilizaremos en nuestro sistema para recibir la videoconferencia del servidor o cliente).

La etapa de procesado consiste en un número de códecs (codificador-decodificador) y efectos diseñados para modificar la trama de datos (stream) y prepararla para proporcionarla a la etapa de salida. Estos códecs pueden realizar funciones de compresión o descompresión de datos en diferentes formatos, eliminar ruido, o aplicar algún tipo de efecto (por ejemplo, un efecto de eco). En nuestro caso utilizamos los códecs H.263 para codificación de vídeo, y DVI para codificación de audio, pues son dos códecs recomendados para este tipo de aplicaciones, por su buena relación calidad-ancho de banda utilizado. También en esta etapa realizamos la multiplexación de las dos pistas en una sola fuente, para ser transmitida conjuntamente y evitar problemas de sincronización audio/vídeo.

Una vez la etapa de procesamiento ha aplicado las transformaciones al stream, traslada la información a la etapa de salida, encargada de finalmente almacenar los datos, presentarlos en pantalla (mediante el uso de unos componentes llamados “players” diseñados para tal función) o bien para transmitirlos por la red, para lo que JMF hace uso del protocolo RTP.

Protocolo Real-time Transport Protocol

RTP es un protocolo de nivel de aplicación (no de nivel de transporte, como su nombre podría hacer pensar) utilizado para la transmisión de información en tiempo real, como por ejemplo audio y vídeo en una videoconferencia.

La transmisión de medios a través de la red en tiempo real requiere una gran capacidad de la red. En este tipo de datos es más importante que lleguen sin una gran demora, que lo hagan completamente. Es decir, se tolera descartar algunos datos, siempre que el conjunto llegue sin retrasos (por ejemplo, en una conversación telefónica esto es muy importante, porque de otro modo sería imposible entender a la persona en el otro lado). Es por este motivo que los protocolos utilizados para datos estáticos no funcionan bien para la transmisión de medios en tiempo real..

TCP es un protocolo diseñado para transmisión confiable en redes de bajo ancho de banda y gran tasa de errores. Cuando un paquete se pierde o está corrupto, se retransmite. Todos estos mecanismos para asegurar la integridad de datos son causantes por otro lado de una baja respuesta en la tasa de transmisión.

Por este motivo otros protocolos deben ser utilizados para transmitir datos multimedia. Normalmente se trabaja con UDP (User Datagram Protocol), la versión de protocolo de transporte no orientado a conexión. Es un protocolo no confiable, no garantiza que los datos lleguen completamente a su destino, ni que lo hagan en el orden en que fueron enviados. Es misión del destinatario corregir estos errores si fuesen necesarios.

De este modo, se pueden implementar sobre UDP otros protocolos específicos para cada aplicación. El estándar de Internet para transporte de datos en tiempo real es RTP, protocolo que viene definido en el RFC 3550 del IETF.[19].

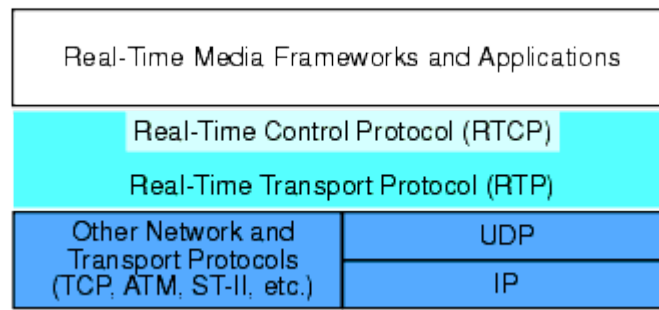


Figura 3.8: Arquitectura de RTP

RTP proporciona servicios de entrega punto a punto, del emisor al receptor, de datos en tiempo real. RTP es independiente de la red y del protocolo de transporte sobre el que esté implementado, a pesar de que como hemos indicado es a menudo utilizado sobre UDP.

RTP puede ser utilizado tanto sobre servicios de red de tipo unicast o multicast. En unicast es el emisario quien se encarga de enviar copias separadas de los datos a cada destinatario, en cambio en la transmisión multicast es la red la responsable de transmitir los datos a diversos destinatarios, mientras el emisor sólo envía los datos una vez. Este modo de funcionamiento, más eficiente para aplicaciones de videoconferencia, es el que hemos utilizado en nuestro proyecto, a pesar de que sólo participen en la conversación un solo emisario y receptor.

Capítulo 4

Funcionamiento del Sistema

Una vez establecidos todos los conceptos teóricos acerca de las pruebas médicas que nuestro sistema va a realizar, y tras la explicación de las tecnologías empleadas para conseguir los objetivos de funcionamiento planteados con anterioridad, estamos ya en disposición de adentrarnos en el funcionamiento propio del sistema, objeto de nuestro proyecto.

Primeramente explicaremos las partes en que el sistema está compuesto, para posteriormente centrarnos en la instalación de todos los elementos necesarios para su correcto funcionamiento, y finalmente explicar con detalle todas las funciones que podemos realizar, tanto desde la parte del paciente, como desde la parte del especialista.

4.1. Cliente y servidor

Nuestro sistema consta de dos partes, de dos programas diferentes. La parte concerniente al especialista es una aplicación de escritorio para el manejo de forma gráfica de todos los datos que el paciente le irá proporcionando a través de la red (videoconferencia, imágenes, sonidos, etc.). Este programa hará las funciones de **servidor**, pues será el programa del especialista el primero que será iniciado desde una máquina conocida (en un hospital, en su clínica particular, o incluso en su casa), a la espera de llamadas por parte de pacientes. Cuando una de estas llamadas llegue, el sistema quedará bloqueado, pues físicamente sólo se puede atender con garantías a pacientes de uno en uno. Este programa será conocido como **Telemédicmai Server** (ver Figura 4.1).



Figura 4.1: Ventana de información con logotipo del sistema

Por otro lado, la contraparte del programa del especialista será la que utilizará el paciente. Si bien es recomendado que sea monitorizado y auxiliado por un asistente (otro médico, un enfermero o bien un técnico), la naturaleza sencilla y autoexplicativa del programa le permitiría al paciente en alguna ocasión hasta utilizarlo sin supervisión.

Esta parte del paciente hará las funciones de programa **cliente**, será conocida como **Telemédomai Client** y efectuará llamadas a alguno de los servidores de la red de telemedicina que deberán estar repartidos en diferentes lugares. La situación normal sería que uno o varios especialistas estuviesen localizados en diferentes hospitales o centros, y todos ellos con sus programas iniciados esperando la llamada de pacientes para atenderles. Es por ello necesario que el programa cliente maneje una lista de especialistas con sus respectivas direcciones IP, para que el programa realice la llamada adecuada. Esta lista debería ser proporcionada y gestionada por la entidad encargada del sistema de telemedicina (Seguridad Social, sanidad privada, alguna ONG, ...).

Queda añadir sólo que el sistema está preparado para volver al estado inicial siempre que ocurra un fallo en la conexión, por lo que el servidor volvería a estar preparado para recibir la llamada de un paciente, y el paciente podría en caso de desconexión realizar con un simple clic otra llamada con los mismos u otros datos a cualquiera de los especialistas.

4.2. Instalación

Para conseguir el correcto funcionamiento de nuestro programa necesitamos a parte de la aplicación cliente y servidor necesitamos tener correctamente instalados tanto la máquina virtual Java (indispensable para el funcionamiento de cualquier programa Java), el API multimedia JMF y por supuesto los certificados de autenticación correctamente firmados por la autoridad pertinente.

4.2.1. Instalación de cliente y servidor

Para la instalación de los programas cliente servidor podemos realizarlo de diversas formas, dependiendo del sistema operativo que estemos utilizando. Todos los archivos nombrados aquí pueden encontrarse acompañando este trabajo, o bien pueden descargar la última versión desde <http://www.jorgemarti.com/projects/teledomai>.

Para Windows ®

Para instalar el cliente o servidor en una máquina con sistema operativo Windows, simplemente se debe ejecutar o bien el archivo de instalación “teleserver.exe” o bien “teleclient.exe”. Una vez iniciado, el programa de instalación guiará al usuario por todo el proceso (ver figura 4.2):

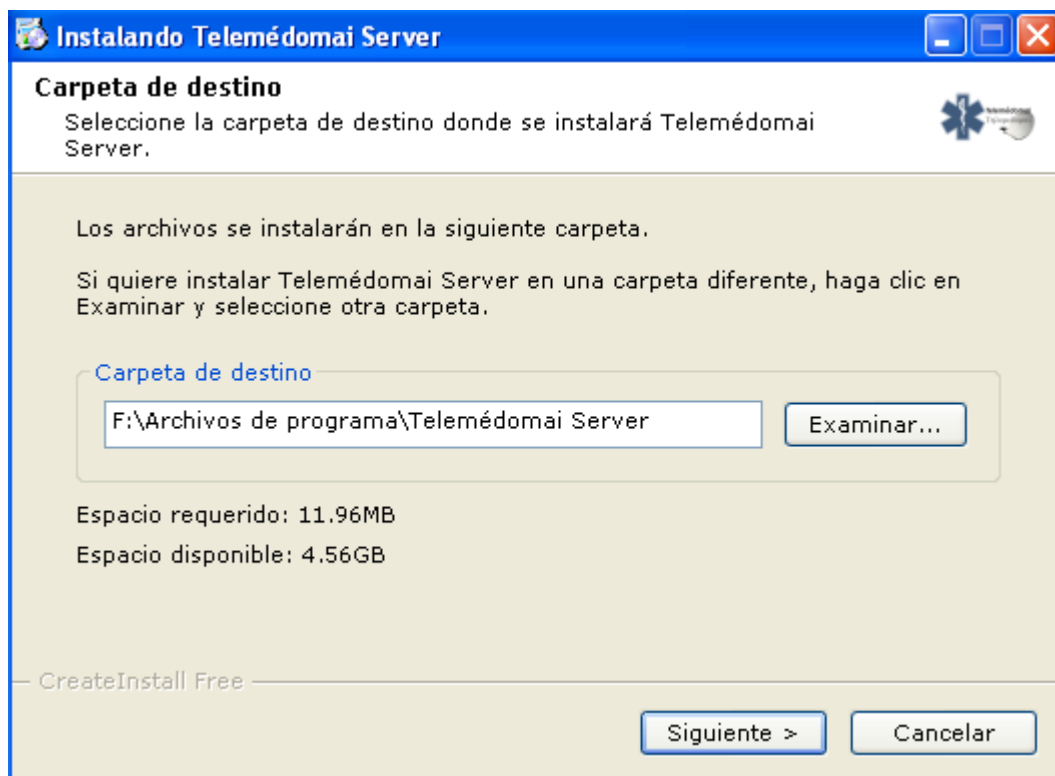


Figura 4.2: Programa de instalación del servidor

Para GNU/Linux, MacOS X y otros

Simplemente descomprimir los archivos “teleserver.zip” o “teleclient.zip” en el directorio deseado. Para descomprimir puede utilizarse cualquier programa de descompresión de archivos ZIP que incluya el sistema. Este modo de instalación también puede ser realizado en un sistema Windows. Puede utilizar cualquier descompresor, pero el programa libre y gratuito 7-Zip sería una buena opción (www.7-Zip.org).

4.2.2. Instalación de la máquina virtual Java y JMF

Para instalar la máquina virtual de Java, sea cual sea el sistema operativo, pueden visitar la web oficial de Sun Microsystems para esta tecnología (<http://www.java.com/es/download/>). Una vez allí pueden seguir las instrucciones detalladas en la propia página y descargar la última versión del JRE (Java Runtime Environment).

Hay que añadir que este paso no es necesario para sistemas MacOS X pues ya vienen preinstalados con su propia versión de la máquina virtual.

En cuanto al Java Media Framework, la cuestión es un poco problemática, pues además de ser un API poco frecuentemente actualizado por Sun, existen enormes problemas de compatibilidades en sistemas que no sean Windows o Solaris. Es por ello que nuestro programa, pese a ser probado completamente con todas sus funciones tanto para Windows, GNU/Linux y MacOS X, sólo ofrece la posibilidad de realizar videoconferencias en sistemas Windows. Esperamos este cierto abandono del framework multimedia de Java sea prontamente solucionado y podamos finalmente ofrecer el 100% de funcionalidades de nuestro sistema a todas las plataformas.

Mientras tanto, si queremos instalar la versión del JMF para Windows, podemos o bien hacerlo desde el propio programa de instalación del cliente o servidor (una vez terminados éstos, el instalador lanza el programa de configuración del JMF), o si escogimos la opción de descompresión manual del programa, acudir a la web oficial del API de Sun Microsystems (<http://java.sun.com/products/java-media/jmf/>).

4.2.3. Creación e instalación de los certificados firmados

Como explicamos en el apartado de seguridad de nuestro programa (sección 3.3), el sistema hace uso de autenticación de los usuarios mediante el empleo de certificados firmados. Para que un especialista o un centro de atención de pacientes puedan hacer uso del programa, deben conseguir su certificado de autenticidad firmado por parte de la autoridad pertinente, que dependiendo de la implantación del sistema podría ser el servicio de Salud Pública, alguna empresa privada de sanidad o incluso una ONG de carácter médico.

La herramienta que hemos utilizado para la configuración de los certificados digitales es **keytool**, una utilidad de administración de claves y certificados, que viene por defecto junto con Java. Esta utilidad permite a los usuarios administrar sus propios pares de claves pública/privada y los certificados asociados para autenticarse. También permite a los usuarios almacenar en una caché las claves públicas, en forma de certificados, de otros usuarios con quien se comunican, almacenando las claves y certificados en un archivo llamado “keystore”, el cual está protegido mediante contraseña.

Para la realización de las pruebas de nuestro sistema, creamos una Autoridad Certificadora que nos sirviera para realizar las funciones que en una aplicación real del sistema realizaría la

autoridad que comentamos en párrafos anteriores: controlar los usuarios con permiso para utilizar el sistema, y otorgarle a éstos los certificados firmado para que puedan hacer uso del mismo. Para crear esta Autoridad Certificadora (CA) virtual, hicimos uso del paquete de herramientas de administración y librerías relacionadas con la criptografía **OpenSSL**, el cual es libremente distribuido. Este paquete se puede descargar e instalar desde su página oficial.

Vamos a relatar los pasos seguidos para obtener como resultado los certificados firmados, tanto para un especialista como para el usuario del programa cliente, transcribiendo exactamente los comandos utilizados:

- Primero, creación de la autoridad certificadora:

```
>> openssl req -new -x509 -keyout ca.key -out ca.crt -days 365
```

Parte Servidor

- Creación del par de claves para cada usuario del servidor (para cada médico). El usuario y contraseña debe ser el mismo que se usa en el programa. Ejemplo con usuario “jorge”.

```
>> keytool -genkey -alias jorge -keypass jorgepassword -validity 365
-storepass jorgepassword -keystore jorge.keystore
```

- Creamos una petición de certificado firmado, que sería enviada a la autoridad certificadora real, o la oficial de nuestro sistema, para cada usuario del mismo:

```
>>keytool -certreq -alias jorge -file jorge.csr -keypass jorgepassword
-storepass jorgepassword -keystore jorge.keystore
```

- La autoridad certificadora (CA) firma las peticiones de certificado recibidas por parte del nuevo usuario del sistema. Estos certificados firmados se devuelven al usuario (el comando “-Cacreateserial” sólo se añadirá para el primer usuario):

```
>>openssl x509 -req -in jorge.csr -out jorge.cer -CA ca.crt -CAkey ca.key
-Cacreateserial
```

- Instalamos el certificado de la CA en el “cacerts keystore” para que sea reconocido por el sistema. Muy importante tener bien escrita la variable del sistema %JAVA_HOME%.

```
>>keytool -import -alias ca -keypass capassword -file ca.crt -keystore
"%JAVA_HOME%/jre1.5.0_11/lib/security/cacerts" -storepass changeit
```

- Instalamos el certificado de la CA también en los keystores de los usuarios, para que estos reconozcan a la autoridad certificadora.

```
>>keytool -import -alias ca -file ca.crt -keystore jorge.keystore
```

- Finalmente instalamos los certificados firmados por la CA:

```
>>keytool -import -trustcacerts -file jorge.cer -keystore jorge.keystore -alias jorge -keypass
jorgepassword -storepass jorgepassword
```

Parte Cliente

Para la parte del cliente, el procedimiento sería exactamente el mismo que la parte del servidor. Simplemente se cambiaría el nombre del usuario “jorge”, que sería un especialista, a un nombre de paciente genérico, que en nuestro programa hemos utilizado “paciente”.

Finalmente, tanto si se está en la parte servidor o en la parte cliente, debemos dejar una copia de los archivos “.keystore” en el directorio “SSL” de nuestro programa, pues es allí donde ambas aplicaciones intentarán buscar los keystores y de allí comprobar las claves de los usuarios. Si no se realiza correctamente el proceso, o si el archivo keystore del usuario (por ejemplo, “paciente.keystore”) no se encuentra en el directorio adecuado, el programa no funcionará, pues nos dará un error de log in.

4.3. Realizar modificaciones del código de los programas

Para realizar modificaciones del código, primero hay que descomprimir los archivos teleclientsource.zip o teleserversource.zip que encontrarán junto los archivos del programa en cualquier directorio de su elección. Se creará un directorio con el nombre “TeleClient” o bien “TeleServer”, que incluyen el código de cada uno de los programas.

Para editar el código puede entrar en la carpeta SRC y abrir los archivos con cualquier editor, pero recomendamos trabajar con el proyecto completo y utilizar el entorno Netbeans. Simplemente arranque Netbeans y seleccione la opción “Open Project” del menú principal (figura 4.3).

