



Desarrollo de un sistema de control remoto:
Remote Anywhere

José Ignacio Bengoechea Isasa

Grado en Ingeniería Informática

Consultor: María Isabel March Hermo

Junio 2017

The MIT License

Copyright (c) 2010-2017 Jose Ignacio Bengoechea Isasa, <http://ranywhere.net>

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions.

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

FICHA DEL TRABAJO FINAL

Título del trabajo:	Desarrollo de un sistema de control remoto: Remote Anywhere
Nombre del autor:	José Ignacio Bengoechea Isasa
Nombre del consultor:	María Isabel March Hermo
Fecha de entrega (mm/aaaa):	06/2017
Área del Trabajo Final:	Redes de computadores
Titulación:	Grado de Ingeniería Informática
Resumen del Trabajo (máximo 250 palabras):	
<p>Un sistema de control remoto es aquel que nos permite conectarnos a un equipo de forma remota. En un mundo cada vez más interconectado estos sistemas nos permiten trabajar sin limitaciones de desplazamientos.</p> <p>Los sistemas de control remoto han permitido realizar tareas de mantenimiento y asistencia desde el inicio de la informática. Los primeros sistemas de control remoto eran terminales de texto que nos permitían mandar comandos de texto. Con el tiempo estos sistemas han evolucionado hasta el punto de que podemos controlar todos los procesos de un equipo situado a miles de kilómetros.</p> <p>El objetivo de este proyecto es el desarrollo de un sistema de control remoto que permita conectarnos a la sesión activa del servidor, ver lo que se está mostrando en pantalla y controlarlo remotamente enviando eventos del teclado y del ratón. Todo esto será realizado mediante un modelo basado en la arquitectura cliente/servidor.</p> <p>Esta es una solución de código abierto basada en el desarrollo mediante capas. Esto nos permite en futuras iteraciones añadirle funcionalidades nuevas que nos permitan acercarlo a las soluciones comerciales.</p>	

Abstract (in English, 250 words or less):

A remote-control system is one that allows us to remotely connect to a computer. In an increasingly interconnected world, these systems allow us to work without displacement limitations.

Remote control systems have made it possible to carry out maintenance and assistance tasks since the beginning of the computer age. The first remote control systems were text terminals that allowed us to send text commands. These systems have evolved along the time to the point that we can control all the processes of a computer located thousands of miles away.

The objective of this project is the development of a remote-control system that allows us to connect to the active session of the server, we can see what is being shown on the screen and control it remotely sending keyboard and mouse events. All this will be done using a model based on the client / server architecture.

This is an open source solution based on development by layers. This allows us in future iterations to add new functionalities to approach it to the commercial solutions.

Palabras clave (entre 4 y 8):

Control remoto, arquitectura cliente/servidor, desarrollo en capas

ÍNDICE DE CONTENIDOS

Capítulo 1. Introducción	1
1.1. Contexto y justificación del trabajo	2
1.2. Objetivos del trabajo	3
1.3. Enfoque y método seguido	4
1.4. Planificación del trabajo	5
1.5. Breve resumen de productos obtenidos.....	7
1.6. Breve descripción de los otros capítulos de la memoria.....	7
Capítulo 2. Especificaciones, Análisis y diseño	9
2.1. Estudio del mercado	9
2.2. Requisitos y especificaciones	13
2.3. Diagramas	15
2.4. Mockup de la aplicación	16
2.5. Arquitectura de la aplicación	19
Capítulo 3. Desarrollo.....	23
3.1. Justificación de las tecnologías usadas	23
3.2. Capa de acceso de datos	27
3.3. Capa de lógica de negocios	29
3.4. Capa de presentación	30
3.5. Capa de comunicación	35
Capítulo 4. Pruebas de uso y de rendimiento.....	40
Capítulo 5. Conclusiones del proyecto	42
Capítulo 6. Glosario.....	43
Capítulo 7. Bibliografía	44
Anexo. Manual de instalación, uso y compilación	45
Instalación de la aplicación	45
Uso de la aplicación	46
Instrucciones de Compilación	48

ÍNDICE DE FIGURAS

Figura 1. Acceso remoto a servidor a través de internet.....	1
Figura 2. Entregables del proyecto.....	4
Figura 3. Diagrama de casos de uso.	15
Figura 4. Diagrama de clases.	16
Figura 5. Pantalla principal	16
Figura 6. Configuración del servidor	17
Figura 7. Gestión de usuarios.	17
Figura 8. Registros de accesos realizados al servidor	18
Figura 9. Conexión a un servidor remoto	18
Figura 10. Listado de servidores anotados en el cliente.....	18
Figura 11. Modelo de arquitectura en 3 capas.	21
Figura 12. Modelo de arquitectura en 4 capas.	21
Figura 13. Solución organizada por proyectos.	22
Figura 14. Ejemplo de arquitectura WCF.	25
Figura 15. Captura del entorno de desarrollo usado.....	26
Figura 16. Desarrollo visual del formulario de usuarios.....	30
Figura 17. Formulario de edición de servidores.....	32
Figura 18. Formulario de conexión.	32
Figura 18. Conexión cliente/servidor desde el mismo ordenador.	33
Figura 19. Imagen comprimida en JPEG. Tamaño 100 KB. Fuente uoc.edu.....	38
Figura 20. Imagen comprimida en PNG. Tamaño 513 KB. Fuente uoc.edu.	39
Figura 21. Rendimiento de la aplicación.....	41

A mi familia, sin su ánimo, apoyo y cariño no hubiera sido posible realizar y completar estos estudios de grado.

CAPÍTULO 1. INTRODUCCIÓN

Desde el comienzo de la informática moderna las soluciones de control remoto han permitido acceder a proveedores de servicios que estaban conectados a decenas, centenares o incluso miles de kilómetros. El auge de internet y de las conexiones ha facilitado la creación de una industria relacionado con el trabajo remoto que encuentra en las soluciones de control remoto su principal aliado.

Un programa de control remoto que esté basado en la interfaz gráfica debe permitir realizar las siguientes acciones básicas:

- Conectar el equipo cliente al servidor para autenticarnos como un usuario con permiso de acceso.
- Enviar desde el servidor al cliente una visualización de la consola visual.
- Enviar desde el cliente al servidor las entradas que estamos generando en el teclado y en el ratón del cliente.



Figura 1. Acceso remoto a servidor a través de internet.

Este tipo de soluciones han evolucionado de la misma manera que lo ha hecho el mercado. Inicialmente los programas de control remoto eran terminales de texto que se conectaban al servidor mediante una consola de comandos. Posteriormente se desarrolló una interface visual que permitía interactuar con el dispositivo lejano. En ambos casos este tipo de soluciones estaban limitadas por las características de la red.

El uso de este tipo de aplicaciones a través de internet obligaba a modificar la configuración de los dispositivos de red, mediante redirección de puertos, o de creación de reglas de acceso. Con la aparición de las soluciones basadas en la nube hemos visto que este tipo de desarrollo ha evolucionado permitiendo la conexión remota sin modificar las características de los elementos de red, utilizando servidores intermedios.

Este modelo de aplicaciones utiliza una arquitectura cliente/servidor, en la que el cliente se debe instalar en un dispositivo y se conecta a través de la red al servidor, por medio de un servicio que ha sido instalado previamente en el mismo. El servidor envía al cliente las imágenes relativas a la sesión de usuario que desea capturar, y el cliente envía al servidor los eventos relacionados con las pulsaciones y movimientos del ratón y con las pulsaciones de teclado.

En este trabajo fin de grado el objetivo es el desarrollo de un sistema de control remoto que permita conectarnos a la sesión activa del servidor, ver lo que se está mostrando en pantalla y controlarlo remotamente enviando eventos del teclado y del ratón. Todo esto será realizado mediante un modelo mixto basado en la arquitectura cliente/servidor y en una arquitectura basada en capas.

1.1. CONTEXTO Y JUSTIFICACIÓN DEL TRABAJO

La realización de este proyecto está motivada por la realización de una mejora en las soluciones de control remoto *open source* que están disponibles en la actualidad y tratar de acercar sus funcionalidades a las que ofrecen las soluciones comerciales.

Las soluciones comerciales ofrecen una serie de características comunes:

- Son multiplataforma. Podemos controlar un dispositivo desde un cliente con un sistema operativo distinto al del servidor.
- Optimizan el uso del ancho de banda. Esto lo hacen para facilitar las conexiones a través de redes móviles o con un ancho de banda inferior al que ofrece una red local.
- No requieren que modifiquemos los dispositivos de red. Es decir, no es necesario abrir puertos ni crear reglas de accesos de dispositivos externos.
- Suelen requerir un pago por suscripción. Con lo que periódicamente solicitan a sus usuarios el pago de una licencia para poder seguir disfrutando del servicio.
- Su código es privado, no está abierto a auditorías de seguridad. Esto facilita que cualquier fallo de seguridad en el sistema pueda provocar situaciones de robos de datos o de controles no autorizados.

Este proyecto no se plantea llegar a realizar todas estas funcionalidades, pero si se trata de realizar un sistema básico que permita el control remoto, como detallaremos en el siguiente apartado.