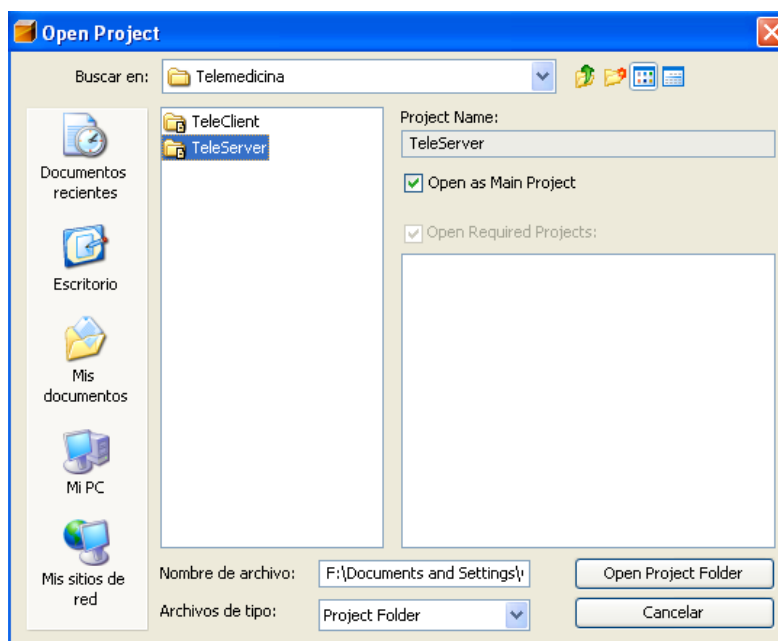


Figura 4.3: Ventana de abrir proyectos existentes en Netbeans

Una vez abierto, Netbeans ya le permite editar el código de todas las clases incluidas en el proyecto, e incluso hacer modificaciones de la interfaz de modo gráfico, haciendo doble clic sobre la clase ClientIF.java o ServerIF que encontrarán navegando por la jerarquía del proyecto.

Finalmente, tras realizar las modificaciones deseadas, se puede volver a generar el archivo ejecutable de tipo JAR seleccionando la opción “Build Project” que aparece en el menú contextual al clicar con el botón derecho sobre un proyecto (Figura 4.4). Para ejecutarlo, no olvide primero colocar los certificados correctamente en la carpeta SSL donde se encuentre el programa (donde se ejecute el archivo JAR), pues de otro modo el programa no funcionaría.

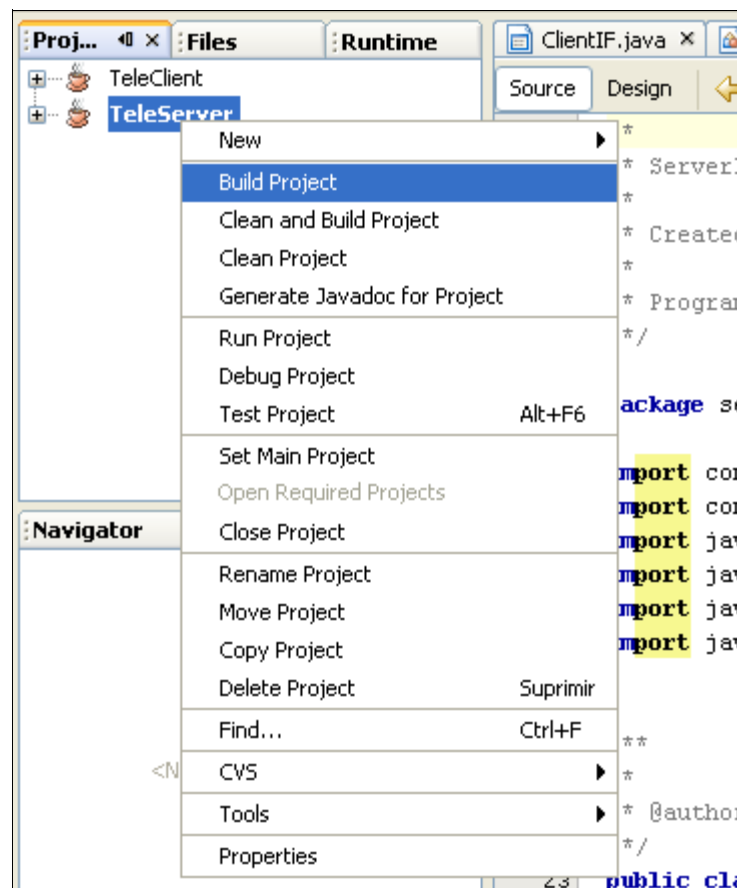


Figura 4.4: Menú contextual de un proyecto en Netbeans

4.4. Funcionamiento parte cliente

Una vez instalados los programas y todas las aplicaciones y certificados necesarios para el correcto funcionamiento de los mismos, es hora de por fin arrancar los programas y comenzar a manejarlos.

Para iniciar el programa cliente, que será utilizado por parte de un asistente o bien directamente por el paciente, simplemente tiene que ejecutar el archivo “TeleClient.jar” que

encontrará en el directorio donde instaló la aplicación. Si no arranca directamente al clicar sobre él, simplemente teclee la instrucción:

```
>> java -jar TeleClient.jar
```

Si se encuentra en un sistema Windows también puede arrancar el programa desde el acceso directo en el Menú Inicio (Start). Una vez arrancado, podemos observar ya el aspecto de nuestro programa (ver figura 4.5). Todas las funciones del programa son bastante intuitivas, y la mayoría de ellas ofrecen una pequeña ayuda si se coloca el cursor sobre ellas.

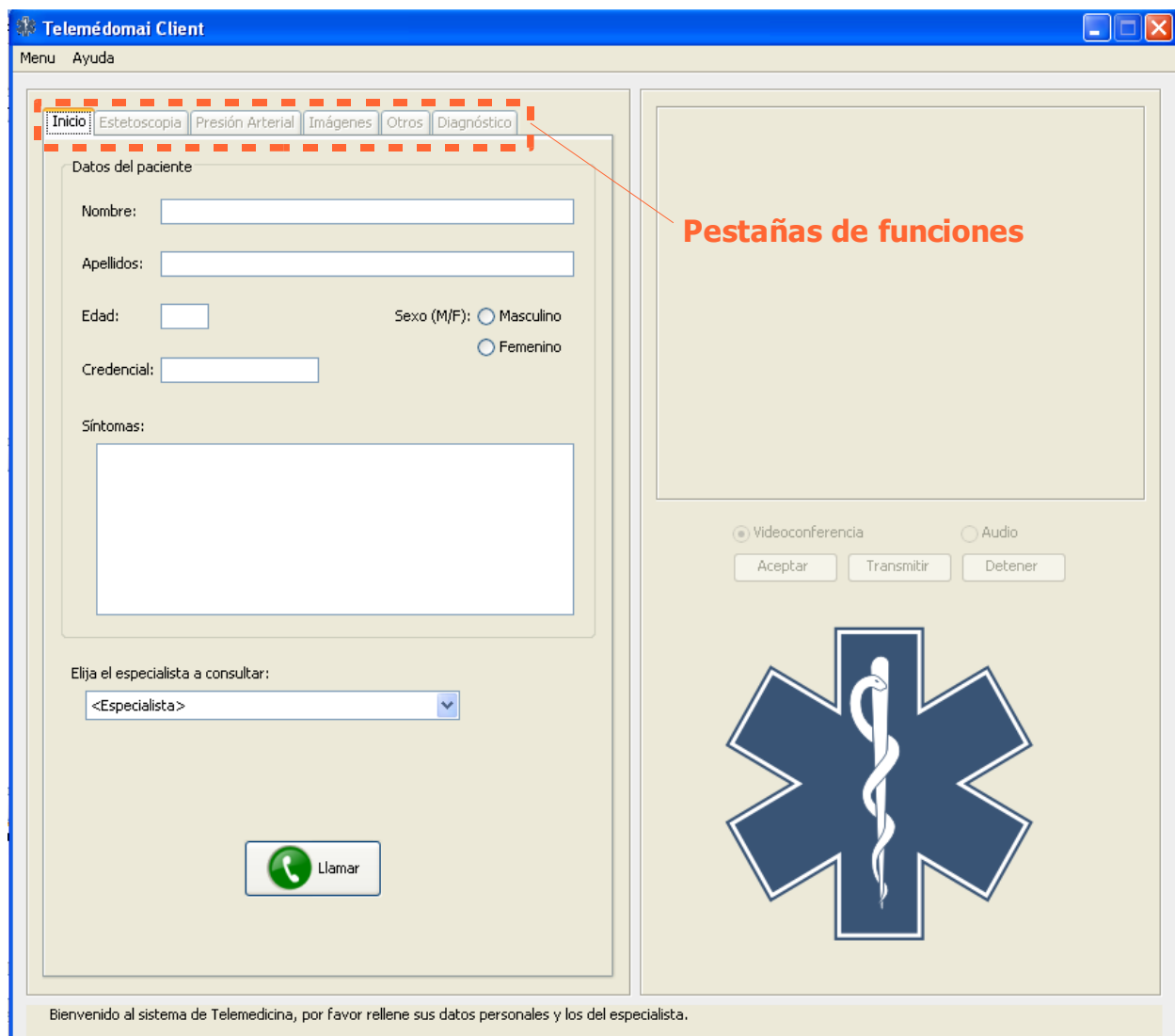


Figura 4.5: Aspecto del programa cliente al iniciar

En la interfaz del programa podemos observar cuatro partes claramente diferenciadas:

- **Barra de menús:** En ella encontramos el menú principal (para una nueva consulta, salir, etc.), y el menú de ayuda.

- **Barra de estado:** Es la barra que encontramos en la parte inferior del programa, desde donde el programa nos dará información acerca del manejo y resultados obtenidos del programa, así como los mensajes de error que puedan ocurrir..
- **Panel funcional:** Es la parte más importante de la interfaz. Su componente principal es panel con pestañas, donde encontramos todas las funcionalidades del mismo que relatamos en la sección 1.4. La primera de estas pestañas, la que el programa muestra al inicio y siempre que empecemos una nueva consulta, es la de introducción de los datos del paciente. El resto de pestañas se activarán cuando realicemos una llamada correcta a un especialista.
- **Panel de Videoconferencia:** Este último panel, que podemos encontrar a la derecha del panel funcional y prácticamente ocupa la otra mitad de la interfaz, es donde se mostrará tanto nuestro vídeo como la videoconferencia proveniente desde el servidor.

4.4.1. Envío de datos del paciente

Para poder iniciar una consulta, es necesario rellenar correctamente todos los campos de la pestaña de inicio con los datos acerca del paciente, con un relato breve de los síntomas que sufre, y finalmente seleccionar uno de los especialistas disponibles para ser consultados.

Cabe añadir que el propio programa se encarga de comprobar que todos los datos estén en los formatos adecuados (por ejemplo, la edad en años, la credencial dentro de un valor permitido, ...) y que se hayan rellenado todos los campos. Si no es así, el programa indicará nuevamente a través de la barra de estado cuál ha sido el problema causante del error.

Esta lista de especialistas debe ser gestionada desde la autoridad encargada del sistema de telemedicina. El programa está preparado para leer un fichero de configuración llamado “ip.txt” desde el directorio “CONF” dentro del archivo .JAR del programa. En este archivo están almacenados los nombres de los especialistas, separados por el carácter “:” de la dirección IP del ordenador donde está instalado el programa servidor. Se puede realizar también una pequeña modificación del código del programa, como vimos en la sección 4.3, para que en vez de leer el fichero local, pueda encontrar el mismo fichero de configuración desde cualquier página web, por ejemplo <http://www.jorgemarti.com/projects/telemedomai/ip.txt>.

Finalmente, una vez todos los datos esté correctamente introducidos, sólo se necesita pulsar el botón de llamada y esperar la respuesta del servidor, que nos aparecerá en la barra de estado.

4.4.2. Inicio de conferencia vídeo y/o audio

Una vez la llamada ha sido aceptada y recibimos respuesta positiva desde el servidor, se activarán las pestañas del resto de funciones, y el botón para iniciar la videoconferencia, por tanto el programa cliente ya estará completamente funcional.

Como indicamos anteriormente, si bien la opción idónea es la comunicación visual utilizando videoconferencia, en el programa se nos ofrece la posibilidad de escoger conferencia con o sin vídeo, en caso de que por problemas de hardware o conexiones lentas no podamos hacer uso del vídeo. Una vez escogida la opción deseada, ya podemos enviar una petición de conferencia al servidor, comenzamos a visualizar nuestro vídeo (si es el caso), se inicia el envío de datos y nos quedamos a la espera de su respuesta (ver Figura 4.6).

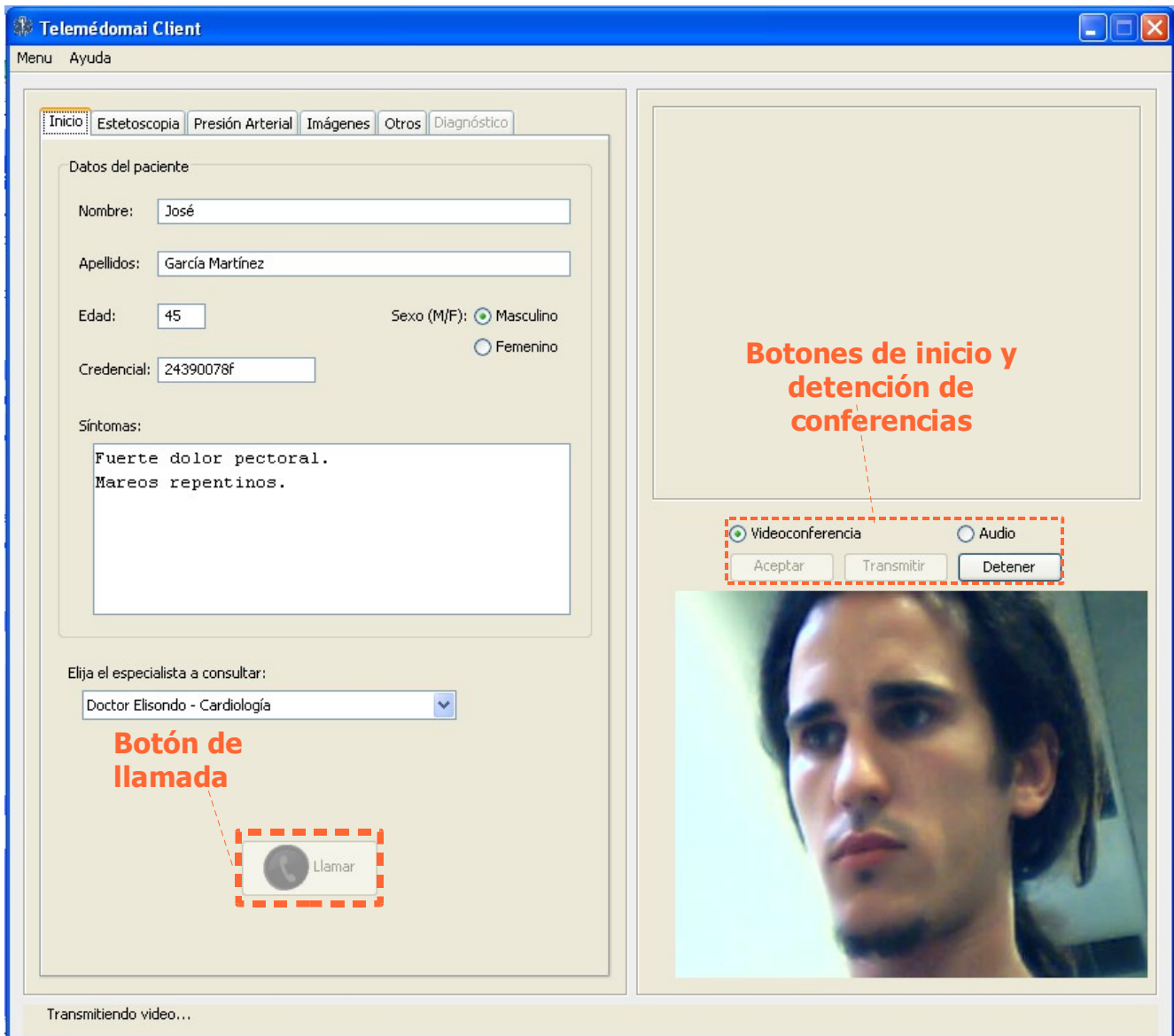


Figura 4.6: Iniciando videoconferencia.

4.4.3. Estetoscopia asistida

La primera de las funcionalidades que encontramos en el programa cliente es la de realización asistida de estetoscopias y su posterior envío. Pulsando en la pestaña correspondiente, nos aparece la pantalla de elección del tipo de estetoscopia a realizar. El programa nos deja elegir

entre los cuatro tipos de auscultaciones que explicamos en el apartado 2.2, que son la auscultación cardíaca, la pulmonar, la abdominal y otros tipos de auscultaciones. Simplemente pinchando en el botón correspondiente cambiamos de una a otra, y un gráfico animado nos ayuda a conocer los puntos exactos en donde se debe colocar el estetoscopio. Si clicamos con el botón derecho en la auscultación pulmonar, podemos cambiar entre parte anterior y posterior del cuerpo, para auscultar también puntos de la espalda.

Clicando en estos puntos en la pantalla, que señalan la parte del cuerpo que queremos auscultar, quedan seleccionados para enviarse junto con los comentarios que queramos añadir a la auscultación realizada, y serán recibidos por el especialista para su posterior análisis.

Para comenzar con la grabación, simplemente hemos de darle al botón correspondiente en la parte inferior del panel. Una vez realizada, tenemos la opción de escuchar la grabación realizada, para observar si ha habido algún error, o si todo está perfecto enviarla ya al especialista, dándole al botón de enviar (ver Figura 4.7).

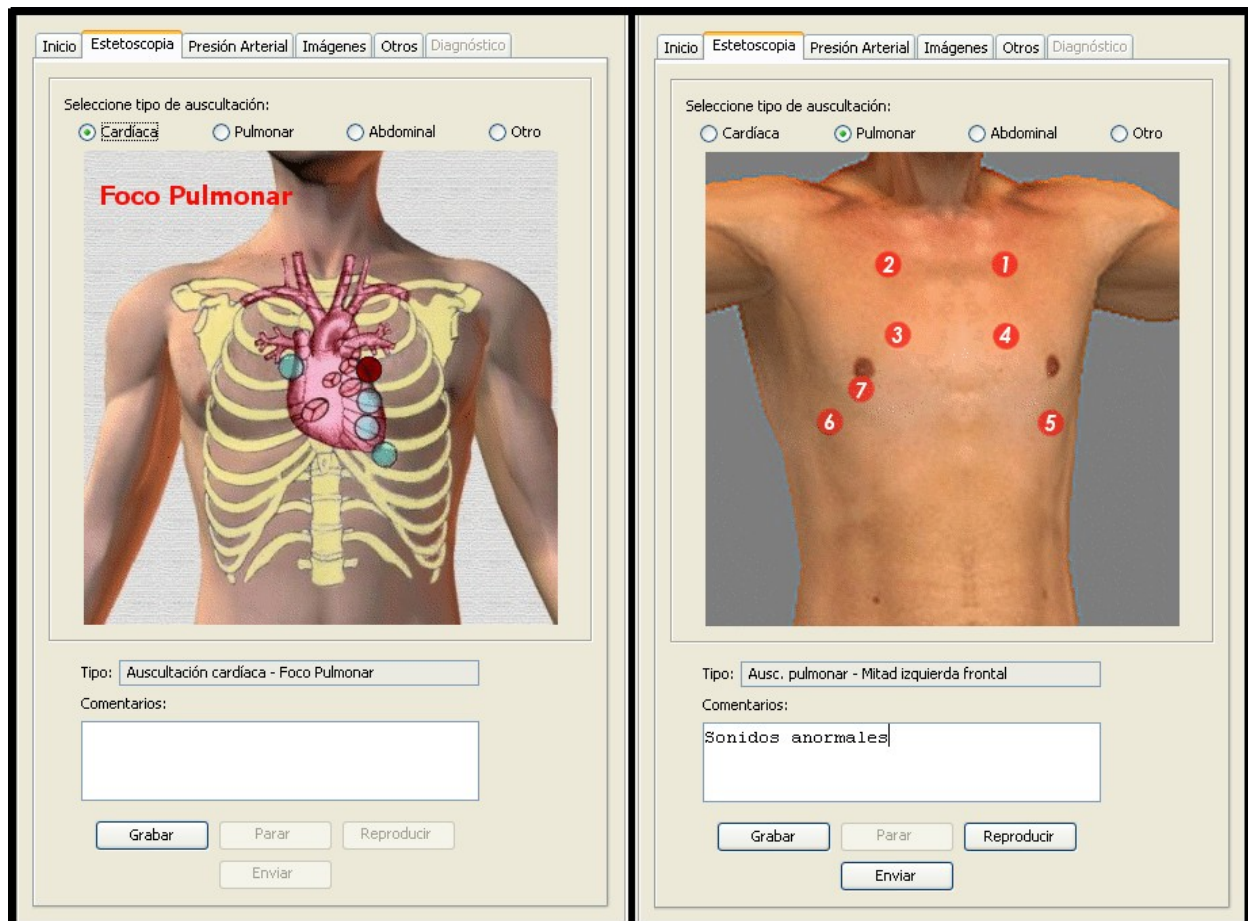


Figura 4.7: Captura de realización dos tipos de auscultaciones guiadas

Paso a paso se van realizando las auscultaciones en los puntos indicados, y se van enviando al programa servidor, junto con sus pertinentes comentarios si es necesario. El especialista tendrá la opción de escuchar cada grabación sin problemas las veces que quiera, pues todas ellas quedarán almacenadas.

4.4.4. Toma de la presión arterial

Al cambiar a la pestaña de toma de presión arterial, encontramos nuevamente un gráfico explicativo con los pasos que hemos de dar para realizar esta medida. Las imágenes reflejan dónde debemos situar los utensilios de medida y qué debemos realizar en cada paso de la técnica de toma de la presión arterial.

Para cambiar de un paso al siguiente simplemente tenemos que clicar en la imagen, tras lo que nos aparecerá la siguiente imagen con su texto explicativo en el campo inferior a ésta (Figura 4.8).



Inicio Estetoscopia **Presión Arterial** Imágenes Otros Diagnóstico



Paso 2:
Coloque el manguito alrededor del brazo, entre el hombro y el codo.

Presión arterial sistólica: mmHg

Presión arterial diastólica: mmHg

 Enviar

Figura 4.8: Pestaña de toma de presión arterial

Una vez seguidas las explicaciones y realizados todos los pasos de la medida, se deben repetir tras unos minutos para comparar los valores y estar más seguros del mismo. Tras esto, sólo queda introducir los datos de presión arterial sistólica y diastólica en los campos adecuados, y mandarlos hacia el especialista.

4.4.5. Captura y envío de imágenes

Nuestro programa dispone de una función de transferencia de imágenes de alta resolución, que como resaltábamos en el punto 2.3 son de una importancia vital en varias ramas de la medicina.

Las imágenes a enviar pueden ser de dos orígenes distintos: o bien tomadas por cualquier dispositivo capaz de almacenar una imagen digital en el ordenador, imagen que seleccionaremos desde una ventana común de selección de archivos, o también podemos hacer uso de la propia cámara web disponible para capturar una imagen. Este hecho es importante (aparte de en entornos con baja disponibilidad de recursos) porque, pese a que evidentemente la resolución de este tipo de cámaras no es muy alta, puede que la región a estudio no necesite de una resolución excesiva. Incluso existe la posibilidad de adaptar a una de estas cámaras una lente o juego de lentes que sirva de ampliación de una pequeña zona, por lo que en este caso la resolución de esta cámara sería aceptable.

En la figura 4.9 tenemos un ejemplo de captura utilizando una cámara web. Como vemos, sea una imagen capturada o bien elijamos una imagen de alta resolución almacenada en el ordenador, el programa nos muestra una pequeña vista previa de la imagen que vamos a enviar, además de animarnos a adjuntarle un pequeño texto explicativo que será enviado conjuntamente hacia el servidor.

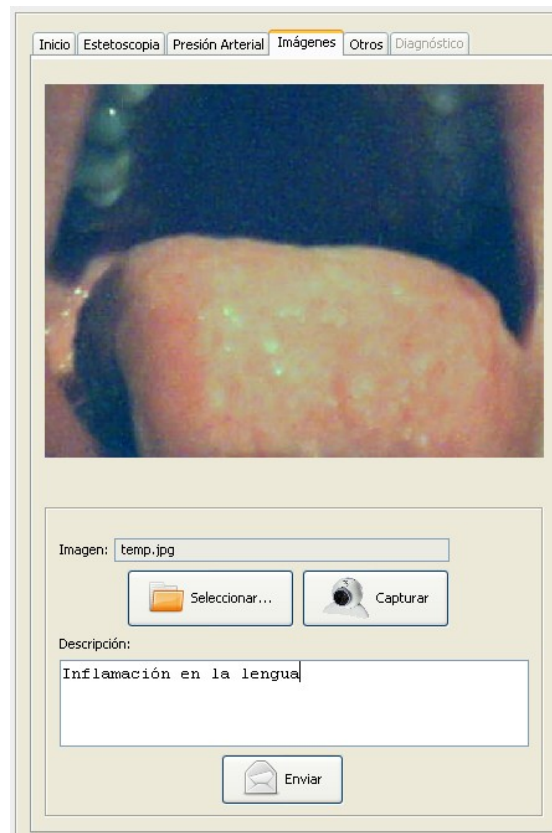


Figura 4.9: Imagen capturada, lista para ser transmitida.

Cabe añadir también que no existe límite en el número de imágenes enviadas, pues todas ellas son almacenadas separadamente por el programa servidor, y pueden ser consultadas tanto en la sesión de consulta como posteriormente.

4.4.6. Envío de cualquier otro tipo de medición

Para completar la funcionalidad del programa y dotarlo de flexibilidad, se le introdujo la opción de poder enviar cualquier tipo de medición que el especialista crea oportuno, como siempre también dependiendo del equipo disponible del lado del paciente.

Cambiando a la pestaña llamada “Otros”, observamos que tenemos unos campos de información. El primero es una lista desplegable con una lista de posibles mediciones, que pueden servir de ejemplo de las posibilidades de esta opción (Figura 4.10).

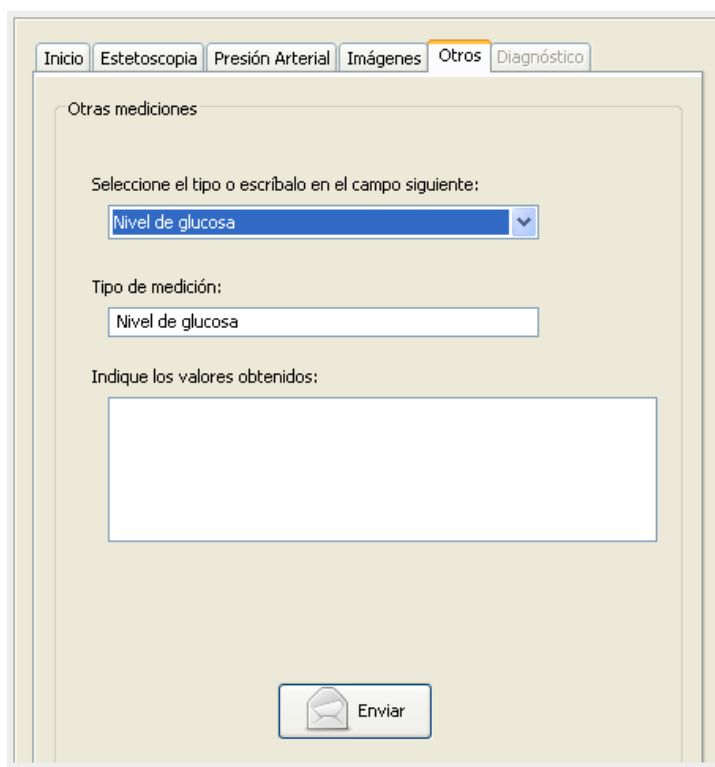


Figura 4.10: Pestaña de otras medidas

Si, en cambio, la medición realizada que se quiere enviar no está en la lista desplegable, simplemente se puede escribir en el campo destinado a tal efecto. Finalmente, en el área de texto siguiente se pueden anotar tanto los niveles medidos, como cualquier comentario que quiera hacerse acerca de la realización de tal medida.

Nuevamente, para trasladar estos datos simplemente hemos de rellenar los campos adecuadamente y pulsar el botón de enviar. Podemos mandar cuantos tipos queramos de mediciones, incluso la misma diversas veces. El programa servidor almacenará todas por separado y le permitirá al especialista analizarlas en cualquier momento.

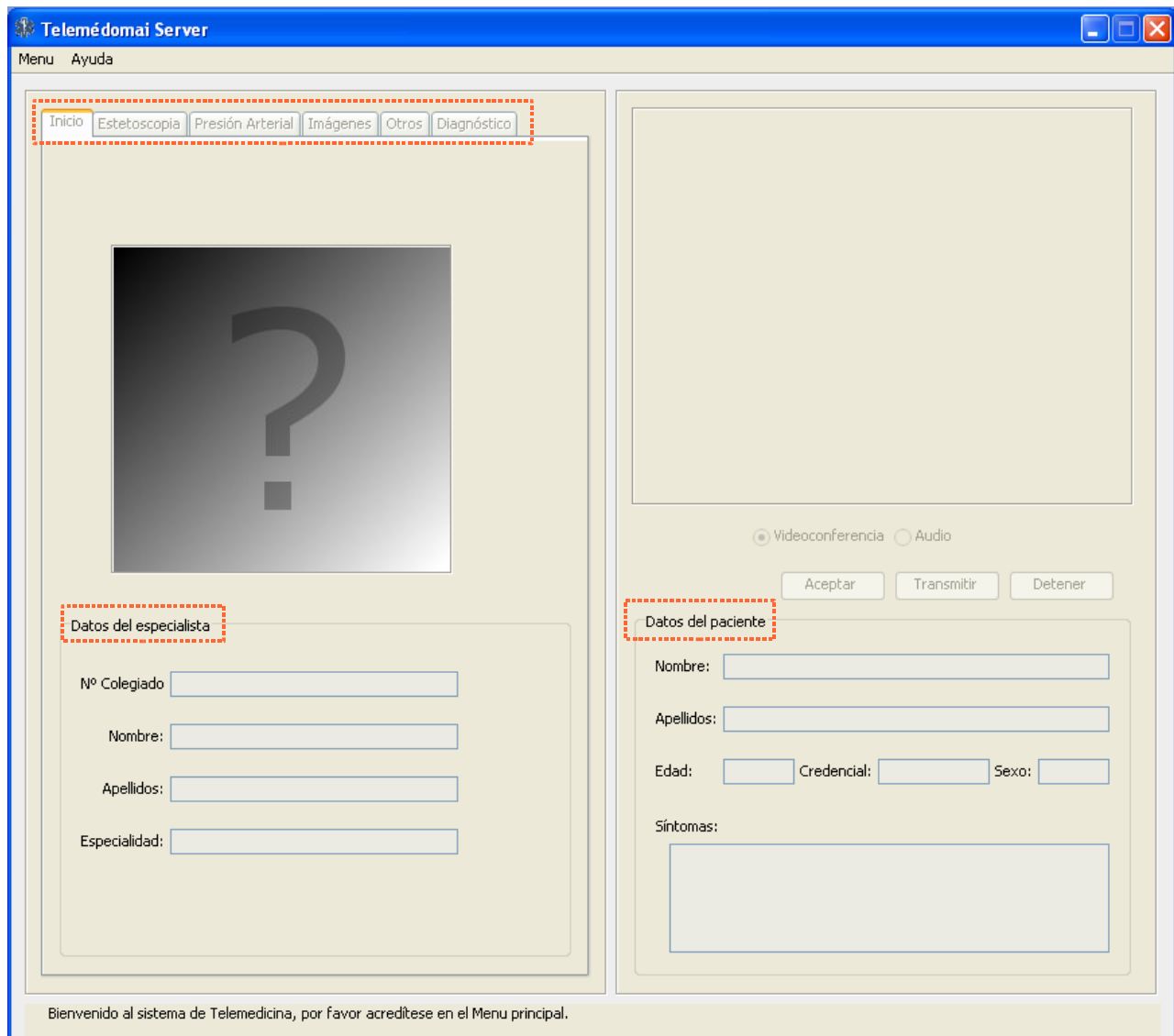


Figura 4.12: Aspecto inicial del programa servidor. Hemos resaltado tres de sus partes más importantes.

4.4.7. Recepción de diagnóstico y recomendaciones

Tras la consulta realizada, incluyendo todos los recursos anteriores de paso de imágenes, sonidos, niveles, etc. y mediante la conversación por videoconferencia el especialista probablemente disponga ya de un diagnóstico acerca de lo que, tras el estudio concienzudo de todas las pruebas, piensa que puede ser el problema del paciente. Es por eso que el especialista deberá rellenar un informe final completo y conciso acerca del diagnóstico que él considera es el adecuado para ese caso, junto con las recomendaciones que aconseja al paciente para su enfermedad.

En la pestaña de diagnóstico, una vez recibido éste por parte del especialista podremos consultar en pantalla estos datos, junto con la información personal del doctor (nombre y número de colegiado). También disponemos de la opción de guardar estos datos en un documento de tipo PDF, preparado por el programa servidor, para conservar permanentemente esta información, y consultarla en otra ocasión o imprimirla (ver Figura 4.11).

Inicio Estetoscopia Presión Arterial Imágenes Otros Diagnóstico

Diagnóstico para el paciente:

Amigdalitis aguda.

Recomendaciones:

- * Hacer enjuagues bucales con producto a tal efecto.
- * Tomar antiinflamatorios 3 veces al día, 500mg por dosis.
- * No fumar.
- * Reducir exposición al frío.

Doctor:

Doctor Jorge Martí Coronil

Nº Colegiado:

123456-X

Guardar

Figura 4.11: Recepción de diagnóstico y recomendaciones

4.4.8. Iniciar nueva consulta

Una vez terminada la sesión de consulta del paciente, el programa está preparado para realizar inmediatamente una nueva consulta, con otro paciente e incluso consultando a cualquiera de los especialistas incluidos en la lista cargada en el inicio.

Para empezar una nueva sesión simplemente necesitamos seleccionar la opción “Nuevo Paciente” en el menú principal, y comenzar de nuevo todo el proceso explicado anteriormente. El programa servidor también está en este momento preparado para empezar una nueva sesión, con todos los datos nuevos del paciente actual.