Asimismo, el interés de este proyecto es crearlo en algún tipo de tecnología o lenguaje que este abierta a la filosofía *open source* y que no se haya usado utilizado previamente en los estudios del grado de Ingeniería Informática, para tratar de ver las posibilidades y las limitaciones que nos ofrezca esta tecnología.

1.2. OBJETIVOS DEL TRABAJO

El objetivo general es establecer un sistema de control remoto que nos permita, desde un cliente, conectarnos remotamente, mediante el protocolo de comunicaciones TCP, orientado a conexión.

A continuación, vamos a indicar los objetivos principales que deseamos conseguir con este proyecto.

- Aplicación de control remoto. Es nuestro objetivo principal. Conseguir programar una aplicación cliente/servidor que permita ofrecer servicios de control remoto.
- Esa aplicación deberá ser realizada mediante una arquitectura por capas, con el objetivo de separar funcionalidades. Más concretamente se establecen cuatro capas. Una capa de acceso de datos, una capa de lógica de negocios, una capa de presentación y una capa de conexión al servicio o de comunicación.
- El desarrollo será *open source*, con lo que el código será liberado bajo la licencia MIT, que permite su redistribución y modificación.
- Con respecto a la conexión debemos establecer que sea fiable, es decir deberá poder ser cifrada para evitar intrusiones externas, y que pueda estar comprimida, para reducir el ancho de banda necesario por parte de la aplicación.
- Se realizará un módulo de servidor para control remoto. Este módulo nos permite controlar remotamente el servidor. Tiene las siguientes funcionalidades.
 - Control de acceso de usuarios. No se puede permitir el acceso de usuarios que no hayan sido dados de alta en el servidor.
 - Control de log de accesos. Nos permite revisar los accesos que se han realizado en el servidor.
- Se realizará un módulo de cliente para control remoto. Este módulo nos permite acceder remotamente a un servidor. Tiene las siguientes funcionalidades.
 - Creación, edición y borrado de los servidores a los que puede acceder.
 - Captura de eventos por parte del cliente para ser enviados al servidor.
 - Estudio de elaboración de clientes en dispositivos móviles.

Estos son los entregables que se plantean realizar.

- Código fuente, binarios e instalador de la aplicación.
- Documento de memoria del trabajo.
- Documento de presentación del trabajo.

1.3. ENFOQUE Y MÉTODO SEGUIDO

Este trabajo de fin de grado se ha enfocado con la funcionalidad de establecer un proyecto *open source* que pueda ser ampliado mediante futuras aportaciones, con el objetivo de llegar a alcanzar funcionalidades propias de soluciones comerciales.

Para el desarrollo de este proyecto se ha marcado una planificación por fases, las cuales están marcadas en el calendario mediante hitos de entrega. La elaboración de estos hitos nos permite definir una estructura de descomposición del trabajo, o EDT, que consiste en una descomposición jerárquica orientada al entregable del proyecto para cumplir con los objetivos del mismo.

En la siguiente figura podemos ver los entregables del proyecto. Los cuales establecen en su versión final la aplicación final y la memoria que se entrega.



Figura 2. Entregables del proyecto.

Tenemos tres etapas diferenciadas dentro de las entregas, de izquierda a derecha.

- Etapa de análisis y diseño. Nos permite definir la estructura de la aplicación, las clases y componentes que la forman, y el tipo de tecnología sobre el que se desarrolla.
- Etapa de implementación. Que dividimos en dos entregables, un prototipo que nos muestra el aspecto y las funcionalidades del mismo. Este entregable itera hacia un prototipo que nos permite el control remoto de otro dispositivo.
- Etapa de documentación. La cual integra la documentación que se ha realizado en las etapas anteriores y nos permite ver el aspecto global del proyecto. A partir de esta etapa se desarrolla la presentación del trabajo final.

1.4. PLANIFICACIÓN DEL TRABAJO

El resumen de la planificación que se ha elaborado es la siguiente:

Fase	Fecha de fin	Descripción
Fase 1	19 de marzo	Recogida de información
Fase 2	2 de abril	Análisis y diseño
Fase 3	24 de abril	Implementación del prototipo
Fase 4	28 de mayo	Implementación de la versión final
Fase 5	11 de junio	Integración de la memoria
Fase 6	18 de junio	Creación de la presentación

La descomposición de las fases por detalles se realiza con la siguiente planificación:

FASE 1

Entrega: 19 de marzo

Puntos que se deben realizar:

- Recogida de información relacionada con:
 - Temas relacionados con la comunicación, compresión y encriptación de datos por TCP y UDP.
 - Temas relacionados con el desarrollo y la implementación de software cliente / servidor.
 - Temas relacionados con el desarrollo de aplicaciones basadas en dos y tres capas.
- Estudio de opciones existentes en el mercado relacionadas con los objetivos planteados.

Objetivos:

- Documentación relativa a los procesos de recogida de información.
- Documentación relativa a las funcionalidades que se ofrecen en el mercado comercial y open source.

FASE 2

Entrega: 2 de abril.

Puntos que se deben realizar:

- Establecimiento de requisitos funcionales y no funcionales.
- Análisis de las funcionalidades que se establecen en el cliente y en el servidor.
- Diagrama de clases organizado por capas.
- Establecer el diseño de la aplicación mediante *mockups*.
- Establecer el lenguaje en el que se realizara la aplicación y la plataforma inicial.

Objetivos:

- Documentación relativa a los requisitos del sistema.
- Documentación relativa a los diagramas.
- Documentación relativa al diseño.

FASE 3

Entrega: 24 de abril.

Puntos que se deben realizar:

- Entrega de una versión no funcional que nos muestra el aspecto grafico de la misma.
- Se podrá ver el aspecto grafico del cliente y el del servidor.

En el cliente:

- Se podría crear, editar, seleccionar y borrar el servidor al que se conectara remotamente.
- El servidor se define con 3 parámetros. El nombre, la ip y el puerto de acceso.

En el servidor:

- Puede crear, editar, seleccionar y borrar el listado de contraseñas que se podrán usar para conectar remotamente.
- Puede seleccionarse el puerto de escucha del mismo

Objetivos:

- Prototipo del cliente.
- Prototipo del servidor
- Documento sobre las decisiones tomadas en esta etapa

FASE 4

Entrega: 28 de mayo

Puntos que se deben realizar:

- Entrega de una versión final que debe permitir los siguientes objetivos:

En el cliente

- Establecer conexión de acceso remoto con un servidor, mediante el protocolo TCP.
- Visualizar la pantalla del servidor y controlarla remotamente enviando eventos del ratón y del teclado.
- En el momento de la conexión se deberá indicar el usuario y la contraseña para acceder al servidor.
- Pruebas y modificaciones necesarias en el cliente.

En el servidor

- Insertar en un log las conexiones que se han establecido.
- Pruebas y modificaciones necesarias en el servidor.
- Estudio de los recursos requeridos por la aplicación. Consumo de CPU y de memoria.

Objetivos:

- Aplicación final del cliente.
- Aplicación final del servidor.
- Documento sobre las decisiones tomadas en la fase de pruebas.
- Documento con las características de rendimiento de la aplicación final.

FASE 5

Entrega: 11 de junio

Puntos que se deben realizar:

- Se ensamblará la memoria a partir de la generación de la documentación creadas en las fases anteriores.

- Documentación relativa a la instalación y configuración de las aplicaciones cliente y servidor.

Objetivos:

- Síntesis de la memoria del proyecto.

FASE 6

Entrega: 18 de junio

Puntos que se deben realizar:

- Creación de la presentación del proyecto.
- Subida del documento a la plataforma *Open Access* de la UOC.

Objetivos:

- Entrega final del proyecto.

1.5. BREVE SUMARIO DE PRODUCTOS OBTENIDOS

La realización del proyecto proporciona los siguientes productos obtenidos:

- Binarios e instalador de la aplicación de control remoto, la cual nos permite bajo el sistema operativo Windows hacer conexiones remotas.
- Código fuente de la aplicación de control remoto. El código tiene licencia MIT, la cual es una licencia de software permisiva que impone muy pocas limitaciones en la reutilización de dicho código.
- Memoria del proyecto realizado. Es esta memoria en la cual se analiza el proceso de creación del proyecto.

1.6. BREVE DESCRIPCIÓN DE LOS OTROS CAPÍTULOS DE LA MEMORIA

En los siguientes capítulos se explicará con detalle los siguientes apartados relativos a este proyecto.

- Capítulo 2. En este capítulo estudiaremos las soluciones que ofrece el mercado relativas a aplicaciones de control remoto, así como el análisis y diseño de la solución que se quiere desarrollar.

Para ello se elaborarán unas especificaciones a partir de las necesidades que se han planteado, y a partir de las características de las aplicaciones de control remoto que existen en el mercado. En el análisis y diseño se mostrarán los diagramas necesarios para establecer las funcionalidades del software.

- Capítulo 3. En este capítulo se dan detalles sobre la tecnología que se usa, sobre el sistema operativo que queda cubierto por dicha tecnología y sobre las funcionalidades que nos ofrece el modelo de desarrollo planteado.

También se muestra el desarrollo realizado, la utilización de código propio y de terceros, el resultado de las pruebas de rendimiento del sistema, y las posibilidades que nos ofrece el código para posibles extensiones.