4.5. Funcionamiento parte servidor

El programa servidor será utilizado en este caso por un especialista situado en un hospital, o en una clínica, en un ordenador cuya IP será conocida por los usuarios del sistema de telemedicina. Para iniciarlo actuamos del mismo modo que en el programa cliente: simplemente tiene que ejecutar el archivo “TeleServer.jar” que encontrará en el directorio donde instaló la aplicación. Si no arranca directamente al clicar sobre él, simplemente teclee la instrucción:

```
>> java -jar TeleServer.jar
```

Si se encuentra en un sistema Windows también puede arrancar el programa desde el acceso directo en el Menú Inicio (Start). Una vez arrancado el programa servidor, podemos observar ya el aspecto del mismo y comprobar que la interfaz es muy similar a la del programa cliente (ver figura 4.12). De nuevo podemos obtener ayuda de las funciones si colocamos el ratón un momento sobre ellas.

Podemos observar nuevamente cuatro partes claramente diferenciadas también en la interfaz del programa servidor:

- **Barra de menús:** En ella encontramos el menú principal (con funciones de log in y out, salir, etc.), y el menú de ayuda.
- **Barra de estado:** Como ya indicamos para el cliente, desde esta barra el programa nos dará información acerca del manejo y resultados obtenidos del programa, así como los mensajes de error que puedan ocurrir..
- **Panel funcional:** En el panel de pestañas encontramos todas las funcionalidades del programa. Muchas de estas pestañas están desactivadas en el inicio, pues son activadas cuando llega un tipo de datos relacionado con ellas (por ejemplo, la llegada de una imagen en la pestaña de imágenes).
- **Panel de Videoconferencia y datos:** Este panel consta de dos partes: la parte de arriba nos mostrará la imagen de videoconferencia proveniente del paciente, y en la parte de abajo tenemos una relación de los datos personales del paciente que ha realizado la llamada de consulta

4.5.1. Log in del especialista

Para poder trabajar con el programa servidor es necesario que el usuario, normalmente un médico especialista, se acredite mediante la opción del Menú principal de Log in. (Muchas de las funciones de ambos programas pueden realizarse más rápidamente mediante “atajos” del teclado. El programa nos da una pista de cuáles están disponibles tras pulsar la tecla ALT. Por ejemplo para realizar el Log in habría que pulsar consecutivamente las teclas ALT+M+L).

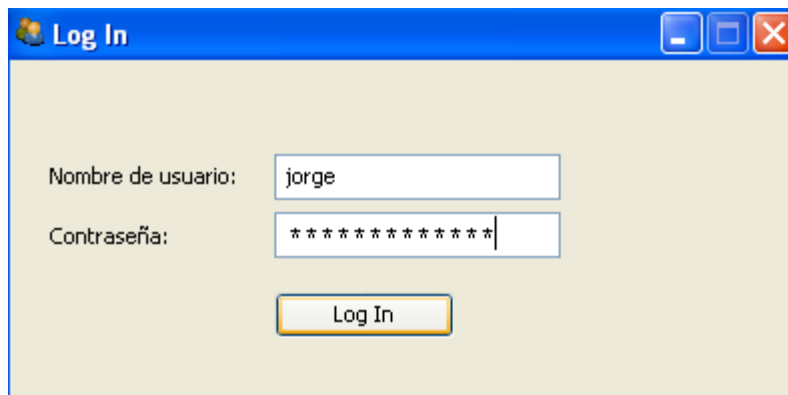


Figura 4.13: Ventana emergente de log in

Tras seleccionar esta opción, aparecerá sobre nuestro programa la ventana emergente de log in. En ella debemos introducir correctamente los datos como nombre de usuario para el sistema y contraseña.

Estos datos, que deberán ser proporcionados por la entidad que gestione todo el sistema, deben ser exactamente los mismos que los utilizados en los certificados digitales, pues el programa a la hora de comprobarlos buscará el keystore relacionado con el nombre de usuario y posteriormente comprobará que la contraseña sea la adecuada.

En caso de usuario inexistente, error en la instalación de los certificados o error de contraseña, el programa nos lo indicará a través de la barra de estado, y nos invitará a revisar las configuraciones y reintentar la introducción de nombre de usuario y contraseña.

Si el log in resulta exitoso, el programa nos mostrará en la parte izquierda toda la información concerniente al especialista: nombre completo, número de colegiado y especialidad, y también nos mostrará una fotografía del mismo (ver Figura 4.14). Todos estos datos son leídos desde un archivo de configuración llamado con el nombre de usuario del especialista y la extensión TXT, por ejemplo “jorge.txt”, que se encuentra en la carpeta CONF del programa.

La imagen es leída también desde un archivo con extensión JPG, que se encuentra en la carpeta IMAGENES. Siguiendo el ejemplo de la sección correspondiente (sección 4.4.1), se podrían hacer unas modificaciones para que el programa lea estos datos desde un servidor web centralizado, que fuese gestionado por la autoridad del sistema de telemedicina.

Tras mostrar los datos del especialista, el servidor es arrancado y el sistema se encuentra en este momento en disposición de atender a cualquier paciente que realice una llamada. El puerto en que el servidor escucha fue elegido aleatoriamente entre los puertos no reservados, los disponibles para cualquier aplicación, y en este caso es el 19820. Este número no es de importancia, siempre que sea bien conocido por todos los usuarios del programa cliente.

Figura 4.14: Tras el log in correcto, el programa muestra la información del especialista.

4.5.2. Inicio de conferencia de vídeo y/o audio

Cuando recibimos una llamada por parte de un cliente, el programa la acepta automáticamente, y nos muestra los datos del paciente que solicitó la consulta: nombre completo, edad, sexo, número de credencial, y lo más importante, un pequeño resumen de los síntomas que sufre y por los cuales quiere realizar una consulta.

Tras la recepción de los datos, se activan algunas opciones del programa como las pestañas de estetoscopia y toma de la presión arterial (para que el especialista pueda guiar al paciente en la toma de estos parámetros). La pestaña de diagnóstico queda activada y desde este momento, se podrá enviar el mismo cuando el especialista desee.

También se activará la opción de iniciar una videoconferencia con el paciente. Tenemos nuevamente la opción de elegir entre conferencia con o sin vídeo, con sólo elegir la opción deseada. Es incluso posible que uno de los dos envíe vídeo y el otro no, si por ejemplo el doctor no dispone de una cámara o no lo ve conveniente.

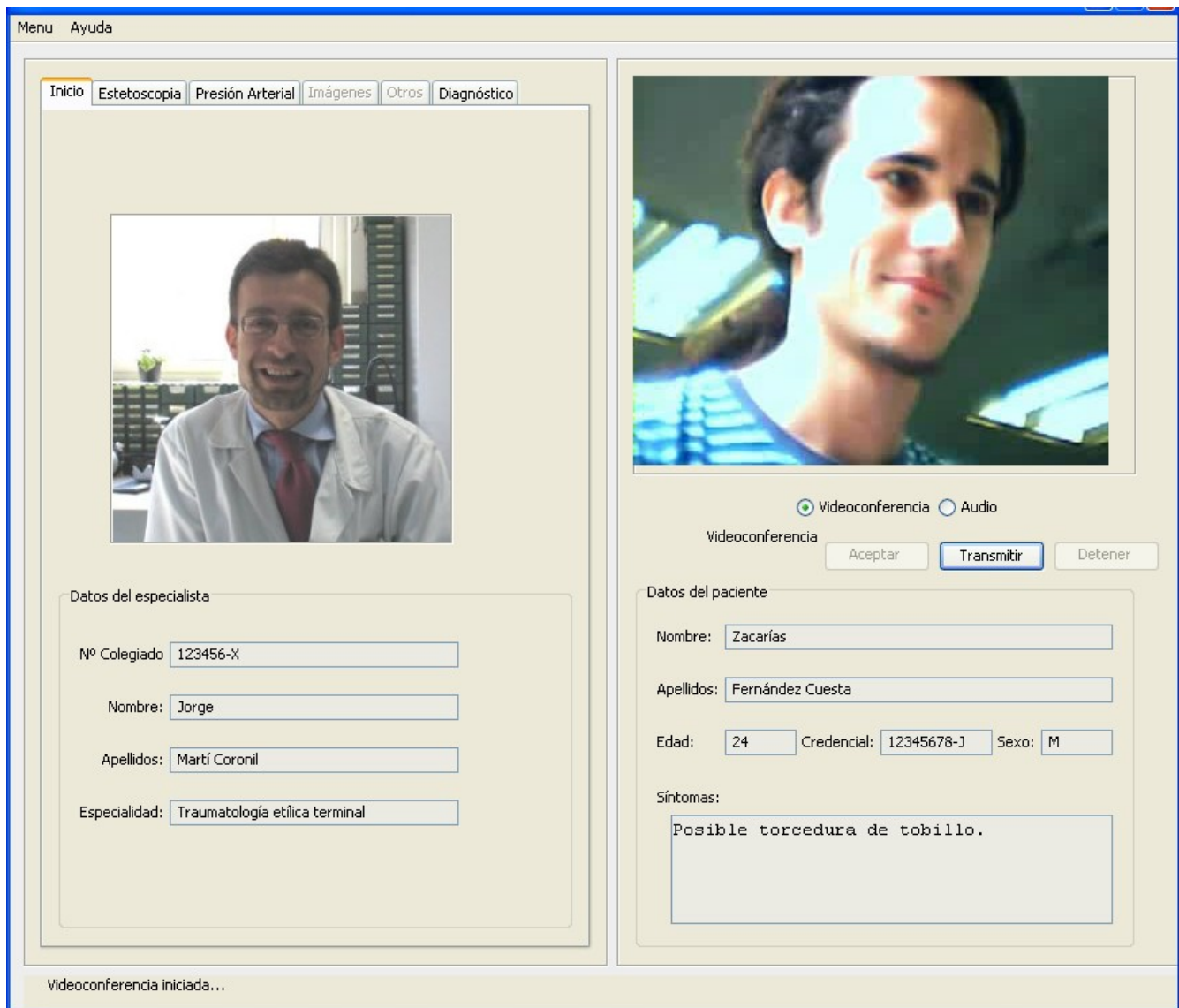


Figura 4.15: Recepción de datos y videoconferencia desde paciente. Preparado también para transmitir.

Para poder visualizar la imagen del otro participante, o bien escuchar su audio, es necesario que previamente éste acepte la petición de conferencia (Figura 4.15). Esto es tanto para el caso del programa servidor como del programa cliente. El programa nos informará de la llegada de una petición de conferencia desde el otro extremo mediante la barra de estado, y el botón de aceptar conferencia será también activado, preparado para que el usuario lo presione cuando desee.

4.5.3. Recepción de sonidos corporales

Como podemos observar en la figura 4.15, la pestaña de estetoscopia se encuentra en estado activo tras la llegada de una llamada por parte de un cliente. El motivo de este comportamiento es para que el especialista pueda cambiarse a esta pestaña y desde allí poder guiar mejor al paciente o al asistente que le ayude, pues dispondrá de las mismas imágenes de ayuda que el programa cliente.

El especialista invitará al paciente a realizar las pruebas paso por paso, y a ir mandándole los archivos resultantes de las diferentes auscultaciones. Para cambiar la visuazación de la imagen de ayuda para cada tipo de auscultación el especialista simplemente tendrá que seleccionar el botón adecuado. Para auscultaciones pulmonares, puede cambiar la imagen de frente a espalda con el uso del botón derecho, clicando en la pantalla.

El paciente irá enviándole al especialista tantas auscultaciones como éste crea necesario, anotando en ellas cualquier comentario que crea oportuno. Estos registros de sonido llegarán al programa servidor, y serán almacenados para su análisis en cualquier momento. El especialista entonces dispone de una serie de sonidos con su descripción de localización y con un comentario opcional por parte del paciente.

Utilizando la lista desplegable de sonidos recibidos, el especialista puede recorrer todos los audios, escucharlos de nuevo, leer los comentarios e incluso guardarlos en otro sitio, mediante el uso de los botones a tal efecto (figura 4.16).

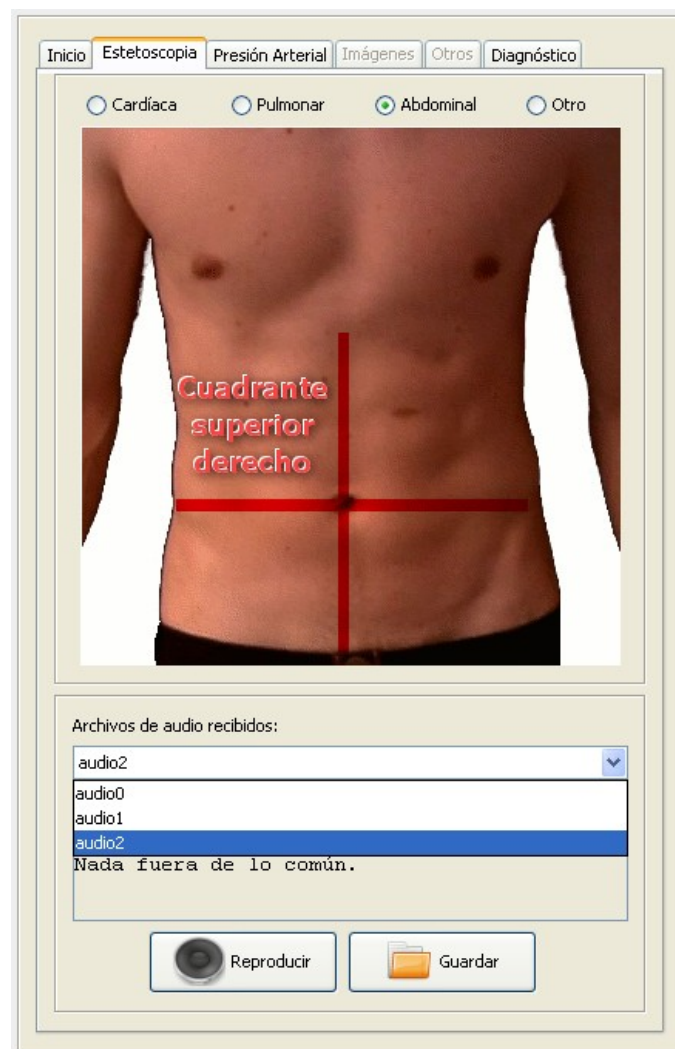


Figura 4.16: Pestaña de estetoscopias, mostrando tres auscultaciones recibidas.

4.5.4. Control de la presión arterial.

De igual modo que para las auscultaciones, la pestaña de presión arterial se encuentra activa desde el mismo momento que recibimos la llamada por parte del paciente. De este modo podemos, sirviéndonos de ayuda el mismo gráfico que el paciente estará visualizando, guiarle paso a paso por la técnica de la toma de presión arterial, que si bien no es un proceso difícil, requiere de cierta práctica, y nunca viene mal una ayuda.

La mejor combinación para tal objetivo sería que el paciente se colocase en una posición delante de la cámara web que permitiera al especialista distinguir si los pasos realizados son los correctos. Una vez realizada la medición, y los datos recibidos por el programa servidor, estas mediciones quedan también almacenadas para su posterior consulta en caso necesario. Los formatos de almacenamiento y estructura de directorios serán explicados en la sección 5.3.

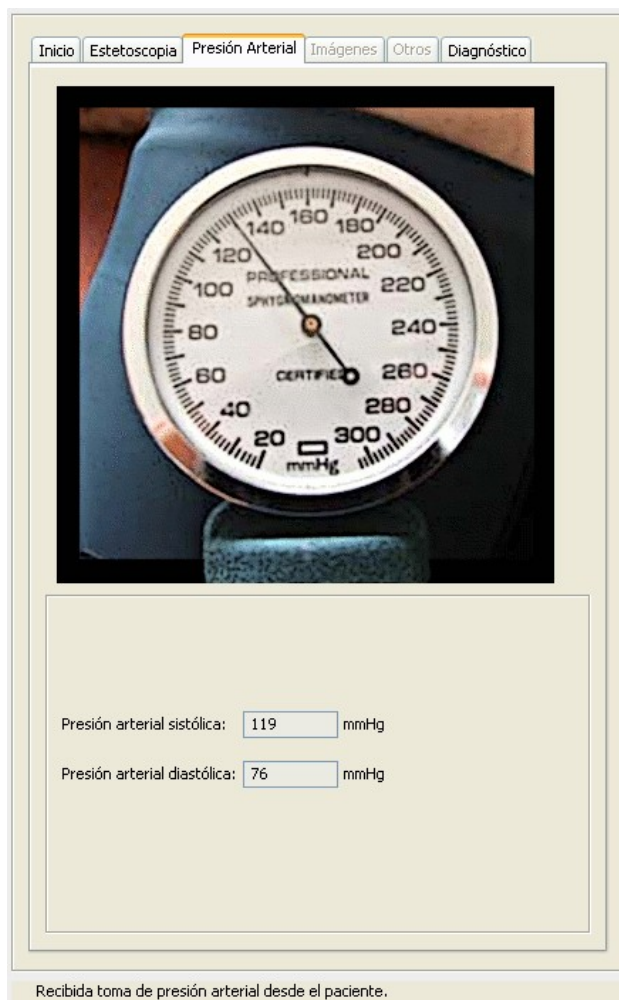


Figura 4.17: Recepción de medidas de presión arterial.

4.5.5. Análisis de imágenes

En este caso, la pestaña de imágenes sólo es activada para su uso cuando es recibida alguna imagen de alta resolución desde el programa cliente. Así se le será indicado al usuario a través de la barra de estado, y tras seleccionar la pestaña “imágenes” ya podremos utilizar todas sus funciones.

Del mismo modo que ocurría con los sonidos recibidos, cada imagen queda almacenada por el sistema, junto con su comentario asociado. Tras una serie de imágenes recibidas, cada una de un tipo diferente (una radiografía, una fotografía del tímpano, una captura de electrocardiograma, etc.), el especialista dispone de la opción de visualizarlas las veces que quiera, bien en la propia ventana del programa, o bien utilizando el programa por defecto del sistema operativo mediante la pulsación del botón “Abrir” (o el atajo del teclado ALT+B).

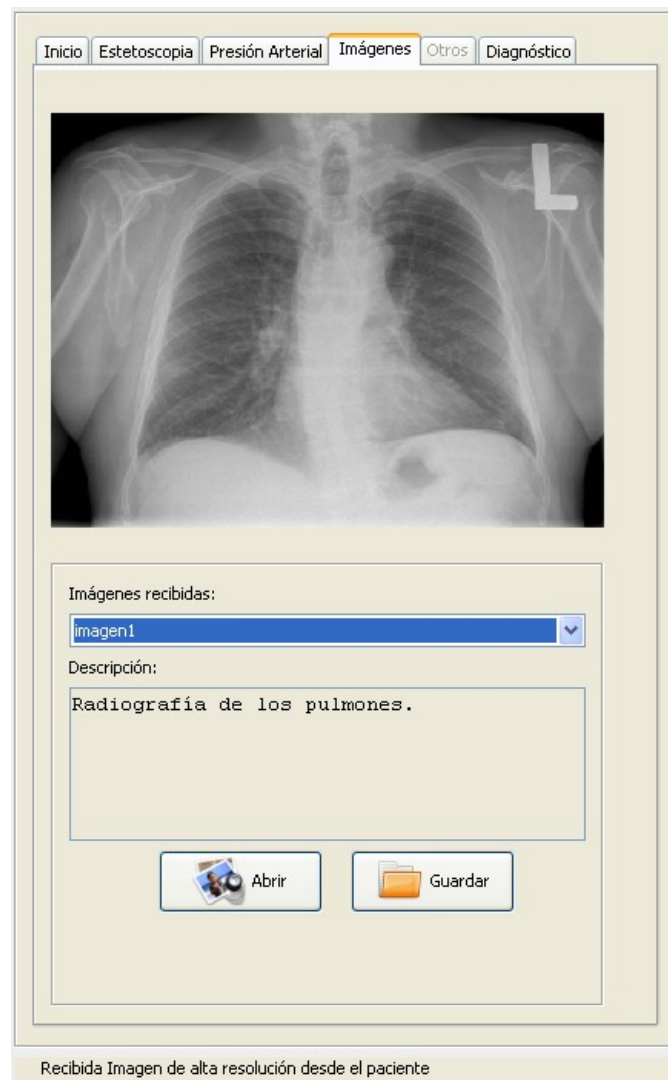


Figura 4.18: Pestaña de imágenes recibidas, mostrando un ejemplo de radiografía.

4.5.6. Otros niveles a analizar

En esta pestaña, que también se activa al recibir una primera medida, nos permite recibir cualquier tipo de medición que considere tanto el paciente como el especialista, y del mismo modo que las funciones anteriores éstos niveles recibidos son almacenados para hacer posible su consulta en cualquier momento.

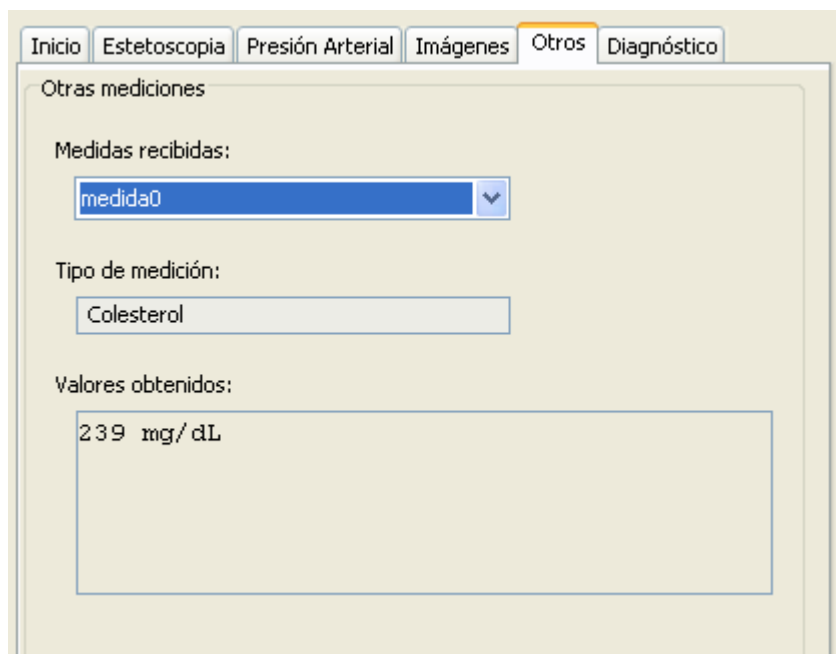


Figura 4.19: Recepción de un nivel de colesterol, con la activación de la pestaña "Otros"

4.5.7. Redacción y envío del diagnóstico

Una vez analizadas todas las pruebas, tras la consulta realizada, incluyendo todos los intercambios de datos y con el apoyo de la conversación por videoconferencia el especialista probablemente disponga ya de un diagnóstico acerca de lo que, tras el estudio concienzudo de todas las pruebas, piensa que puede ser el problema del paciente. Es por eso que el especialista deberá rellenar un informe final completo y conciso acerca del diagnóstico que él considera es el adecuado para ese caso, junto con las recomendaciones que aconseja al paciente para aliviar su enfermedad.

Para ello deberá acceder a la pestaña de diagnóstico, activa desde el mismo momento en que se recibe la información de un nuevo paciente, y rellenar adecuadamente los campos de diagnóstico y recomendaciones.

Una vez terminado, esta información se manda al paciente, junto con la generación de un archivo en formato PDF (figura 4.20) que resume toda la información de la consulta (datos del paciente, datos del especialista, diagnóstico, recomendaciones, fecha y hora). Este documento puede ser perfeccionado para incluir ya recetas médicas, aunque para ello sería necesaria la

utilización de un sistema de marcas de agua para evitar falsificaciones por parte de pacientes fraudulentos, y evitar que puedan conseguir medicamentos de forma no legal.

Informe de diagnóstico del paciente
24-05-2007, 13:15

Telemédicos
Τηλεμεδομαι

1. Datos del paciente
 Nombre: José
 Apellidos: Cuervo Especial
 Edad: 34
 Sexo: M
 N°Credencial: 12456456X

2. Sintomatología
 Respiración dificultosa

3. Diagnóstico
 Neumonía leve.

4. Recomendaciones
 * Tomar sobres de Neomotin 500mg cada 8 horas.
 * Guardar cama durante una semana.
 * Evitar sitios contaminados.

Doctor Jorge Martí Coronil
N° Colegiado: 123456-X

Figura 4.20: Ejemplo de informe generado con los datos de una consulta.

4.5.8. Otras funciones: nuevo paciente, ayuda, ...

Para completar la funcionalidad del programa encontramos algunos comandos que nos sirven para diferentes funciones. Podemos empezar una nueva consulta a otro paciente pulsando en el comando destinado a tal efecto en el menú principal (Figura 4.21-a).

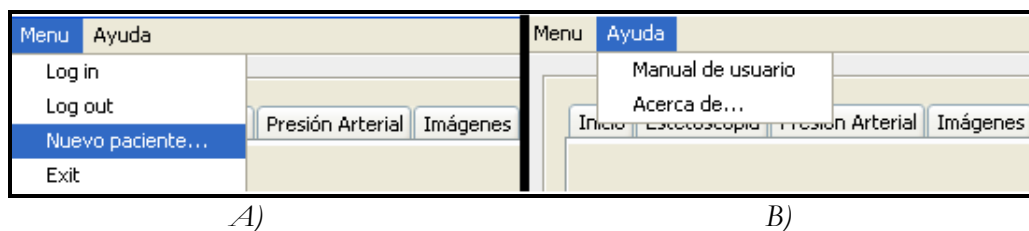


Figura 4.21: Detalle de los menús principal (a) y de ayuda (b)

Tras esto el sistema volverá al estado inicial, y quedará a la espera de la llamada de un nuevo paciente, para iniciar todo el proceso de nuevo. Si en cambio lo que queremos es que otro especialista pueda realizar consultas, elegiremos la opción de “log out”, con lo que volveremos al estado completamente inicial del programa, y el nuevo especialista deberá realizar el log in con su nombre de usuario y contraseña.

En cuanto al menú de Ayuda (figura 4.21-b), encontramos en él dos opciones (este menú es idéntico en el programa cliente). La primera de las opciones es “Manual de usuario”, y como indica su propio nombre, al clicar esta opción el programa hará una llamada al navegador web por defecto del sistema para poder navegar por las páginas de ayuda del sistema, un conjunto de páginas HTML que vienen acompañando al sistema.

Finalmente, al clicar en la opción de “Acerca de” nos aparecerá sobrescrita una ventana de diálogo mostrando el logotipo del sistema Telemédomei, así como el correo electrónico del autor del programa, para cualquier tipo de consulta. Simplemente clicando sobre esta dirección de correo, se abrirá el programa por defecto para enviar e-mails del sistema, preparado para escribir una consulta, duda o problema al autor. (ver figura 4.1).

Capítulo 5

“Destripando” el programa

Llegados a este punto ya tenemos toda la información acerca de la definición del problema que este proyecto intenta abordar, los conceptos teóricos relacionados con la telemedicina en general y con las pruebas médicas que podemos realizar utilizando nuestro programa, así como una explicación extensa del funcionamiento práctico del mismo, tanto para la versión del especialista como para el programa que utilizará el paciente.

También dedicamos un capítulo a explicar brevemente las tecnologías que hacían posible el funcionamiento del mismo, por lo tanto, sólo nos queda para dar por completamente definido nuestro proyecto adentrarnos en su estructura interna, en cómo hemos llevado a la práctica la teoría de programación y comunicación de aplicaciones de modo seguro a través de la red.

Como hemos visto en el capítulo anterior, el programa funciona perfectamente, comunicando paciente con servidor, intercambiando información de un lado a otro de forma transparente para el usuario. Pero, ¿cómo se ha logrado todo esto? ¿Qué mecanismos de control se han utilizado? ¿Cómo se han representado y almacenado los datos internamente? Estas dudas son las que intentaremos resolver en este capítulo.

5.1. Protocolo interno de comunicaciones

Considerando la variedad de formatos y cantidad de tipos diferentes de datos que nuestro programas debían intercambiar, consideramos que debíamos establecer un protocolo de comunicaciones hecho a medida, que nos permitiera saber exactamente qué se estaba transmitiendo o recibiendo en cada momento.

Evidentemente, debía de ser un protocolo totalmente nuevo, pues ningún protocolo existente se podía ceñir a nuestros resultados deseados. Este debía ser sencillo, para no complicar ni retrasar la conexión, pero eficiente pues cualquier error en la interpretación de los datos recibidos no podría ser corregido.

Finalmente, definimos un protocolo básico con sólo 9 tipos de mensajes posibles, que sería común tanto para el cliente como para el servidor. Estos son los mensajes que intercambia nuestro sistema:

- '0': Envío de un objeto de datos de un nuevo paciente.
- '1': Envío de un objeto imagen (archivo de imagen + comentario sobre la imagen).
- '2': Envío de un objeto de audio (archivo de audio + parte auscultada + comentario sobre el audio).
- '3': Envío de medida de la presión arterial.
- '4': Envío de otro tipo de medida, con sus campos de tipo y resultado de la medida.
- '5': Petición de inicio de videoconferencia.
- '6': Petición de inicio de audioconferencia.
- '7': Comunicación de fin de conferencia (sea de audio o vídeo).
- '8': Envío de diagnóstico.

El número en cada tipo de mensaje es utilizado en la implementación del protocolo para reconocer cuál de los mensajes estamos recibiendo. La comunicación se realiza de la siguiente manera: primero el gestor del protocolo del programa cliente o del programa servidor escribe a través de la salida del socket un número entero correspondiente con el mensaje que quiere enviar, para posteriormente mandar un objeto del tipo adecuado según el mensaje.

Por ejemplo, para envío de un mensaje de imagen, se escribiría en el OutputStream primero el entero '1', y posteriormente el objeto imagen, que consta de un archivo de imagen y una campo de texto de descripción de la misma. Podemos observar un ejemplo de objeto imagen en el anexo A.3..

Para aclarar qué tipos de mensajes disponibles tenemos y quién envía cada tipo, ofrecemos el siguiente esquema:



Figura 5.1: Esquema de mensajes del protocolo.

5.2. Jerarquía de clases

Tras la explicación del protocolo de comunicación utilizado, pasamos a adentrarnos en la jerarquía de las todas las clases implementadas, con la explicación de las más importantes.

5.2.1. Programa cliente

Para empezar, nos centraremos en la explicación de las clases que componen el proyecto del paciente, llamado *TeleClient*. En la figura 5.2 podemos observar el árbol de clases que lo forman, según nos lo presenta NetBeans:

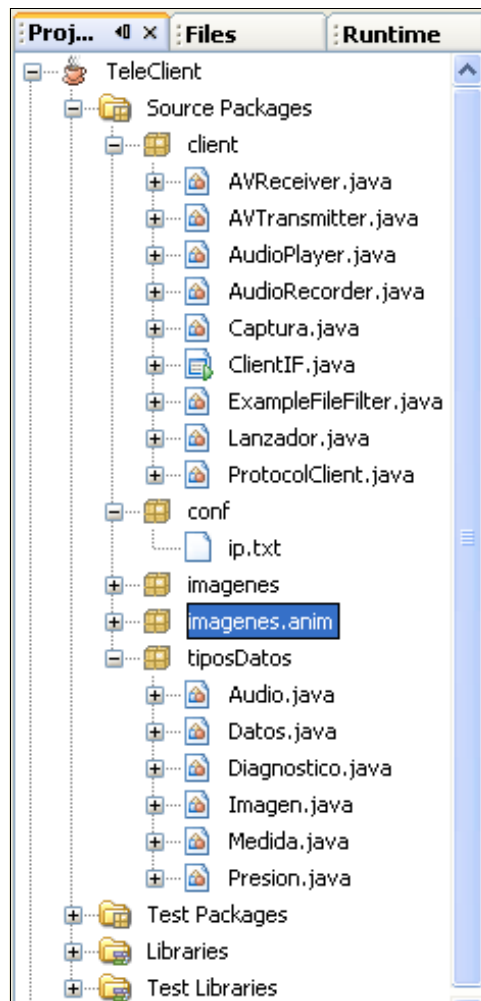


Figura 5.2: Jerarquía de clases del programa cliente.

En el gráfico podemos observar que el código y los archivos del proyecto están divididos en carpetas, que a nivel interno de Java son consideradas como paquetes o “packages”. A continuación ofrecemos la explicación de la utilidad de cada uno de estas carpetas, junto con las clases o archivos que contienen:

Carpeta “client”: Esta es la carpeta más importante de todas, pues es la que contiene prácticamente la totalidad del código de nuestro programa, incluyendo las clases del interfaz, de la captura de datos, transmisión de conferencias, etc. Para comprender mejor el funcionamiento de nuestro programa, explicaremos a grandes rasgos cuál es la misión de cada una de estas clases:

- **AVReceiver.java:** Esta clase es la encargada de recibir las transmisiones realizadas mediante el protocolo RTP. Al instanciar un objeto de esta clase e inicializarlo, el receptor se pone a escuchar en el puerto determinado (normalmente 10000 o 10002) por si recibe alguna transmisión de un stream sea tanto de vídeo como de audio. Una vez recibido, podemos añadir a nuestra interfaz los elementos multimedia recibidos (audio y vídeo), mediante su función “getPlayerComponent”, que devuelve un componente gráfico adaptado al interfaz

de una aplicación Java.

- **AVTransmitter.java:** Esta clase es la complementaria a la anterior. Dependiendo del tipo de conferencia deseada, se encarga de preparar los dispositivos de captura, crear una fuente de datos con los datos provenientes de los dispositivos, combinar los flujos de audio y vídeo en una sola fuente de datos con 2 pistas, y finalmente transmitir esta fuente mediante una sesión RTP previamente configurada. Hay que añadir que también realiza la clonación de esa fuente de datos, para poder visualizar localmente el propio vídeo del usuario, mediante la llamada por parte del interfaz a una función parecida a la “getPlayerComponent” de la clase anterior.
- **AudioPlayer.java** y **AudioRecorder.java:** Clases auxiliares simples de reproducción y grabación de sonidos utilizando cualquier entrada de audio del sistema, y el paquete de sonido de Java javax.sound. Tras instanciar un objeto de cada tipo, obtenemos el resultado deseado llamando a las funciones play(“archivo”) y start(“archivo”) respectivamente, para que lean o escriban desde un archivo de audio.
- **Captura.java:** Clase auxiliar utilizada para la captura de imágenes desde una cámara web. Al iniciar el capturador, se inicia la cámara web desde una ventana emergente que aparece en la interfaz. Al pulsar el botón de captura, el interfaz llama a una función de la clase Captura.java que es la que toma una imagen estática proveniente de la cámara, imagen que podemos pasar a la pestaña de imágenes para su envío al servidor.
- **ExampleFileFilter.java:** Clase sencilla que utilizamos como filtro de archivos a la hora de guardar o leer de un tipo de ficheros determinado (sólo imágenes, o sonidos, o PDF,...).
- **Lanzador.java:** Clase utilizada para abrir documentos externos al programa (como imágenes, páginas web de ayuda, envío de correos electrónicos) con las aplicaciones predeterminadas del sistema. Para ello primero se encarga de averiguar en qué sistema operativo estamos trabajando (Windows, MacOS X, GNU/Linux con Gnome o KDE), para posteriormente hacer la llamada al programa adecuado.
- **ProcotolClient.java:** Esta clase, segunda más importante del programa, es la encargada de gestionar las comunicaciones, utilizando los sockets y el protocolo definido en el punto 5.1. Un objeto de esta clase es instanciado e iniciado cuando pulsamos al botón de llamada en la pestaña de inicio. Se ejecuta de modo paralelo al interfaz en forma de thread (hilo de ejecución paralela). Entre las tareas que maneja están la creación de los sockets seguros para comunicarse con el servidor (comprobando las contraseñas con los certificados de autenticación), la llamada al mismo para iniciar una sesión de consulta, el envío de los mensajes del protocolo para cada tipo de transmisión que deseemos (inicio de videoconferencia, envío de audios,...) y el reconocimiento de los mensajes continuamente recibidos desde el servidor, con las consecuentes llamadas a las funciones necesarias de modificación del interfaz (por ejemplo, si se recibe un mensaje de diagnóstico, no sólo es la clase encargada de recibir este diagnóstico, sino también de llamar a la función del interfaz encargada de indicárselo al usuario y de rellenar los campos adecuados en la pestaña

“diagnóstico”). También es encargado de gestionar las caídas en la comunicación, llamando a la función del interfaz que resetea el programa volviéndolo al estado inicial.