- Capítulo 4. En este capítulo se mostrarán las pruebas realizadas y el análisis del rendimiento de la aplicación.
- Capítulo 5. En este capítulo se mostrarán las conclusiones del proyecto.
- Anexo 1. En este anexo se mostrarán las instrucciones de compilación del código fuente, el manual de uso de la aplicación y los requisitos necesarios para la instalación.

CAPÍTULO 2. ESPECIFICACIONES, ANÁLISIS Y DISEÑO

2.1. ESTUDIO DEL MERCADO

Existe una amplia diversidad de sistemas de control remoto, ya que desde muchos años antes de que se desarrollasen los protocolos actuales de comunicación ha existido la necesidad de conectarse a equipos remotos y administrarlos. Vamos a fijar las características de las funcionalidades que queremos estudiar para este proyecto.

- **La aplicación de control remoto debe basarse en el modelo OSI.**

La mayoría de las aplicaciones que se conectan a internet utilizan el modelo de 7 capas de OSI, en el cual se relega el transporte de datagramas a la capa 3 de red, y el transporte de paquetes a la capa 4 de transporte.

- **Debe utilizar IPv4 o IPv6.**

Existen aplicaciones de control remoto que utilizan otro tipo de direccionamiento, como RS485, RS232, directo a MAC, o incluso por infrarrojos. En este caso nos centramos en aplicaciones que realizan su conexión a través del protocolo IP.

- **Puede utilizar UDP o TCP.**

Lo cual debe resultar obvio, si se usa OSI entonces estas limitado a usar TCP o UDP, ya que son las dos opciones que se tienen en la capa de transporte, pero no estamos limitados a utilizar solo una de ellas, las dos por separado tienen sus ventajas e inconvenientes.

TCP. Está orientado a conexión. Lo cual nos simplifica mucho el desarrollo ya que los paquetes se recibirán ordenados y en caso de pérdida se enviarán de nuevo. Es fácil de programar, pero su uso puede limitarnos porque nos obligara a redireccionar puertos para conectarnos a equipos que están detrás de un firewall, o puede provocar que la latencia del programa sea muy elevada porque está a la espera de recibir un paquete.

UDP. Es un protocolo sin negociación de conexión en el que los paquetes pueden no llegar nunca, o pueden recibirse desordenados. Es más complicado de desarrollar porque obliga a crear una lógica de recepción de paquetes, pero da un mayor control sobre la latencia ya que podemos ignorar los paquetes no recibidos y nos permite jugar

con las conexiones para tratar de conectarnos a los equipos que están detrás de firewalls.

- **Redirección de puertos**

Al utilizar TCP abrimos un puerto en el servidor que se queda a la espera de recibir una conexión. Si el servidor está detrás de un firewall es necesario redireccionar ese puerto en el firewall, mediante NAT, para que todas las peticiones recibidas por el firewall a ese puerto sean redireccionadas al equipo servidor.

Esto es posible en ámbitos domésticos o en pequeñas y medianas empresas, pero en empresas corporativas las restricciones de seguridad no van a permitir que se vayan redireccionando puertos por cada equipo que se quiera acceder remotamente. Por ello, los administradores remotos suelen tener accesos por VPN que permiten el acceso a la red situada detrás del firewall.

Existen varias soluciones comerciales como *Remote Admin* o *RealVNC* que permiten el acceso por TCP. Son soluciones con un pago único, a diferencia de las basadas en UDP como veremos a continuación.

- **UDP perforado**

Al utilizar UDP podemos desarrollar una técnica que nos permite sobrepasar el NAT del firewall sin necesidad de realizar un redireccionamiento de puertos. El nombre de esta técnica es UDP perforado, o *hole punching* en inglés.

Imaginemos dos equipos A y B, que están situados detrás del NAT de sus firewalls, los cuales son NAT_A y NAT_B. Asimismo, existe un equipo llamado S el cual admite conexiones externas.

- A y B se han conectado con S mediante UDP.
- Los dispositivos NAT_A y NAT_B han creado puertos temporales de salida para estas conexiones, los cuales se llaman EP_A y EP_B. Las IP de salida respectivas son IP_A, la IP_B.
- S anota la IP_A, la IP_B, EP_A, EP_B. Le envía a B los datos IP_A, EP_A y a A los datos IP_B, EP_B.
- A manda un paquete UDP a IP_B por el puerto EP_B. El NAT_B se lo admite y lo redirecciona a la dirección local de B, porque ya estaba creado el registro previo.
- B manda un paquete UDP a IP_A por el puerto EP_A. El NAT_B se lo admite y lo redirecciona a la dirección local de A, porque ya estaba creado el registro previo.

- Los dos equipos están conectados por UDP y pueden enviarse datos.

Como vemos UDP puede mejorar la funcionalidad de la aplicación, pero desarrollar en UDP implica crear nuestra propia pila de la capa de transporte. Existen varias soluciones de control remoto como *Teamviewer* o *Logmein* que realizan esta aproximación al sistema. Como es necesario mantener un equipo externo para asegurar las conexiones son soluciones comerciales con pago por suscripción.

- **Protocolo de la capa de aplicación**

Como cualquier otra aplicación que se distribuye por internet, las soluciones de control remoto tienen un protocolo definido en la capa de aplicación, el cual va a definir el funcionamiento de la misma.

En esta capa es donde vamos a definir si se va a usar un protocolo con UDP perforado, si se van a comprimir dinámicamente las imágenes, o sencillamente si se va a utilizar otro protocolo que ya se ha usado con anterioridad. Veamos que protocolos existen.

- **RFB. Remote Framebuffer.** Desarrollado por Olivetti. Es el protocolo utilizado por *RealVNC* o sus derivados. Para poder enviar los datos de la pantalla remota se realiza una captura de la pantalla pixel a pixel, la cual se envía por la red y se recrea en el cliente.

Esto en la práctica es la forma más lenta de enviar la información, ya que es necesario enviar toda la información, aunque solo haya cambiado una parte de la pantalla. Por eso *RealVNC* suele ir muy lento en conexiones que no son locales. En la actualidad el protocolo RFB es un protocolo libre que se puede usar en implementaciones con licencia *open source*.

- **RDP. Remote Desktop Protocol.** Desarrollado por Microsoft. Es un protocolo que trabaja con un driver del núcleo del sistema operativo. En este caso lo que se envía no es la información gráfica, sino la información para poder reconstruir la imagen remota mediante llamadas al subsistema gráfico de Windows. Es decir, se envían los comandos que crean las imágenes.

Esto hace que las conexiones RDP a través de internet sean más fluidas que en RFB, aunque en ciertas ocasiones la carga de ventanas complica el proceso de renderizado y crea cierta latencia.

RDP estaba inicialmente limitado a trabajar con clientes de Windows, pero con el tiempo se han desarrollado implementaciones libres del mismo que permiten extenderlo a otros sistemas operativos como Linux y Mac.

- **Protocolos propietarios.** Existen soluciones que utilizan protocolos propios como *Teamviewer* o *Logmein*. Estas son soluciones que ofrecen facilidad de instalación y de uso, las cuales están basadas en pago por suscripción.

Su principal premisa es la posibilidad de conectarse a equipos que están situados detrás de firewall corporativos. Asimismo, permiten el trabajo remoto en condiciones de bajo ancho banda, ya que usan compresión dinámica de imágenes y envían solo las diferencias entre una imagen y la siguiente.

- **Características adicionales.**

Estas son funcionalidades que no son básicas para el servicio de acceso remoto, sino que son extensiones que permiten abarcar necesidades que el mercado ha ido solicitando en este tipo de soluciones.

- **Encriptación de comunicaciones.** Las necesidades de seguridad actuales obligan a que las comunicaciones estén encriptadas para evitar que haya problemas de seguridad en la autenticación o en el envío de datos.
- **Transferencia de archivos.** Esta funcionalidad permite la transferencia de archivos entre el cliente y el servidor.
- **Modos de acceso limitado.** Los modos de acceso limitados son aquellos en los que el cliente solo puede ver la pantalla, no puede interactuar con el sistema.
- **Reuniones.** Esta funcionalidad permite que haya vaya varios clientes conectados a un mismo servidor en modo de acceso limitado. De esta forma el ponente, que está en el servidor puede moderar la reunión y mostrar una presentación.
- **Soporte de audio.** Las sesiones de audio se han popularizado con el uso de reuniones virtuales. El ponente puede hablar y explicar la presentación que está mostrando.
- **Sesiones múltiples.** Las sesiones múltiples son usadas en reuniones y en protocolos como RDP o SSH, que permiten que varios clientes hagan sesión.

2.2. REQUISITOS Y ESPECIFICACIONES

La aplicación *Remote Anywhere* trata de ser una aplicación de control remoto robusta y sencilla. Su desarrollo se fundamenta en los siguientes puntos:

MODOS DE TRABAJO

La aplicación puede trabajar en tres modos, los cuales son configurables desde una pantalla en la que se selecciona el modo en el que debe arrancar.

- Modo de cliente. La aplicación solo arrancará el módulo del cliente, el cual nos permite conectarnos a un servidor remoto.
- Modo de servidor. La aplicación solo arrancará el módulo del servidor. En este modo la aplicación se queda a la espera de recibir conexiones externas a través del puerto que se ha configurado en la misma.
- Modo mixto cliente/servidor. En este modo la aplicación tiene activos ambos módulos. Puede actuar como cliente y como servidor.