- **ClientIF.java:** Esta clase, la más importante del programa, contiene la función “main” que es la que permite la ejecución del programa. Cuando arrancamos esta clase principal inicia todos los elementos de la interfaz, poniéndolos en su estado inicial. Se encarga de administrar y dar respuesta a todos los eventos ocurridos en la interfaz, como pulsación de botones, selección de pestañas, etc. Contiene además muchas funciones auxiliares como las de lectura y escritura de información en ficheros de texto, comprobación de datos escritos en los campos, copia de ficheros,... También contiene las funciones que se comunicarán con el protocolo, para que este a su vez mande los mensajes al servidor, y a las que proporcionarán de los datos adecuados (objetos).

Carpeta “conf”: Esta carpeta deben ir todos los archivos de configuración del programa que deban ser leídos por este mismo. En la actual implementación del programa sólo hacemos uso de un archivo, llamado *ip.txt*, desde donde leemos la lista de especialistas a los que podemos consultar, con sus correspondientes direcciones IP para poder realizar la llamada con éxito.

Carpeta “imagenes”: Como su propio nombre indica, esta carpeta almacena todas las imágenes utilizadas en la interfaz de nuestro programa, desde los pequeños botones y logotipos, hasta las imágenes animadas utilizadas en las auscultaciones y toma de presión arterial.

Carpeta “tiposDatos”: Para hacer el intercambio de datos lo más independiente y extensible posible, decidimos tomar la filosofía de transmitir cada tipo de datos como un objeto, y definirlo en su propia clase almacenada en esta carpeta. Cada tipo de datos está definido por sus elementos (campos de texto como descripciones, valores de medidas, etc. o archivos como imágenes o sonidos) y por sus métodos de escritura y lectura del propio objeto. Por ejemplo, para el objeto de tipo imagen tenemos definidos dos campos: nombre del archivo que almacena la imagen en sí, y la descripción. Luego aparte tenemos definido cómo transmitir este objeto, que sería primero transmitir los dos campos de texto y después transmitir la propia imagen kilobyte a kilobyte. Evidentemente, también tenemos definida la función de recepción del objeto imagen, que actúa de forma simétrica, pues las clases de la carpeta “tiposDatos” son compartidas tanto por el programa cliente por el servidor, son las mismas.

5.2.2. Programa servidor

Ahora ya podemos encargarnos de explorar el árbol de clases de la parte del servidor, que como observaremos es muy similar a la del cliente, pues comparten muchas funcionalidades y todos los tipos de datos recientemente definidos. Aquí tenemos la imagen de la jerarquía de clases y carpetas ofrecida por NetBeans (figura 5.3):

Podemos observar que, como en el caso del programa cliente, la jerarquía de clases y ficheros está dividida en carpetas. Podemos nuevamente realizar una explicación breve de las carpetas, junto con las clases o archivos que contienen:

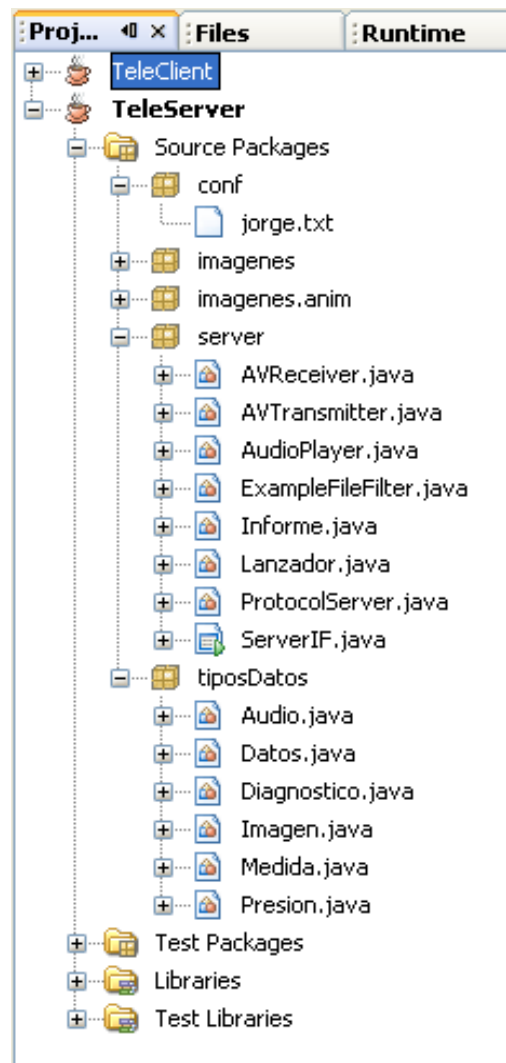


Figura 5.3: Jerarquía de clases del programa servidor.

Carpeta “server”: Esta es la equivalente a la carpeta client del programa cliente, la más importante de todas, pues es la que contiene prácticamente la totalidad del código de nuestro programa. Para comprender mejor el funcionamiento de nuestro programa, explicaremos a grandes rasgos cuál es la misión de cada una de estas clases:

- **AVReceiver.java:** Exactamente el mismo funcionamiento que en el programa cliente, pues la videoconferencia es totalmente bidireccional.
- **AVTransmitter.java:** Ídem de lo anterior.
- **AudioPlayer.java:** Esta vez no es necesario el uso de una clase de grabación, pues el programa servidor no utiliza esta funcionalidad. En cambio la clase AudioPlayer.java es la misma que la del paciente, y nos sirve para poder escuchar las auscultaciones recibidas.

- **ExampleFileFilter.java:** Al igual que en el cliente, es una clase sencilla que utilizamos como filtro de archivos a la hora de guardar o leer de un tipo de ficheros determinado (sólo imágenes, o sonidos, o PDF,...).
- **Informe.java:** Esta clase, que es una característica única del programa servidor, es la encargada de la creación de los informes del diagnóstico en formato PDF. Para ello incluyen en el documento toda la información de paciente y especialista, junto con el diagnóstico y las recomendaciones, la fecha y la hora del sistema y los logotipos del programa. El documento generado será posteriormente enviado por el interfaz.
- **Lanzador.java:** Clase idéntica en funciones a la del programa cliente (ver apartado 5.2.1).
- **ProcotolServer.java:** Esta clase, segunda más importante del programa, realiza las mismas funciones que su homónima en el programa cliente: es la encargada de gestionar las comunicaciones, utilizando los sockets y el protocolo definido en el punto 5.1, y su comunicación con la interfaz del programa.
- **ServerIF.java:** Esta clase, la más importante del programa, sólo se difiere de la ClientIF.java en unos pocos detalles, principalmente de aspecto del interfaz sus funciones y posibles eventos. Contiene la función “main” que es la que permite la ejecución del programa.

Carpeta “conf”: En la actual implementación del programa sólo utilizamos esta carpeta para leer los datos del especialista, una vez éste ha realizado un log in correcto con su nombre de usuario y contraseña.

Carpeta “imagenes”: Al igual que la del programa cliente, almacena todas las imágenes utilizadas en la interfaz de nuestro programa.

Carpeta “tiposDatos”: Contiene las mismas clases utilizadas para la representación de los objetos de tipos de datos que en el programa cliente, pues en cada clase vienen definidas, además de sus características, sus métodos para enviar y recibir objetos de este tipo.

5.3. Almacenamiento de datos

Por almacenamiento de datos entendemos a la acción del programa servidor de almacenar cada registro, cada dato recibido, cada archivo de imagen o sonido o cualquier cosa desde el programa cliente.

Cuando el servidor recibe una llamada de un cliente, automáticamente éste crea una carpeta utilizando como nombre de carpeta el número de credencial del paciente (número que es único para cada persona). Tras esto, el programa irá almacenando todos los datos recibidos de forma distinta dependiendo de su carácter. Aquí tenemos la lista de todos los datos almacenados:

Imágenes: El programa crea una carpeta llamada “imagenes” dentro de la carpeta del paciente, y en ella almacena tanto las imágenes, numerándolas a partir del 0 como imagenX.jpg, como un documento de texto asociado con la información relativa a la imagen, con el mismo nombre pero con la extensión de archivo de texto TXT.

Audios: Exactamente igual que en el caso de las imágenes, el programa también crea una carpeta para clasificar los archivos de audio recibidos, los numera del mismo modo e igualmente crea archivos de texto asociados para almacenar los comentarios.

Presión arterial: Cuando una medida de presión arterial es recibida, esta queda escritas en el directorio del usuario dentro del archivo “presion.txt”.

Otras medidas: Del mismo modo es almacenada cada medida recibida, en archivos numerados desde el 0, pues a diferencia de la presión arterial podemos recibir y consultar una cantidad indeterminada de medidas.

Informe de diagnóstico: Al mismo tiempo que se envía el documento PDF con diagnóstico al paciente, se almacena una copia en la carpeta con el nombre de “diagnostico.pdf”, con el que posteriormente podremos ver el resumen de toda la sesión de consulta médica.

Como pueden haber adivinado, todos estos archivos quedan almacenados en la computadora del especialista, y pueden ser consultados con posterioridad, o bien archivados mediante algún sistema de copias de seguridad. En la figura 5.4 podemos observar cómo quedaría la carpeta personal de un paciente tras una sesión de telemedicina con el sistema Telemédmai (recordamos que esta carpeta con datos sobre la consulta queda almacenada en la computadora del especialista).

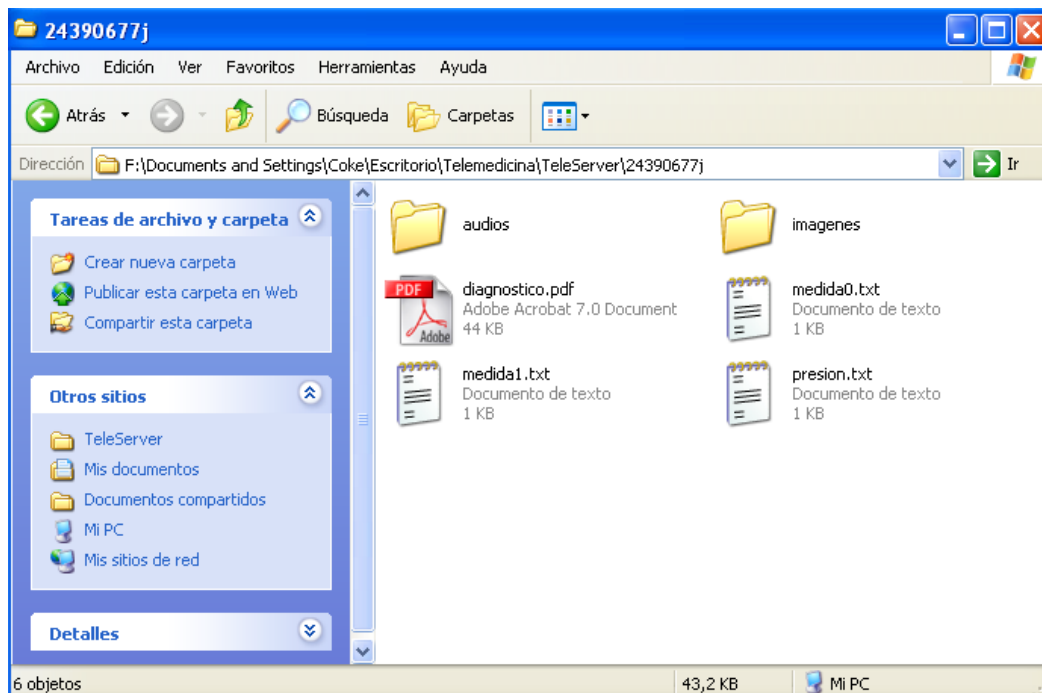


Figura 5.4: Ejemplo de carpeta con los datos de la sesión de consulta.

Capítulo 6

Conclusiones

Una vez concluida la explicación del proyecto en su totalidad, desde su motivación, hasta su justificación teórica, pasando por la explicación de la tecnología empleada para conseguir los objetivos, el funcionamiento del programa conseguido y por último la definición de la estructura interna del mismo, es hora de que evaluemos hasta qué punto hemos alcanzado los objetivos marcados en un comienzo.

Primero analizaremos qué resultados hemos obtenido, y los compararemos con los pretendidos en un principio, para posteriormente hacer una relación de los objetivos que nos gustaría poder cumplir en el futuro, como posibles extensiones del sistema. Cabe recordar que conseguir un sistema extensible era uno de los objetivos principales, y creemos que lo hemos cumplido.

6.1. Resultados

El proyecto realizado tenía como finalidad intentar ayudar a la implantación y uso de las ventajas de la Telemedicina, y tratar de contribuir a una atención sanitaria global, accesible y de calidad. Para ello se pensó en la realización de un *sistema de teleconsulta y telediagnos*, con el que el paciente pudiese obtener una consulta de calidad a pesar de encontrarse a mucha distancia del especialista y para que el especialista pueda analizar y catalogar con facilidad la información recibida desde el paciente y proporcionarle un diagnóstico adecuado.

Si consiguiéramos el apoyo de una institución, organización o empresa adecuada que nos permitiera conseguir un despliegue extenso de este sistema, el paciente podría consultar desde su

centro de atención primaria a diferentes especialistas en cualquier campo de la medicina directamente en el lugar que éstos se encuentren, recibiendo una atención inmediata, e intentando ésta fuese de la mejor calidad.

Desde un principio tuvimos bien claro que teníamos que trabajar mucho en conseguir un programa accesible y fácil de manejar, pues cuanto menor fuesen los requisitos de conocimientos para el usuario del sistema, mayor alcance (posibles usuarios) podría tener este.

Evidentemente, el uso por parte del paciente sin asistencia siempre estaba basado en la premisa de que el especialista podría guiarle gracias a un sistema de videoconferencia, por lo que este objetivo también se ve cumplido.

En cuanto a los aspectos técnicos, conseguimos un programa prácticamente independiente de la plataforma utilizada, poco exigente técnicamente y en cuestión de recursos, y con facilidad de distribución e instalación, lo que permitiría un despliegue mucho más sencillo.

Finalmente, creemos haber cumplido también con el objetivo de crear un programa flexible para su aplicación a diversas patologías, y sobretodo con facilidad para ser extendido. Para ello trabajamos en pos de la claridad del código e intentando ofrecer una explicación detallada de las funciones, protocolos, clases, etc. y de cómo realizar la modificación del código para que un posible usuario o entidad que quisiera ampliar el programa pudiese hacerlo con facilidad.

Por todos estos motivos, podemos concluir que hemos cumplido los objetivos marcados al principio de la etapa de desarrollo, pese a lo que aún no estamos completamente satisfechos pues el objetivo máximo de este programa, de todo este proyecto, es alcanzar en un futuro próximo la implantación del mismo, y cumplir con ello la función para la que fue ideado.

6.2. Posibles extensiones

Siguiendo la filosofía de “la perpetua beta” (NUNCA un programa está completamente terminado, siempre hay mejoras que realizar), queríamos dedicar un apartado a aquellas características que, bien por falta de tiempo o por no ser de prioridad extrema ni formar parte de los objetivos principales, no hemos incluido en el programa, pero nos gustaría hacerlo en un futuro próximo.

La lista evidentemente no está tampoco finalizada, pues no siempre es posible imaginar todas las opciones que podrían añadirse, y siempre es positiva la opinión de personas ajenas al desarrollo del programa, por lo que el autor agradecería de antemano las ideas que al lector le pudiesen surgir.

- Intentar conseguir la conexión directa a nuestro programa de dispositivos de captura de imágenes, por mediación del puerto USB, como puedan ser endoscopios, ecógrafos, oftalmoscopios, otoscopios,...

- Conseguir también que el programa pueda comunicarse con otros tipos de aparatos de medida, como medidores de niveles de glucosa, coagulómetros, ...
- Posible realización de electrocardiogramas y su envío hacia el programa del servidor, para su representación gráfica en tiempo real.
- Adición de un sistema de recetas médicas a los informes PDF enviados al paciente, para que éste pueda comprar los medicamentos directamente.
- Proporcionar a los informes de un sistema de marca de agua para evitar falsificaciones.
- Añadir todas las pruebas realizadas (imágenes enviadas, sonidos, etc.) a la redacción del informe, para que sea más completo.
- Conectar nuestro programa con una base de datos centralizada del sistema de telemedicina, para almacenar todos los datos de las consultas, para que puedan ser consultados posteriormente para un nuevo análisis, tanto por el especialista que realizó la consulta, como por otro doctor (petición de segunda opinión).
- Añadirle la característica de que el programa cliente ofreciera al paciente la información de qué especialistas se encuentran en este instante en activo e incluso cuáles de ellos están ocupados, para poder elegir uno disponible. Esta información sería también centralizada por el sistema.