Para poder establecer esta funcionalidad es necesario que se creen módulos separados para el cliente y el servidor.

PROTOCOLO DE COMUNICACIÓN

La aplicación usará el protocolo TCP, orientado a conexión, para facilitar la comunicación entre cliente y servidor y evitar la pérdida de paquetes. Sin embargo, al ser un software que se va a liberar se quiere diseñar el sistema de forma que se pueda modificar esta opción en el futuro y utilizar el protocolo UDP. Para cumplir esta funcionalidad es necesario que se cree una capa de comunicación que esté separada del resto de capas.

El uso del protocolo TCP implica que si queremos conectarnos a un sistema remoto que no esté en la misma red local sea necesario establecer una redirección de puertos hacia el servidor.

El módulo de comunicación debe tener la opción de permitirnos encriptar y comprimir la comunicación.

SESIONES MÚLTIPLES

La aplicación no está diseñada para que varios usuarios puedan autenticarse en un servidor a la vez. Si el servidor detecta que un usuario trata de entrar, cuando está en uso, se le denegará el acceso. El razonamiento de esta funcionalidad es que si más de un usuario entra en el sistema

a la vez se recibirán peticiones de entrada de ambos usuarios, lo cual genera un conflicto.

FUNCIONALIDADES DEL MODULO CLIENTE

El módulo del cliente nos permitirá realizar las siguientes funciones:

- Crear, editar y borrar la lista de servidores a las que nos podemos conectar.
- Establecer comunicación remota con un servidor tras realizar una autenticación válida en dicho servidor.
- Enviar los eventos relacionados con el movimiento y las pulsaciones del ratón al servidor.
- Enviar los eventos relacionados con las pulsaciones del teclado al servidor.
- Reconponer la imagen que se envía a través del módulo de comunicación desde el servidor al cliente y escalarla adecuadamente según el tamaño de la pantalla del cliente.

FUNCIONALIDADES DEL MODULO SERVIDOR

El módulo del servidor nos permitirá realizar las siguientes funciones:

- Crear, editar y borrar la lista de usuarios y contraseñas que pueden tener acceso remoto en ese servidor.
- Activar el servidor en el puerto que se haya configurado y dejarlo a la espera de recibir una conexión remota.
- Reconponer los eventos relacionados con el movimiento y las pulsaciones del ratón del cliente.
- Insertar en un log las conexiones que se han establecido.
- Reconponer los eventos relacionados con las pulsaciones del teclado del cliente.
- Enviar una captura de pantalla de la sesión actual del servidor.

2.3. DIAGRAMAS

En esta etapa se presentan los diagramas relacionados con el diseño de la aplicación, como el diagrama de casos de uso que nos muestra las funcionalidades relacionadas con los agentes, el diagrama de clases que nos muestra la división en clases funcionales y la organización de las mismas en capas.

DIAGRAMA DE CASOS DE USO

Los casos de uso nos permiten identificar las entidades y actores que interactúan con el sistema, así como la funcionalidad básica que se desea implementar. La siguiente figura muestra el diagrama de casos de uso para nuestra aplicación. Dicho diagrama lo hemos dividido en dos paquetes, uno para el cliente y otro para el servidor.

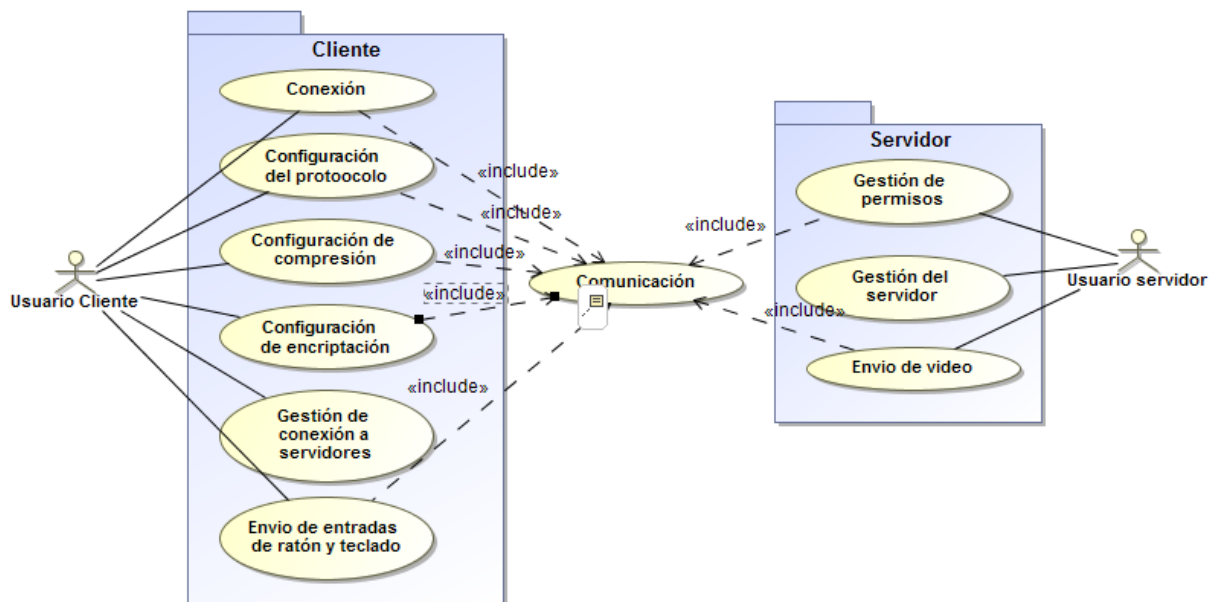


Figura 3. Diagrama de casos de uso.

Como podemos ver en el diagrama, los casos de uso relacionados con la conexión, con la configuración del mismo, con envío de datos y con la autenticación tienen como prerrogativa el caso de uso de comunicación, ya que necesitan que se establezca para que se llegue a realizar cada caso de uso.

DIAGRAMA DE CLASES

Este es el diagrama de clases donde podemos ver la interacción que existen entre las clases de la aplicación.

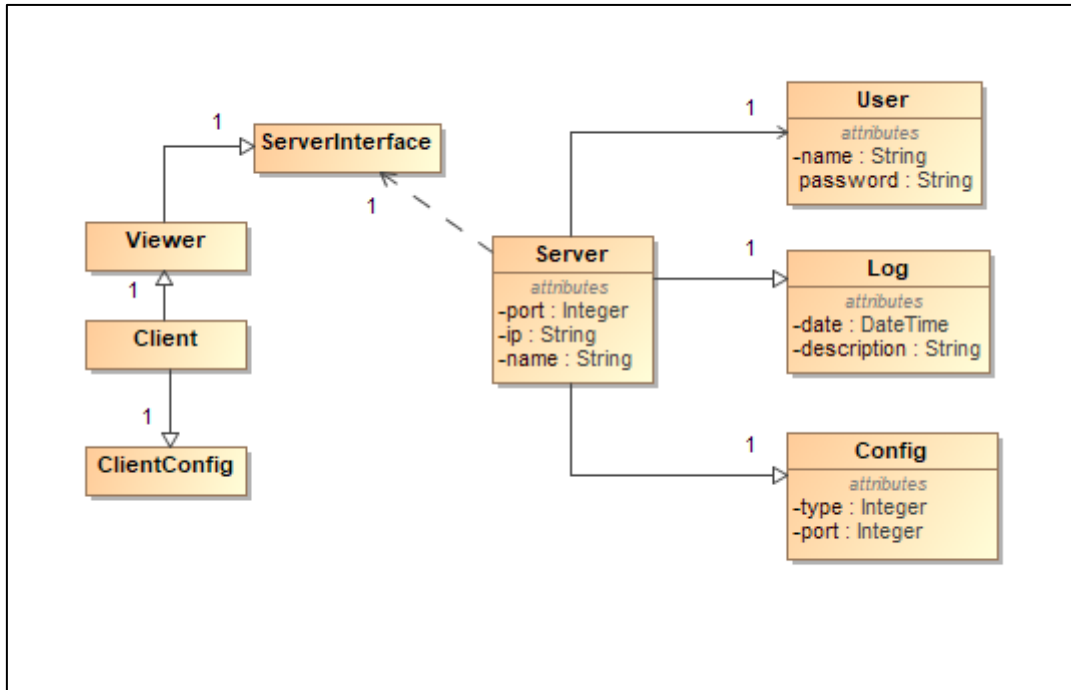


Figura 4. Diagrama de clases.

Por un lado, podemos identificar las clases que relacionan al servidor como su configuración, los registros de acceso y la clase de usuarios que pueden acceder al servidor. Por otro lado, tenemos la interacción entre el cliente y el servidor que se modela a través de una interface.

2.4. MOCKUP DE LA APLICACIÓN

En ingeniería del software un mockup es un bosquejo que muestra la interfaz de usuario, sin necesidad de tener que crear un prototipo. Nos da una idea de las funcionalidades que se pueden realizar desde la interfaz. Este es el mockup de la aplicación.

La figura 5 nos muestra la pantalla principal desde la que tenemos la opción de ver las opciones de cliente y de servidor.

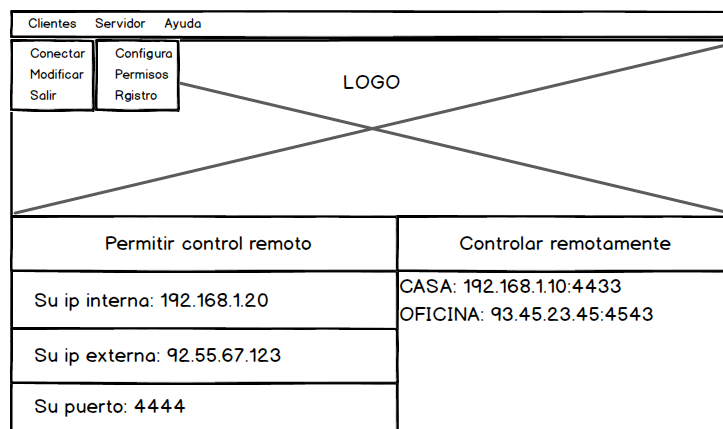


Figura 5. Pantalla principal

Como cliente podemos:

- Conectar a un equipo remoto desde la opción Clientes/Conectar.
- Modificar el listado de clientes que se han creado desde Clientes/Modificar.
- Conectarnos directamente desde la pantalla principal pulsando el servidor al que queremos conectarnos desde el lado derecho de la pantalla principal.

Como servidor podemos:

- Configurar el servidor desde Servidor/Configura
- Establecer los usuarios que pueden acceder al servidor desde Servidor/Permisos.
- Revisar los registros de los accesos que se han realizado desde Servidor/Registro.
- También podemos ver en la pantalla principal los datos de nuestro servidor, en el lado izquierdo de la pantalla.

La figura 6 nos muestra la pantalla de configuración del servidor donde podemos indicar el puerto que debe escuchar o modificar el estado del servidor.

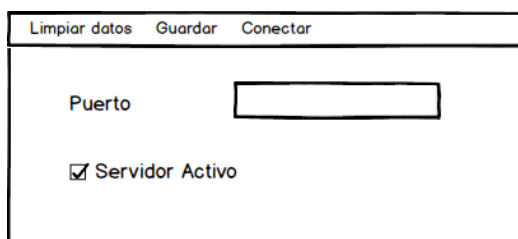
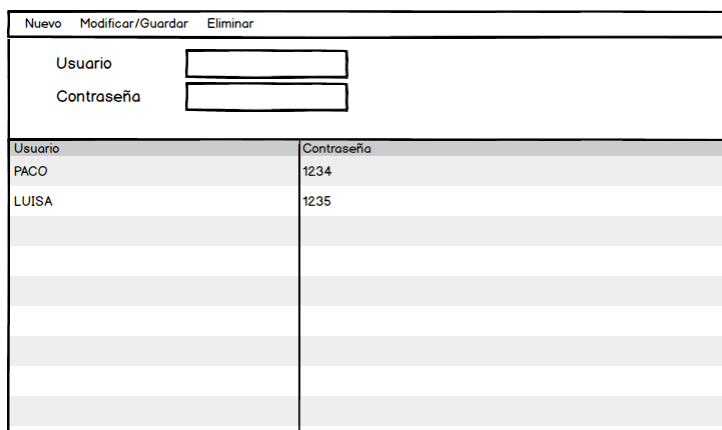


Figura 6. Configuración del servidor

La figura 7 nos muestra la pantalla de gestión de usuarios que tienen acceso al servidor. Donde podemos crear, modificar y eliminar los usuarios que tienen acceso al servidor.



Usuario	Contraseña
PACO	1234
LUIISA	1235

Figura 7. Gestión de usuarios.

La figura 8 nos muestra la pantalla de registro de accesos del servidor, donde vemos el histórico de accesos ordenados por fecha descendente.

Logs ...	
Descripcion	Fecha
CONEXION FINALIZADA POR PACO	12/04/2017 15:42
CONEXION REALIZADA POR PACO	12/04/2017 15:25
FALLO DE AUTENTICACION EN LA CONEXION	12/04/2017 15:24

Figura 8. Registros de accesos realizados al servidor

La figura 9 nos muestra la pantalla de conexión a un servidor remoto, donde podemos conectarnos en el momento o darlo de alta para conectarnos después. El servidor se identifica con el nombre, la ip y el puerto.

Limpiar datos Guardar Conectar	
Nombre	<input type="text"/>
IP	<input type="text"/>
Puerto	<input type="text"/>

Figura 9. Conexión a un servidor remoto

La figura 10 nos muestra la pantalla de gestión de servidores que hemos dado de alta en el cliente.

Nuevo Modificar/Guardar Eliminar				
Nombre	<input type="text"/>			
IP	<input type="text"/>			
Puerto	<input type="text"/>			
Nombre	Ip	Puerto	Compresion	Encriptación
CASA	192.168.1.10	4433	SI	SI
OFICINA:	93.45.23.45	4543	SI	SI

Figura 10. Listado de servidores anotados en el cliente

2.5. ARQUITECTURA DE LA APLICACIÓN

En este apartado se determina que tipos de arquitecturas pueden, y deben, ser usadas en la aplicación, para poder cubrir las necesidades requeridas. Este es el grupo de necesidades que nos van a permitir seleccionar el tipo de arquitectura que vamos a utilizar.

- El desarrollo de la aplicación quiere ser portable en un futuro. Esto quiere decir que queremos independizar el acceso a la plataforma, es decir que se puedan realizar desarrollos en Windows, en Android o en iOS. Una opción viable de realizar este requerimiento es separar en capas la parte de la lógica de negocios y la parte de presentación.
- El desarrollo trata de ser independiente de la base de datos. En plataformas Windows se puede usar como base de datos Sql Server Compact, pero en plataformas Android no hay acceso a esa base de datos, por lo que se plantea usar SqlLite. Este proceso requiere independizar el acceso a la base de datos.
- Tenemos necesidad de realizar pruebas con diferentes tipos de comunicaciones. Es decir, debemos probar conexiones TCP, UDP, distintos tipos de compresión de datos, distintos tipos de encriptaciones. Por ello, la arquitectura seleccionada debería permitirnos comunicar objetos del cliente con objetos del servidor mediante una interface definida que no nos obligue a utilizar un determinado tipo de comunicación.
- La aplicación debe ser distribuida, es decir, las tareas no se realizan en un único dispositivo, sino que se distribuyen entre los proveedores de servicios, o servidores, y los consumidores de servicios, o clientes.

Tras revisar los estilos arquitectónicos que podemos utilizar observamos que existen varias opciones posibles de combinación de arquitecturas. Teniendo en cuenta varios factores, como el punto de vista de información, el computacional y el ingenieril llegamos a la conclusión de que podemos construir la aplicación como una combinación de los siguientes estilos de arquitectura.

- Arquitectura cliente servidor. La cual nos va a permitir diferenciar las funcionalidades que deben estar en el módulo del cliente y en el módulo del servidor. De esta forma podemos separar ambas funcionalidades y evitar que el cliente se vea obligado a realizar tareas del servidor y viceversa.
- Arquitectura por capas o niveles. La cual nos va a permitir separar la capa de acceso de datos, la capa de lógica del negocio y la capa de presentación. De forma que, cuando en el futuro se quiera ampliar el desarrollo a otras plataformas mediante el uso de Xamarin se pueda reutilizar código.

ARQUITECTURA CLIENTE/SERVIDOR

La arquitectura cliente servidor en una aplicación se modela como un conjunto de componentes llamados servidores, que nos ofrecen unos servicios, y un conjunto de componentes llamados clientes que consumen dichos servicios. Podemos ver una serie de características asociadas a nuestra aplicación:

- Tenemos un conjunto de servidores donde se va a gestionar.
 - Configuración relativa al servidor. Puerto de escucha, servidor activado/desactivado.
 - Control de los usuarios que pueden acceder al mismo.
 - Control de registros de accesos, con autenticación válidas y fallidas.
 - Envío y recepción de datos al cliente.
- Tenemos un conjunto de clientes donde se va a gestionar.
 - Gestión de servidores a los que puede acceder.
 - Proceso de conexión al servidor.
 - Parámetros de configuración de la conexión.
 - Envío y recepción de datos al servidor.

Desde el punto de vista de la ingeniería tenemos dos opciones:

- Podemos diferenciar estas funcionalidades en módulos separados, de forma que tendemos un módulo para los clientes y otro para los servidores. Esta opción se suele utilizar en entornos administrativos para que los ordenadores donde se tenga acceso no tengan capacidad de conectarse a otros, ya que solo los administradores tienen acceso a la aplicación cliente.
- Podemos usar el mismo modulo para ambos y diferenciar en cada instalación si debe actuar como un servidor, como un cliente, o como ambos. Esta opción se suele utilizar en programas donde se quiere simplificar el manejo, de forma que el usuario no se vea obligado a instalar 2 aplicaciones. En el desarrollo del prototipo se ha escogido esta opción para simplificar la cantidad de aplicaciones que se van a desarrollar.