Apéndice A

Clases importantes

En este primer apéndice del proyecto hemos querido añadir el código de aquellas clases que consideramos de gran importancia en el sistema, para que los lectores iniciados puedan observar con precisión la estructura y funcionamiento a bajo nivel del programa.

A.1. ProtocolClient.java

```
/*
 * ProtocolClient.java
 *
 * Created on 6 de marzo de 2007, 17:09
 *
 * Clase creada para gestionar las comunicaciones entre el
 * interfaz del programa y la red, encargada de mandar
 * los mensajes al servidor.
 */

package client;

import java.net.*;
import java.io.*;

import javax.net.ssl.*;
import javax.security.cert.X509Certificate;
import java.security.KeyStore;
import javax.net.*;
```

```

/**
 *
 * @author Coke
 */
public class ProtocolClient extends Thread{

    private ClientIF interfaz;
    private String server="localhost";
    private SSLSocket socketCliente;
    private OutputStream output;
    private InputStream input;
    private ObjectOutputStream salidaDatos;
    private ObjectInputStream entradaDatos;

    private boolean connected;

    /**
     * Crea una instancia de ProtocolClient
     */

    public ProtocolClient()
    {

    }

    /**
     * Crea una instancia de ProtocolClient, con el servidor
     * como parámetro.
     */

    public ProtocolClient(String servidor)
    {
        server=servidor;
    }

    /**
     * Método que conecta con el servidor
     *
     */

    public boolean startProtocol(ClientIF iface)
    {
        interfaz=iface;
        try
        {
            SSLSocketFactory factory=getSocketFactory();
            socketCliente=(SSLSocket) factory.createSocket(server, 19820);

            socketCliente.startHandshake();

            output=socketCliente.getOutputStream();
            input=socketCliente.getInputStream();

```

```

        salidaDatos=new ObjectOutputStream(output);
        entradaDatos=new ObjectInputStream(input);
        System.out.println("Conexión correcta con el servidor"
+socketCliente.getInetAddress());
        this.start();
        connected=true;
        return true;
    }
    catch (Exception e)
    {
        System.out.println(e.getMessage());
        return false;
    }
}

/**
 * Método run() necesario para la clase thread.
 * Continuamente realiza comprobaciones de si hemos
 * recibido algún mensaje desde el servidor.
 */

public void run()
{
    while (connected)
    {
        int type=typeofDataReceived(entradaDatos);
        if (type==8)
        {
            tiposDatos.Diagnostico diag=receiveDiagnostico(entradaDatos);
            interfaz.setDiagnostico(diag);
        }
        else if (type==5)
        {
            interfaz.setWebcam();
        }
        else if (type==7)
        {
            interfaz.stopConferencia();
        }
        else if (type==-1)
        {
            interfaz.resetConnection();
            connected=false;
        }
    }
}

/**
 * Método para detener la conexión y que se detenga el thread
 *
 */

```

```

public void stopProtocol()
{
    connected=false;
    try
    {
        output.close();
        input.close();
        salidaDatos.close();
        entradaDatos.close();
        socketCliente.close();
    }
    catch (Exception e)
    {
        System.out.println("Error cerrando la conexión: "+e);
    }
}

public void enviaDatos(tiposDatos.Datos d)
{
    try
    {
        // Enviamos primero el indicador de que estamos mandando
        //los DATOS DEL PACIENTE
        //('0'),para que el servidor esté preparado para la recepción
        //de un objeto detipo Datos, especificado en la clase Datos.java

        salidaDatos.writeInt(0);

        //Y ahora mandamos los datos propiamente
        salidaDatos.writeObject(d);

    }
    catch (Exception e)
    {
        System.out.println("NO SE PUDO ESCRIBIR LOS DATOS");
    }
}

public boolean enviaImagen(String file,String descr)
{
    try
    {
        // Enviamos primero el indicador de que estamos mandando
        // UN ARCHIVO DE IMAGEN ('1'),para que el servidor
        // esté preparado para la recepción de un objeto de
        // tipo imagen
        salidaDatos.writeInt(1);
        salidaDatos.flush();

        //Y ahora mandamos los datos propiamente
        salidaDatos.writeObject(new tiposDatos.Imagen(file,descr));
    }
}

```



```

    }
    catch (Exception e)
    {
        System.out.println("NO SE PUDO ESCRIBIR LA IMAGEN");
        return false;
    }
    return true;
}

public boolean enviaAudio(String file,String descr)
{
    try
    {
        // Enviamos primero el indicador de que estamos mandando
        // UN ARCHIVO DE AUDIO('2'),para que el servidor esté
        // preparado para la recepción de un objeto
        // de tipo AUDIO

        salidaDatos.writeInt(2);
        salidaDatos.flush();

        //Y ahora mandamos los datos propiamente
        salidaDatos.writeObject(new tiposDatos.Audio(file,descr));

    }
    catch (Exception e)
    {
        System.out.println("NO SE PUDO ESCRIBIR EL ARCHIVO DE SONIDO");
        return false;
    }
    return true;
}

public boolean enviaPresion(String sist,String diast)
{
    try
    {
        // objeto de tipo PRESION

        salidaDatos.writeInt(3);
        salidaDatos.flush();

        //Y ahora mandamos los datos propiamente
        salidaDatos.writeObject(new tiposDatos.Presion(sist,diast));

    }
    catch (Exception e)
    {
        System.out.println("NO SE PUDO ESCRIBIR EL LOS DATOS" +
        +"DE PRESIÓN ARTERIAL");
        return false;
    }
}

```

```

        return true;
    }

    public boolean enviaMedida(String tipo,String med)
    {
        try
        {
            // de tipo MEDIDA
            salidaDatos.writeInt(4);
            salidaDatos.flush();

            //Y ahora mandamos los datos propiamente
            salidaDatos.writeObject(new tiposDatos.Medida(tipo,med));

        }
        catch (Exception e)
        {
            System.out.println("NO SE PUDO ESCRIBIR EL LOS DATOS"+
            +" DE MEDIDA");
            return false;
        }
        return true;
    }

    public boolean enviaWebcam()
    {
        try
        {
            // Enviamos el indicador de petición de transmisión de webcam
            salidaDatos.writeInt(5);
            salidaDatos.flush();
        }
        catch (Exception e)
        {
            System.out.println("Error mandando petición de videoconferencia!");
            return false;
        }
        return true;
    }

    public boolean enviaAudioConferencia()
    {
        try
        {
            // Enviamos el indicador de petición de transmisión de audioconferencia
            salidaDatos.writeInt(6);
            salidaDatos.flush();
        }
        catch (Exception e)
        {
            System.out.println("Error mandando petición de audioconferencia!");
            return false;
        }
    }

```

```

        }
        return true;
    }
}
/**
 * Método para comunicar al servidor que vamos a detener la
 * conferencia, sea webcam o sólo audio
 */
public boolean stopConferencia()
{
    try
    {
        // Enviamos el indicador de finalización de transmisión
        salidaDatos.writeInt(7);
        salidaDatos.flush();
    }
    catch (Exception e)
    {
        System.out.println("Error mandando finalización de conferencia!");
        return false;
    }
    return true;
}

/**
 * Método para recibir un diagnóstico por parte del servidor,
 * que posteriormente
 * será pasado al interfaz.
 */

private tiposDatos.Diagnostico receiveDiagnostico(ObjectInputStream
    entrada)
{
    try
    {
        Object aux=entrada.readObject();
        System.out.println("Llega a leer el objeto de Diagnóstico");
        tiposDatos.Diagnostico diag=(tiposDatos.Diagnostico)aux;
        return diag;
    }
    catch (Exception e)
    {
        System.out.println("Error leyendo datos de usuario: "+e);
        return null;
    }
}

/**
 * Método que lee de la entrada un entero para diferenciar el tipo de
 * mensaje que nos envió el servidor.
 */
private int typeOfDataReceived(ObjectInputStream entrada)
{

```

```

        try
        {
            return entrada.readInt();
        }
        catch (EOFException eof)
        {
            return -1;
        }

        catch (Exception e)
        {
            System.out.println("Error leyendo el tipo de datos: "+e);
            return -1;
        }
    }

    /**
     * Método auxiliar para configurar la conexión
     * segura SSL
     */
    private SSLSocketFactory getSocketFactory()
    {
        SSLSocketFactory factory = null;
        try
        {
            SSLContext ctx;
            KeyManagerFactory kmf;
            KeyStore ks;
            TrustManagerFactory tmf;

            ctx = SSLContext.getInstance("TLS");
            kmf = KeyManagerFactory.getInstance("SunX509");
            tmf = TrustManagerFactory.getInstance("SunX509");
            ks = KeyStore.getInstance("JKS");
            ks.load(new FileInputStream("../SSL//paciente.keystore"),
"pacientepassword".toCharArray());
            //OJO, REVISAR
            kmf.init(ks, "pacientepassword".toCharArray());
            tmf.init(ks);
            ctx.init(kmf.getKeyManagers(), tmf.getTrustManagers(), null);

            factory = ctx.getSocketFactory();
            return factory;
        }
        catch (Exception e)
        {
            System.out.println(e.getMessage());
            return null;
        }
    }
}

```

A.2. ProtocolServer.java

```

/*
 * ProtocolServer.java
 *
 * Created on 6 de marzo de 2007, 17:09
 *
 * Clase creada para gestionar las comunicaciones entre el
 * interfaz del programa y la red, encargada de mandar
 * los mensajes al cliente.
 */

package server;

import java.net.*;
import java.io.*;
import java.security.KeyStore;
import javax.net.*;
import javax.net.ssl.*;
import javax.security.cert.X509Certificate;
/**
 *
 * @author Coke
 */
public class ProtocolServer extends Thread {

    private ServerIF interfaz;
    private ServerSocket serverSocket;
    private SSLSocket clientSocket;
    private InputStream input;
    private OutputStream output;
    private ObjectInputStream entradaDatos;
    private ObjectOutputStream salidaDatos;

    private int port=19820; //Puerto elegido al azar entre los no ocupados
    private boolean connected,serverStarted,passwordError=false;

    /**
     * Crea una instancia de ProtocolServer
     */
    public ProtocolServer()
    {
    }

    public ProtocolServer(int puerto)
    {
        port=puerto;
    }

```

```

/**
 * Método que crea el socket SSL de servidor y pone el sistema
 * a la espera de clientes.
 * Tras esto se queda en un bucle recibiendo los diferentes
 * mensajes desde el programa cliente y dándoles la respuesta
 * adecuada, hasta que se cancela la conexión.
 */
public void run()
{
    try
    {
        ServerSocketFactory ssf = getServerSocketFactory();
        serverSocket= ssf.createServerSocket(port);
        ((SSLServerSocket) serverSocket).setWantClientAuth(true);
        ((SSLServerSocket) serverSocket).setNeedClientAuth(true);
        connected=true;

        // Se acepta una conexión con un cliente. Esta llamada se queda
        // bloqueada hasta que se arranque el cliente.

        System.out.println ("Esperando cliente");
        clientSocket = (SSLSocket)serverSocket.accept();
        System.out.println ("Conectado con cliente de "+
clientSocket.getInetAddress());
        input=clientSocket.getInputStream();
        output=clientSocket.getOutputStream();
        entradaDatos=new ObjectInputStream(input);
        salidaDatos=new ObjectOutputStream(output);
        int type;
        while (connected)
        {
            type=typeOfDataReceived(entradaDatos);
            switch (type)
            {
                case 0:
                    tiposDatos.Datos datos=receiveDatos(entradaDatos);
                    interfaz.setDatos(datos);
                    break;

                case 1:
                    System.out.println("El tipo de datos lo lee bien" +
                        " que es de imagen.");
                    tiposDatos.Imagen imagen=receiveImagen(entradaDatos);

                    interfaz.setImagen(imagen);
                    break;

                case 2:
                    System.out.println("El tipo de datos lo lee bien" +
                        " que es de audio.");
                    tiposDatos.Audio audio=receiveAudio(entradaDatos);

```

```

        interfaz.setAudio(audio);
        //connected=false;
        break;

    case 3:
        System.out.println("El tipo de datos lo lee bien" +
            " que es presion arterial.");
        tiposDatos.Presion presion=receivePresion(entradaDatos);
        interfaz.setPresion(presion);
        break;

    case 4:
        System.out.println("El tipo de datos lo lee bien" +
            " que Otras Medidas.");
        tiposDatos.Medida medida=receiveMedida(entradaDatos);
        interfaz.setMedida(medida);
        break;

    case 5:
        System.out.println("El tipo de datos lo lee bien "
            + "que VIDEOCONFERENCIA.");
        interfaz.setWebcam();
        break;

    case 6:
        System.out.println("El tipo de datos lo lee bien" +
            " que AUDIOCONFERENCIA.");
        interfaz.setAudioConferencia();
        break;

    case 7:
        System.out.println("Recibida petición de " +
            "finalización de conferencia.");
        interfaz.stopConferencia();
        break;

    default:
        System.out.println("No entra bien, no pilla " +
            "la opción que le mandamos.");
        if (connected!=false){
            interfaz.resetPatient(); // Si hemos entrado a
            connected=false;} //esta opción voluntariammnt
        break;                                // no hay que resetear
    }
}

}
catch (Exception e)
{
    System.out.println("Error en algún punto: "+e+"\n");
    e.printStackTrace();
}

```

```

        passwordError=true;
        connected=false;
    }
}

/**
 * Método que lee de la entrada un entero para diferenciar el tipo de
 * mensaje que nos envió el cliente.
 */
private int typeOfDataReceived(ObjectInputStream entrada)
{
    try
    {
        return entrada.readInt();
    }
    catch (EOFException eof)
    {
        return -1;
    }

    catch (Exception e)
    {
        System.out.println("Error leyendo el tipo de datos: "+e);
        return -1;
    }
}

/**
 * Método para recibir un objeto de datos por parte del cliente,
 * que posteriormente será pasado al interfaz.
 */
private tiposDatos.Datos receiveDatos(ObjectInputStream entrada)
{
    try
    {
        Object aux=entrada.readObject();
        System.out.println("Llega a leer el objeto de Datos");
        tiposDatos.Datos datos=(tiposDatos.Datos)aux;
        return datos;
    }
    catch (Exception e)
    {
        System.out.println("Error leyendo datos de usuario: "+e);
        return null;
    }
}

/**
 * Método para recibir un objeto de IMAGEN por parte del cliente,
 * que posteriormente será pasado al interfaz.
 */
private tiposDatos.Imagen receiveImagen(ObjectInputStream entrada)
{

```



```

    try
    {
        Object aux=entrada.readObject();
        tiposDatos.Imagen imagen=(tiposDatos.Imagen) aux;
        return imagen;
    }
    catch (Exception e)
    {
        System.out.println("Error leyendo el objeto imagen: "+e);
        return null;
    }
}

/**
 * Método para recibir un objeto de AUDIO por parte del cliente,
 * que posteriormente será pasado al interfaz.
 */
private tiposDatos.Audio receiveAudio(ObjectInputStream entrada)
{
    try
    {
        Object aux=entrada.readObject();
        tiposDatos.Audio audio=(tiposDatos.Audio) aux;
        return audio;
    }
    catch (Exception e)
    {
        System.out.println("Error leyendo el objeto audio: "+e);
        return null;
    }
}

/**
 * Método para recibir un objeto de PRESION ARTERIAL por parte
 * del cliente, que posteriormente será pasado al interfaz.
 */
private tiposDatos.Presion receivePresion(ObjectInputStream entrada)
{
    try
    {
        Object aux=entrada.readObject();
        tiposDatos.Presion presion=(tiposDatos.Presion) aux;
        return presion;
    }
    catch (Exception e)
    {
        System.out.println("Error leyendo el objeto presion: "+e);
        return null;
    }
}

/**

```

```

* Método para recibir un objeto de MEDIDA por parte del cliente,
* que posteriormente será pasado al interfaz.
*/

```

```

private tiposDatos.Medida receiveMedida(ObjectInputStream entrada)
{
    try
    {
        Object aux=entrada.readObject();
        tiposDatos.Medida medida=(tiposDatos.Medida)aux;
        return medida;
    }
    catch (Exception e)
    {
        System.out.println("Error leyendo el objeto medida: "+e);
        return null;
    }
}

public boolean enviaWebcam()
{
    try
    {
        // Enviamos el indicador de petición de transmisión de webcam
        salidaDatos.writeInt(5);
        salidaDatos.flush();
    }
    catch (Exception e)
    {
        System.out.println("Error mandando petición de videoconferencia!");
        return false;
    }
    return true;
}

public boolean enviaDiagnostico(tiposDatos.Diagnostico d)
{
    try
    {
        // Enviamos primero el indicador de que estamos mandando
        // EL DIAGNÓSTICO al cliente esté preparado para
        // la recepción de un objeto de tipo Diagnóstico
        salidaDatos.writeInt(8);
        salidaDatos.flush();
        //Y ahora mandamos los datos propiamente
        salidaDatos.writeObject(d);
        return true;
    }
    catch (Exception e)
    {
        System.out.println("NO SE PUDO ESCRIBIR EL DIAGNÓSTICO");
        return false;
    }
}