ARQUITECTURA DE N CAPAS

El objetivo principal en una arquitectura de n capas es separar el código de diseño, el código asociado a las funcionalidades y el código asociado al acceso de la base de datos. La principal ventaja de este desarrollo es que nos permite realizar futuros cambios del código centrándonos en las funciones que ofrece cada capa.

De esta forma, si en el futuro queremos cambiar la base de datos a la que accedemos, por ejemplo, pasando de Sql Server Compact a SqlLite, solo debemos modificar el código dentro de

la capa de acceso a datos. De la misma forma, si nos planteamos el desarrollo de esta aplicación en Android debemos migrarla a Xamarin, lo cual nos obliga a modificar la capa de presentación.

En la siguiente figura podemos ver el modelo planteado con 3 capas. Donde cada capa puede acceder a los métodos y propiedades públicos de la capa previa.

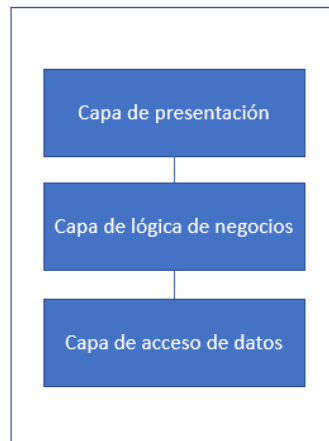


Figura 11. Modelo de arquitectura en 3 capas.

Hay que tener en cuenta que este acceso puede ser local si todas las capas se encuentran en el mismo proceso, o puede ser remoto si las capas se encuentran distribuidas en procesos diferenciados. En nuestro caso el acceso será local, y las tres capas se ejecutarán desde el mismo proceso. Sin embargo, existirán comunicaciones entre las capas de lógica de negocio del cliente y del servidor.

En nuestro prototipo nos planteamos el uso de una cuarta capa donde se establece la lógica de la comunicación entre cliente y servidor. Esa capa es accesible desde la capa de presentación y nos permite lanzar el proceso de comunicación si tenemos activado el módulo del cliente, o poner el servicio de control remoto en escucha, si tenemos activado el módulo del servidor.

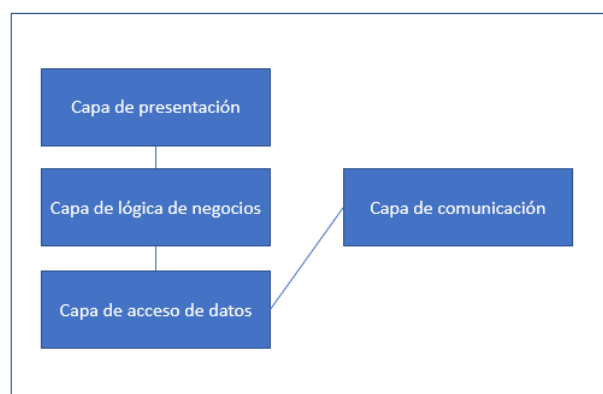


Figura 12. Modelo de arquitectura en 4 capas.

Desde el punto de vista del desarrollo, en Visual Studio, esta arquitectura se puede modelar mediante el uso de proyectos, de forma que cada proyecto representa una capa. Dentro de cada proyecto se referencia aquellos a los que podemos acceder.

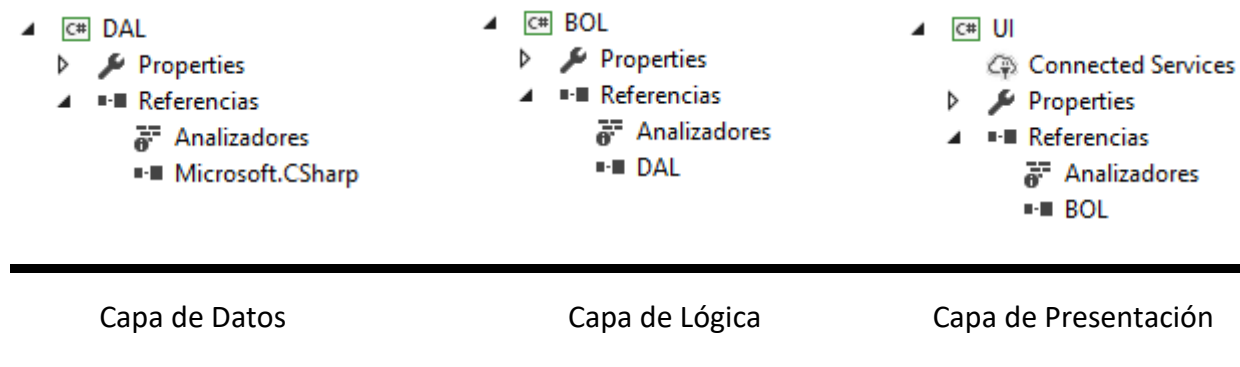


Figura 13. Solución organizada por proyectos.

CAPÍTULO 3. DESARROLLO

En este capítulo analizamos las tecnologías que se han usado en este proyecto. También realizamos el desarrollo e implementación del mismo mediante el uso de capas. Todo el código fuente del proyecto está disponible en la carpeta *src*.

3.1. JUSTIFICACIÓN DE LAS TECNOLOGÍAS USADAS

Uno de los objetivos principales es utilizar una tecnología que no se haya visto durante el grado, eso descarta Java y hace muy atractivo el uso de la tecnología Net Framework de Microsoft.

Hace unos años el uso de tecnología de Microsoft en una aplicación *open source* no era bien visto por el resto de la comunidad, ya que Microsoft ha tenido desde su inicio, una política privada y comercial en lo relativa a la apertura de su código fuente.

Sin embargo, en la actualidad existe un movimiento de portabilidad y apertura por parte de Microsoft, que, aunque es parcial, ha permitido el desarrollo de un framework portable entre diversos sistemas operativos, conocido como Net Core. Net Core nos permite desarrollar aplicaciones de consola que se pueden ejecutar en sistemas operativos Windows, GNU/Linux y Mac. La limitación que nos impide usarlo es que solo sirve para aplicaciones de consola.

Esta es una aplicación de tipo cliente servidor, por lo que para permitir la comunicación entre objetos se plantea que tecnología de las que ofrece Net Framework nos puede resultar más interesante:

- Net Remoting. Es un conjunto de librerías que nos permite utilizar métodos remotos, así como el paso de tipos de datos.
- Web Services. Es una capa lógica de negocio accesible por medio de protocolos web, como SOAP o REST, que usan XML y JSON, para recibir y enviar datos. Requiere la instalación de un servidor de aplicaciones como Internet Information Server, o IIS.
- Windows Communication Foundation, o WCF. Es un conjunto de librerías que permite la construcción de aplicaciones orientadas a servicios. Es la evolución de los dos sistemas anteriores, ya que unifica Net Remoting, y Servicios Web.

PLATAFORMAS DE DESARROLLO

El uso de tecnologías de Microsoft puede limitarnos el rango de plataformas con el que podemos desarrollar. El objetivo inicial de este proyecto era realizar el desarrollo en Xamarin el cual nos permite el desarrollo multiplataforma en Windows 10, Mac, Android y en iOS, mediante el lenguaje de programación C# y la librería Mono que es compatible con Net Framework.

Sin embargo, Xamarin no nos permite el desarrollo en versiones anteriores de Windows como Windows 7 u 8. Como el desarrollo multiplataforma no es el objetivo principal del proyecto se plantea continuar la línea de trabajo en *WinForms*, el cual nos permite desarrollar para cualquier versión de Windows, manteniendo el lenguaje de programación C# y las librerías de Net Framework. En futuras iteraciones del proyecto cuando se desee pasar el código a dispositivos móviles se tendrá en cuenta la portabilidad a Xamarin, pasando la capa de presentación y la de comunicación a este sistema.

En comparación Java facilita muchísimo más el desarrollo multiplataforma, ya que se puede usar el mismo código, con ciertas adaptaciones, en Windows, Mac y Linux. Sin embargo, se tratar de estudiar una nueva tecnología que no se haya visto en el grado, por lo que nos decantamos por el uso de Net Framework.

NET FRAMEWORK

Net framework es un framework de desarrollo de software que nos permite la ejecución de aplicaciones en un entorno de ejecución limitado, denominado CLR, el cual nos provee de servicios de seguridad, gestión de memoria, manejo de excepciones.

Al ser un framework disponemos de un conjunto de librerías, denominado FCL que nos provee de diversas funcionalidades como interfaz de usuario, acceso a datos, conectividad de bases de datos, comunicaciones de red, criptografía y desarrollo de aplicaciones web.

WINDOWS COMMUNICATION FOUNDATION

WCF es una API que permite a una aplicación establecer objetos que pueden estar disponibles desde diferentes dominios, procesos, o incluso desde diferentes ordenadores.

WCF facilita el desarrollo de aplicaciones distribuidas, abstrayendo al desarrollador de las particularidades de la comunicación entre el cliente y el servidor. De esta forma, en el lado del cliente cualquier petición al objeto remoto se canaliza a través de un objeto de tipo canal, en el cual se encapsula el modo de transporte actual, ya sea una conexión por TCP, por UDP, o por HTTP.

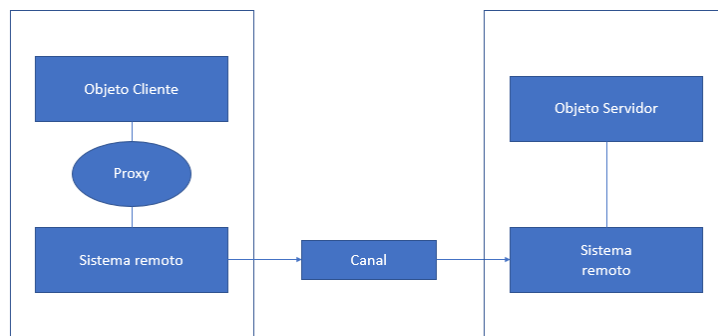


Figura 14. Ejemplo de arquitectura WCF.

Una aplicación basada en WCF puede soportar diversos protocolos sin necesidad de recompilar el programa. Sin embargo, es necesario que tanto el cliente como el servidor estén configurados en esos modos de funcionamiento.

WCF es el equivalente al uso de la tecnología RMI en Java que nos permite realizar llamadas a funciones remotas como si fuesen locales.

VISUAL STUDIO

Visual Studio es un entorno de desarrollo integrado, o IDE, desarrollado por Microsoft para sistemas operativos Windows. La primera versión de Visual Studio se comercializó en 1997, aunque con anterioridad ya se habían comercializado versiones previas de Visual Basic, Visual FoxPro y de Visual C++. La primera versión con Net Framework fue Visual Studio .Net 2002, la cual incluía Net Framework 1.0.

Este IDE no soporta de manera intrínseca un lenguaje determinado de programación, ya que permite añadir lenguajes o soluciones de programación por medio del desarrollo de paquetes. De esta forma, permite el desarrollo en múltiples lenguajes de programación, como C#, C++, Visual Basic.NET, F#, Python, Ruby on Rails, e incluso Java.

Desde la versión 2013 Visual Studio dispone de una versión gratuita conocida como *Community* que ofrece las mismas capacidades de la versión comercial, pero limitando su uso a empresas de pequeño tamaño, desarrolladores de software libre y estudiantes. La versión que se ha utilizado para este proyecto es la *Community 2017*, la cual permite usar trabajar con las versiones de Net Framework comprendidas entre la 2.0 y la 4.5.2.

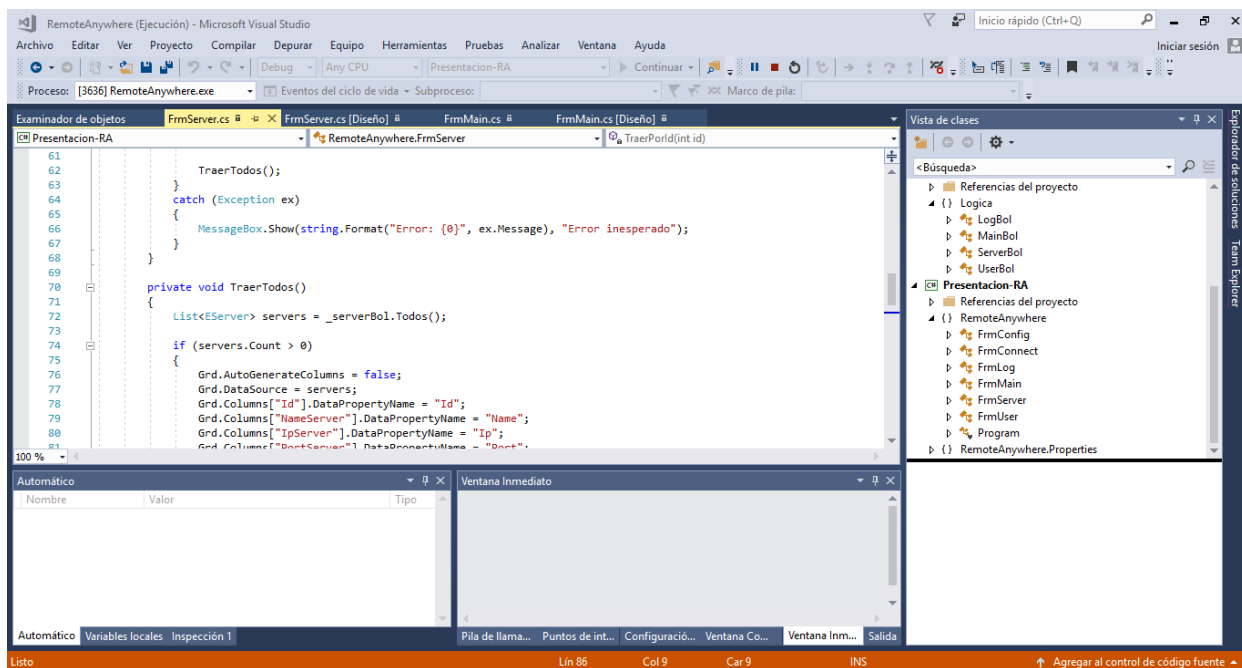


Figura 15. Captura del entorno de desarrollo usado.

BASES DE DATOS

Este tipo de aplicación requiere una base de datos local de pequeño tamaño. Solo es necesario almacenar datos de servidores, de usuarios que tienen permiso de acceso, de accesos de usuarios y de configuración del servidor. Por ello, se puede utilizar bases de datos que permiten conexiones únicas como:

- SQLite. Desarrollada por Richard Hipp, es ampliamente utilizada como almacenamiento local en aplicaciones móviles. Esta base de datos está basada en un archivo, el cual es accesible mediante un conector ADO.NET.
- Sql Server Compact. Desarrollada por Microsoft para dispositivos móviles. Es una base de datos que esta almacenada en un archivo *sdf*, cuya conexión se realiza a través de ADO.NET.

Ambas bases de datos son portables. El único requisito es incluir las referencias de acceso a sus conectores en el proyecto. La aplicación revisa cada vez que se inicia si existe una base de datos, si no es así crea la base de datos y las tablas que son necesarias.

El acceso a la base de datos se realiza desde la capa de acceso de datos. Esto nos permite independizar las funcionalidades de creación, edición y borrado de datos de la estructura lógica del programa.

3.2. CAPA DE ACCESO DE DATOS

La capa de acceso de datos nos permite acceder directamente a la base de datos. Es también la capa donde se definen las entidades de relación que forman parte de la aplicación, este es el listado de entidades que manejamos, el cual se vio anteriormente en el diagrama de clases.

- EServer. Es el modelo de datos que define cada servidor.
- ELog. Es el modelo de datos para el registro de accesos.
- EUser. Es el modelo de datos para los usuarios que se autentifican.
- EConfig. Es el modelo de datos para la configuración de la aplicación. Nos permite configurar la aplicación en modo cliente o en modo servidor.

```
namespace DAL
{
    public class EServer
    {
        public int Id { get; set; }
        public string Name { get; set; }
        public string Ip { get; set; }
        public string Port { get; set; }
    }

    public class ELog
    {
        public int Id { get; set; }
        public string Description { get; set; }
        public string DateInserted { get; set; }
    }

    public class EUser
    {
        public int Id { get; set; }
        public string User { get; set; }
        public string Password { get; set; }
    }

    public class EConfig
    {
        public int Type { get; set; }
        //0 means mode Client-Server
        //1 means mode Server only
        //2 means mode Client only
        public string Port { get; set; }
    }
}
```

Cada entidad tiene asociada una clase donde se definen los métodos que hacen uso de la misma. Desde esas clases se puede crear la base de datos, y también insertar, editar y borrar esos datos. A continuación, vamos a ver como se realizan estos métodos.

El método de creación de la base de datos se ejecuta únicamente si no existe la base de datos, revisa si no existe el archivo, lo crea, y crea las tablas y los índices de las entidades.

```
public void crearBBDD()
{
    string fileName = "Database.sdf";
    if (File.Exists(fileName))
    {
        File.Delete(fileName);
    }
    //Crea la BBDD de Sql Compact con la tabla necesarias para el prototipo
    SqlCeEngine en = new SqlCeEngine(strConCE);
    en.CreateDatabase();
    string sSql = "";
    sSql = "CREATE TABLE[Config] ([Port] nvarchar(5) NOT NULL, [Type] int
NOT NULL)";
    EjecutarComandoCE(sSql, "");
    sSql="CREATE TABLE[Logs] ([Id] int IDENTITY(1,1) NOT NULL, [Description]
nvarchar(1000) NOT NULL, [DateInserted] nvarchar(25) NULL)";
    EjecutarComandoCE(sSql, "");
    sSql = "ALTER TABLE[Logs] ADD CONSTRAINT[PK_Logs] PRIMARY KEY([Id])";
    EjecutarComandoCE(sSql, "");
    sSql = "CREATE TABLE[Servers] ([Id] int IDENTITY(1,1) NOT NULL, [Name]
nvarchar(100) NOT NULL, [Ip] nvarchar(100) NOT NULL, [Port] int NOT NULL)";
    EjecutarComandoCE(sSql, "");
    sSql = "ALTER TABLE[Servers] ADD CONSTRAINT[PK_Server] PRIMARY
KEY([Id])";
    EjecutarComandoCE(sSql, "");
    sSql = "CREATE TABLE[Users] ([Id] int IDENTITY(1,1) NOT NULL, [Name]
nvarchar(100) NOT NULL, [Password] nvarchar(100) NOT NULL)";
    EjecutarComandoCE(sSql, "");
    sSql = "ALTER TABLE[Users] ADD CONSTRAINT[PK_Users] PRIMARY KEY([Id])";
    EjecutarComandoCE(sSql, "");
}
}
```