```

```

    }
}

/**
 * Método para comunicar al servidor que vamos a detener la conferencia,
 * sea webcam o sólo audio
 */
public boolean stopConferencia()
{
    try
    {
        // Enviamos el indicador de finalización de transmisión
        salidaDatos.writeInt(7);
        salidaDatos.flush();
    }
    catch (Exception e)
    {
        System.out.println("Error mandando finalización de conferencia!");
        return false;
    }
    return true;
}

public boolean startServer(ServerIF iface)
{
    interfaz=iface;
    this.start();
    while (!connected) //Bloqueamos el sistema hasta que se conecta
        if (passwordError) // o nos devuelve un error de password
            return false;
    serverStarted=true;
    return true;
}

public void stopServer()
{
    connected=false;
    try
    {
        serverStarted=false;
        if(clientSocket!=null)
        {
            input.close();
            output.close();
            clientSocket.close();
        }

        serverSocket.close();
        System.out.println("Conexión cerrada...");
    }
    catch (Exception e)

```

```

        {
            System.out.println("Problemas cerrando los sockets...");
            System.out.println(e.getMessage());
        }
    }

    public boolean isStarted()
    {
        return serverStarted;
    }

    public boolean isConnected()
    {
        return connected;
    }

    public Socket getClientSocket() {
        return clientSocket;
    }

    /**
     * Método auxiliar para configurar la conexión
     * segura SSL, que lee las contraseñas del keystore
     * del usuario.
     */

    private ServerSocketFactory getServerSocketFactory() throws Exception
    {
        SSLServerSocketFactory ssf = null;
        String [] userPassword=interfaz.getUserPassword();

        SSLContext ctx;
        KeyManagerFactory kmf;
        KeyStore ks;
        TrustManagerFactory tmf;
        char[] passphrase = userPassword[1].toCharArray();

        ctx = SSLContext.getInstance("TLS");
        kmf = KeyManagerFactory.getInstance("SunX509");
        tmf = TrustManagerFactory.getInstance("SunX509");
        ks = KeyStore.getInstance("JKS");
        ks.load(new FileInputStream("../SSL/"+userPassword[0]+".keystore")
        , passphrase);
        kmf.init(ks,passphrase);
        tmf.init(ks);
        ctx.init(kmf.getKeyManagers(), tmf.getTrustManagers(), null);
        ssf = ctx.getServerSocketFactory();
        return ssf;
    }
}

```

A.3. Ejemplo de tipo de datos: Imagen.java

```

*
* Created on 9 de marzo de 2007, 18:35
*
* Clase de datos de imagen. En ella definimos tanto las
* características de un objeto de este tipo, como los
* métodos utilizados para transmitirlos.
*
*/

package tiposDatos;

import java.io.*;

/**
 *
 * @author Coke
 */
public class Imagen implements Serializable
{
    String file,descrip;

    /**
     * Constructor de la clase imagen, a la que se le pasa de parámetros
     * el nombre del archivo de imagen y la descripción de la misma.
     */
    public Imagen(String f,String d)
    {
        file=f;
        descrip=d;
    }

    /**
     * Como nuestro protocolo está orientado la escritura de
     * objetos en el socket, debemos definir cómo se realiza
     * esta operación en nuestro tipo de datos imagen. Se llama
     * "serializar" a esta acción.
     */

    private void writeObject(java.io.ObjectOutputStream out)
        throws IOException
    {
        out.writeUTF(descrip);

        //Después de mandar la descripcion, mandamos el archivo de
        // imagen, 1kbyte a 1kbyte
        FileInputStream in=new FileInputStream(new File(file));
        byte[] buffer=new byte[1024];
        int len,i=1;
        while ((len=in.read(buffer))>0)

```

```

        {
            out.write(buffer,0,len);
            out.flush();
            i++;
        }
        System.out.println("Hemos escrito en la salida "+i+
            " bufferes de 1k");
        in.close();
    }

/**
 * Este método es el simétrico del anterior, debemos definir también
 * cómo deserializar el objeto recibido, y pasárselo al programa.
 */

private void readObject(java.io.ObjectInputStream in)
    throws IOException
{
    descrip=in.readUTF();
    file="temp.jpg";
    File temporal=new File(file);

    // Después de recibir la descripción pasamos a escribir
    // físicamente el archivo de imagen
    FileOutputStream out = new java.io.FileOutputStream(temporal);
    byte[] buffer = new byte[1024];
    int len;
    while ((len = in.read(buffer)) > 0)
    {
        out.write(buffer, 0, len);
    }
    out.close();

    // Marcamos el archivo temporal utilizado para ser eliminado
    // al salir del programa
    temporal.deleteOnExit();

}

/**
 * Clase auxiliar que simplemente devuelve un vector con el nombre
 * del archivo que contiene la imagen y la descripción.
 */

public String [] getDatos()
{
    String [] aux=new String[2];
    aux[0]=file;
    aux[1]=descrip;
    return aux;
}
}

```

A.4. Otras clases auxiliares

Captura.java

```

/*
 * Captura.java
 *
 * Clase auxiliar creada para la captura de una imagen estática
 * desde una cámara web, tras la inicialización de ésta.
 */

package client;

import javax.media.*;
import javax.media.control.*;
import javax.media.util.*;
import javax.media.format.*;
import javax.imageio.*;
import java.io.*;
import java.awt.*;
import java.awt.image.*;

public class Captura{

    Player player = null;
    Image img = null;
    String dispositivo = "";

    /**
     * Constructor de la clase, al que pasamos como parámetro
     * el dispositivo desde donde realizará la captura.
     * Ej: "vfw://"
     */
    public Captura (String dispositivo)
    {
        try{
            MediaLocator loc=new javax.media.MediaLocator(dispositivo);
            player = Manager.createRealizedPlayer(loc);
            player.start();
            Thread.sleep(1000);
        }
        catch (Exception e)
        {
            System.out.println("Error al iniciar el player");
        }
    }

    /**

```

```

* Método para conseguir la componente visual del reproductor, y
* poder mostrar el vídeo en cualquier parte de una interfaz.
*/

```

```

public Component getPlayer()

```

```

{
    Component visual, contr;
    if ((visual = player.getVisualComponent()) != null)
    {
        //visual.setSize(visual.getPreferredSize());
        visual.setSize(352,288);
        return visual;
    }
    return null;
}

```

```

/**

```

```

* Método que realiza la captura de la imagen estática
*/

```

```

public String capturaImagen()

```

```

{
    FrameGrabbingControl fgc = (FrameGrabbingControl)
    player.getControl("javax.media.control.FrameGrabbingControl");
    Buffer buf = fgc.grabFrame();
    // creamos la imagen awt
    BufferedImage btoi = new BufferedImage((VideoFormat)buf.getFormat());
    img = btoi.createImage(buf);
    String formato = "JPEG";
    String archivo="temp.jpg";
    File imagenArch=new File(archivo);
    imagenArch.deleteOnExit();
    try
    {
        ImageIO.write((RenderedImage) img,formato,imagenArch);
    }
    catch (IOException ioe)
    {
        System.out.println("Error al guardar la imagen");
        return null;
    }
    return archivo;
}

```

```

/**

```

```

* Método para detener la captura de imágenes
*/

```

```

public void stop()

```

```

{
    player.stop();
    player.deallocate();
    player.close();
}
}

```


Lanzador.java

```

/*
 * Lanzador.java
 *
 * Created on 28 de marzo de 2007, 20:26
 *
 * Clase creada para lanzar la aplicación por defecto del
 * sistema para abrir Imagenes, URLs, etc.
 */

package client;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.File;

public class Lanzador {

    private String linuxDesktop=null;

    public Lanzador()
    {
    }

    public void abrir(String archivoURL){

        if((archivoURL.indexOf("http")!=-1)||
            (archivoURL.indexOf("mailto:")!=-1)) launchURL(archivoURL);
        else launchDefaultViewer(archivoURL);
    }

    private String getEnv(String envvar){
        try{
            Process p = Runtime.getRuntime().exec("/bin/sh echo $" + envvar);
            BufferedReader br = new BufferedReader(new InputStreamReader(
                p.getInputStream()));
            String value = br.readLine();
            if(value==null) return "";
            else return value.trim();
        }
        catch(Exception error){
            return "";
        }
    }
}

```

```

private String getLinuxDesktop(){
    //Sólo se averigua el entorno de escritorio una vez,
    // después se almacena en la variable estática
    if(linuxDesktop!=null) return linuxDesktop;
    if(!getEnv("KDE_FULL_SESSION").equals("") ||
        !getEnv("KDE_MULTIHEAD").equals("")){
        linuxDesktop="kde";
    }
    else if(!getEnv("GNOME_DESKTOP_SESSION_ID").equals("") ||
        !getEnv("GNOME_KEYRING_SOCKET").equals("")){
        linuxDesktop="gnome";
    }
    else linuxDesktop="";

    return linuxDesktop;
}

private Process launchURL(String url){
    try{
        if (System.getProperty("os.name").toUpperCase().indexOf(
            "95") != -1)
            return Runtime.getRuntime().exec(
                new String[]{"command.com", "/C", "start",
                    "\"dummy\"", url} );
        if (System.getProperty("os.name").toUpperCase().indexOf(
            "WINDOWS") != -1)
            return Runtime.getRuntime().exec(
                new String[]{"cmd.exe", "/C", "start","\"dummy\"",
                    url} );
        if (System.getProperty("os.name").toUpperCase().indexOf(
            "MAC") != -1)
            return Runtime.getRuntime().exec(
                new String[]{"open", url} );
        if (System.getProperty("os.name").toUpperCase().indexOf(
            "LINUX") != -1 ) {
            if(getLinuxDesktop().equals("kde"))
                return Runtime.getRuntime().exec(
                    new String[]{"kfmclient", "exec", url} );
            else
                return Runtime.getRuntime().exec(
                    new String[]{"gnome-open", url} );
        }
    }
    catch(IOException ioex){System.out.println(ioex);}

    return null;
}

private Process launchDefaultViewer(String filepath){
    return launchURL( new File(filepath).getAbsolutePath());
}
}

```

Apéndice B

Historial de versiones

Para completar la información sobre el desarrollo completo del programa, ofrecemos una relación del historial de versiones del mismo (con los hitos más importantes conseguidos en cada versión), desde el comienzo de su programación hasta la versión definitiva presentada con este trabajo.

Versión 0.0:

- Creación de una primera interfaz primitiva del servidor.
- Creación de las primeras pestañas con funciones

Versión 0.0.5:

- Creación de la interfaz del cliente, basándose en la del servidor.
- Definición momentánea de las pestañas de ambos interfaces.

Versión 0.1:

- Funciones de log in y log out implementadas en servidor.
- Creada función de envío de datos del paciente desde programa cliente.
- Conseguida conexión entre cliente y servidor mediante el uso de sockets normales.
- Logrado intercambio de datos del paciente en forma de objeto, y su representación en el interfaz del servidor.

Versión 0.1.5:

- Lograda la transmisión de imágenes elegidas desde un fichero del ordenador del cliente, y su visualización tanto en la ventana del cliente como en el servidor al recibirla.

Versión 0.2:

- Lograda la captura y reproducción de audio.
- También la transmisión de estos archivos de audio, acompañados de un campo de texto simple de descripción.

Versión 0.2.5:

- Implementada pestaña de diagnóstico en ambos programas.
- Conseguida generación de informe PDF básico en el programa cliente.

Versión 0.3:

- Videoconferencia primitiva conseguida en sentido cliente-servidor, utilizando clases poco eficientes y una conexión diferente para audio y vídeo.

Versión 0.3.5:

- Mejorado creación de informe en PDF, con datos del especialista.
- Funcionando videoconferencia en sentido paciente-servidor, y audio conferencia.

Versión 0.4:

- Implementado almacenamiento de audio e imágenes en un directorio para cada paciente, en la computadora del servidor.
- Funcionando la posibilidad de almacenar y visualizar varias imágenes y sus comentarios, cambiando de una a otra mediante una lista desplegable.

Versión 0.4.5:

- Creación de PDF cambiada al servidor y no en cliente, por seguridad, y almacenamiento de éste en el directorio de usuario.
- Mejoras simples de funcionamiento (limpieza de campos, IP's de los médicos...).
- Certificados de autenticación ya creados, con instrucciones detalladas.

Versión 0.5:

- Gran mejora en estetoscopia: reconocimiento al clicar de los puntos de auscultación cardíaca. Faltan pulmonar y abdominal.
- Inicio directo del servidor tras login de médico.
- Seguridad SSL implementada.
- Parcialmente conseguida recuperación de caída de servidor y cliente, tanto involuntaria como para otra sesión. Limpieza de campos cuando esto ocurre (reset).

Versión 0.6:

- Terminada sección de estetoscopia, con imágenes de ayuda también en servidor.
- Añadida sección de otras medidas, aunque vacía, y eliminación de la pestaña de electrocardiograma.
- Arreglado error con password erróneo de servidor.
- Totalmente conseguida recuperación de caída de servidor y cliente, tanto involuntaria como para otra sesión.

- Implementado la ventanita de "Acerca de..." con logotipo corporativo, desde donde se puede mandar un correo al programador.

Versión 0.7:

- Mejora generalizada de la imagen de los programas, incluyendo hermosos iconos.
- El servidor ahora reconoce cuando se termina la transmisión webcam o audioconferencia, y sin problemas.
- Revisión intensa de la transmisión de conferencias: audio y vídeo transmitidos conjuntos en una misma sesión pero en diferentes pistas, mejora algoritmos, etc.
- Opción de captura de imágenes desde webcam conseguida, para la pestaña de imágenes.
- Información de los centros hospitalarios y especialistas y sus correspondientes IP's ahora es leída de un fichero, sea local o desde Internet.

Versión 0.8:

- Visualización del vídeo propio en el cliente, podrá verlo simultáneo el suyo y el del doctor (aun no implementado).
- Creada categoría de "Presión arterial", con instrucciones para la toma y paso de los resultados al especialista.
- Creada última categoría de "otras medidas", con opción de elegir un tipo de la lista o escribirla.

Versión 0.9:

- Conseguida conferencia bidireccional, sea tanto de vídeo y audio, como solamente audio.
- Lectura de datos del doctor desde un fichero, sea local o desde Internet.

Versión final 1.0:

- Detalles de interfaz.
- Redacción del manual de usuario, y puesta a punto de la función de ayuda en la barra de menús.
- Creación de los archivos de distribución y los instaladores para sistemas Windows.
- Limpieza de código.
- Comentario de funciones.

Bibliografía

- [0] Luis Buñuel, “Los Olvidados”, Ultramar Films, 1950.
- [1] Programa de telemedicina, Universidad Nacional de Córdoba, <http://www.unctelemedicina.com.ar>.
- [2] Instituto Nacional de Estadística Geografía e Informática, “Censo de población y vivienda”, México 2005.
- [3] Ávila de Tomás JF, “Aplicaciones de la telemedicina en atención primaria”, *Atención primaria*. 2001, vol 27, no 1, p. 54-7 , Sociedad Española de Medicina de Familia y Comunitaria.
- [4] G. Eysenbach, J.E.Till, “Ethical issues in qualitative research on internet communities”, *Information in practice*, BMJ Vol 323, Nov 10 2001, p.1103.
- [5] Mejía S, Zuluaga L., Tamayo A., “Diseño de un fonendoscopio digital para la red de telemedicina de Antioquía”, Artículo T102 / VI congreso de la sociedad Cubana de Bioingeniería. 2005.
- [6] Health Adel Medical Health Information Resource, “Auscultation procedure”, *Procedures*, 2007.
- [7] Instituto Químico Biológico “Cardiología: notas”, *Mediclopedia*, 2007.
- [8] Cruz E. Moreno R., “Aparato Respiratorio: Fisiología Clínica”, Mediterráneo. 1990.
- [9] Dr. Alberto Marín A., “Dolor Abdominal”, *Cirugía digestiva y endocrina*.
- [10] Enciclopedia libre Wikipedia, www.wikipedia.org, “Radiología”.
- [11] G. Reséndiz, C. Cabrera, F. Romero, R. Quezada, “Diseño y construcción de un electrocardiógrafo”, Universidad del Valle de México, 2005.

- [12] Moliner de la Puente et altres, “Automedición de la Presión Arterial”, Grupo de Hipertensión Arterial de la AGAMFEC, 2007.
- [13] James P. Cohoon, Jack W. Davidson, “Programación en Java 5.0”, McGraw-Hill/Interamericana, 2006.
- [14] The Netbeans Community, “Netbeans IDE Docs & Support Resources”, <http://www.netbeans.org/kb/index.html>
- [15] ADR Formación, “El modelo de referencia TCP/IP”, *Curso de redes y Windows 2003 Server*, Lección 1, apartado 4.4, 2007
- [16] Enciclopedia libre Wikipedia, www.wikipedia.org, “Internet Socket”.
- [17] J. Martí, M. Soler, B. Olmedo, P.J. González, “Desing of a secure client-server application using SSL sockets”, Project belonging to the Computer Security course, Lunds Tekniska Högskola, 2007.
- [18] Universidad de Málaga, “Tutorial Java Media Framework”, 2003.
- [19] Audio Video Transport Group of the Internet Engineering Task Force (IETF), “RTP: A Transport Protocol for Real-Time Applications”, *RFC 3550*, 2003.

Epílogo

“Un libro, como un viaje, se comienza con inquietud y se termina con melancolía.” Estaba totalmente en lo cierto el filósofo mexicano José Vasconcelos cuando a principios del siglo XX dijo estas palabras.

Pues aquí termina este libro. Sí, puede que sea una osadía intentar considerar el más mínimo parecido entre una obra literaria, que era a lo que el filósofo se refería, y este trabajo, del que estas palabras son su último aliento. Pero al igual que el auténtico lector disfruta con ese maravilloso viaje, el autor considera haber disfrutado y puesto todo su empeño en la consecución de este proyecto y, ahora al terminar, es momento de analizar las impresiones sobre el trabajo realizado.

Monterrey (México), Junio de 2007