El método de insertar recibe la entidad que corresponde, abre una conexión a la base de datos, y manda una consulta de sql parametrizada por cada uno de los parámetros que conforma la entidad

```
public void Insert(EUser user)
{
    //Creamos nuestro objeto de conexion
    using (SqlCeConnection cnx = new SqlCeConnection(strcon))
    {
        try
        {
            cnx.Open();
            //Declaramos nuestra consulta de Acción Sql parametrizada
            const string sqlQuery ="INSERT INTO Users (Name, Password)
VALUES (@user, @password)";
            using (SqlCeCommand cmd = new SqlCeCommand(sqlQuery, cnx))
            {
                cmd.Parameters.AddWithValue("@user", user.User);
                cmd.Parameters.AddWithValue("@password", user.Password);
                cmd.ExecuteNonQuery();
            }
        }
        catch (SqlCeException e) { Console.WriteLine(e); }
        cnx.Close();}
}
```

3.3. CAPA DE LÓGICA DE NEGOCIOS

La capa de lógica de negocio en este proyecto realiza las funcionalidades de la aplicación que no estén directamente relacionadas con la comunicación. Esto nos permite independizar el método de comunicación de la plataforma.

En la práctica, esto implica que esta capa realiza las siguientes funciones:

- Es intermediaria entre la capa de presentación y la capa de datos. Esto implica que la capa de presentación le pasara una entidad y esta capa debe revisar si debe realizar una solicitud de inserción o una solicitud de actualización.
- Realiza funciones auxiliares como detección de la ip interna del servidor, detección de la ip externa.
- Revisa cual es el rendimiento del sistema solicitando datos sobre el porcentaje de CPU y la cantidad de memoria que usa el proceso actual. Estos datos son solicitados mediante un *thread* o instancia que se ejecuta en paralelo, para evitar que el programa se bloquee.

En el siguiente código podemos ver como se solicita desde la capa de lógica de negocios la ip interna del servidor y la externa del mismo. Para poder establecer la ip externa se utiliza un servicio que nos devuelve, formateado en HTML, una cadena de texto con la ip.

```
public String dameIpExterna()
{
    string myIp = new
    WebClient().DownloadString(@"http://icanhazip.com").Trim();
    return myIp;
}

public String dameIpInterna()
{
    var host = Dns.GetHostEntry(Dns.GetHostName());
    foreach (var ip in host.AddressList)
    {
        if (ip.AddressFamily == AddressFamily.InterNetwork)
        {
            return ip.ToString();
        }
    }
    throw new Exception("Local IP Address Not Found!");
}
```

3.4. CAPA DE PRESENTACIÓN

La capa de presentación está realizada en *WinForms*. Es la API de acceso a elementos de la interface gráfica de Windows. Esta API fue portada mediante Mono para poder ser ejecutada bajo GNU/Linux y Mac, con lo que sería posible utilizar el programa utilizar en esos sistemas operativos. De hecho, las pruebas que se han realizado han permitido ejecutarlo en GNU/Linux

Sin embargo, no existe esta alternativa en Android y en iOS. Para estos sistemas la opción que tenemos es utilizar Xamarin, el cual utiliza otra API conocida como Xamarin Forms. De los motivos por lo que no se usa Xamarin ya hemos hablado anteriormente, básicamente no se puede utilizar para Windows 7 y 8, que es precisamente la plataforma en la que queremos obtener el entregable final.

Si se quiere ampliar el acceso a estos dispositivos, en desarrollos futuros, se puede utilizar Xamarin, o bien independizar del sistema operativo la capa de acceso mediante una capa de presentación multiplataforma como *Cordova* o *PhoneGap*.

El desarrollo en Visual Studio de las pantallas *WinForms* es muy sencillo, existe un editor visual que nos permite añadir los componentes necesarios, tal y como podemos ver en la siguiente figura.

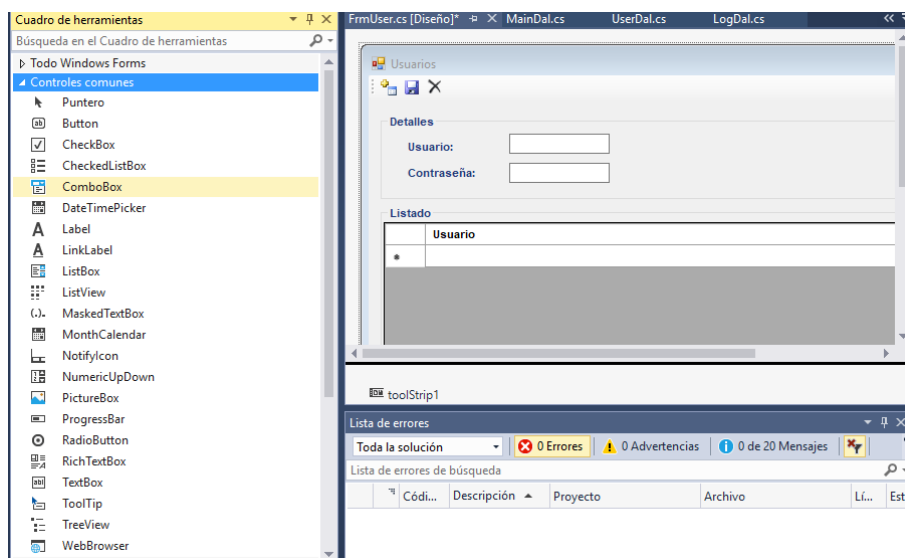


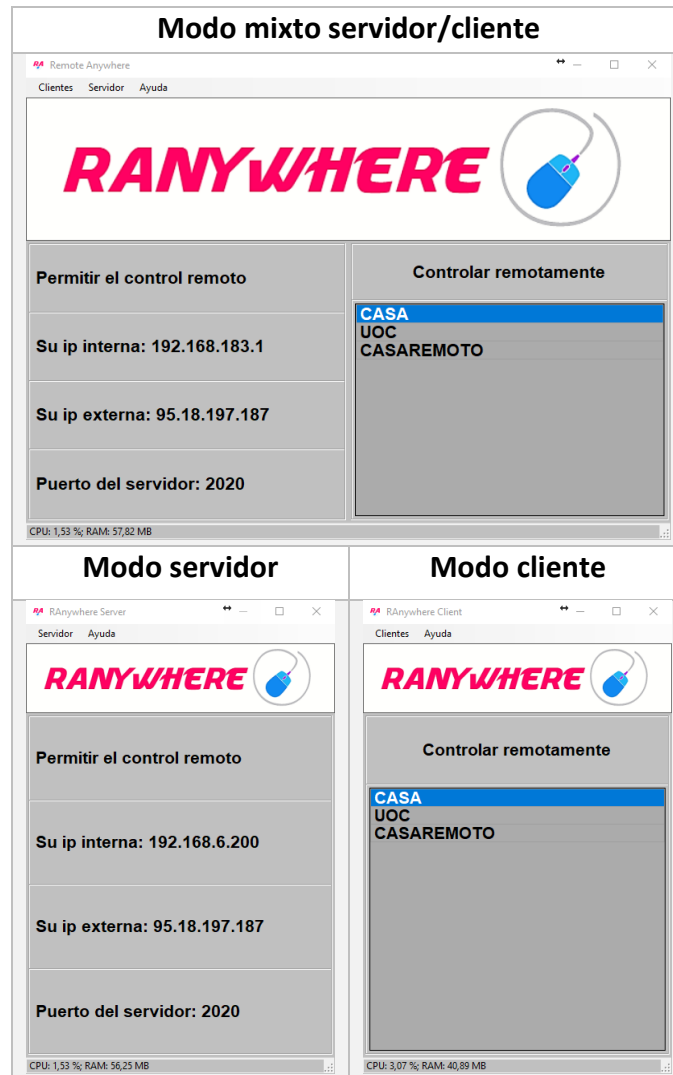
Figura 16. Desarrollo visual del formulario de usuarios.

La capa de presentación tiene los siguientes formularios:

- *FrmConfig*. En este formulario se configura el puerto del servidor y el modo de la aplicación. La aplicación puede funcionar en tres modos: en modo cliente solo permite conectarse a servidores, en modo servidor permite recibir conexiones remotas y en modo cliente/servidor permite ambos modos. Dependiendo del modo que se escoja

cambia la interfaz gráfica y los menús a los que se puede acceder, para ello es necesario reiniciar la aplicación tras activarlo.

- FrmMain. Es el formulario principal, el cual arrancara en el modo en el que se haya configurado.



- FrmLog. Es el formulario en el que se pueden visualizar los registros de acceso. Solo se puede acceder desde el modo servidor.
- FrmUser. Es el formulario en el que se pueden insertar, actualizar y borrar los usuarios que tienen acceso al servidor. Solo se puede acceder desde el modo servidor.
- FrmServer. Es el formulario en el que se pueden insertar, actualizar y borrar los servidores a los que accede el cliente mediante un formulario tipo *CRUD*. Solo se puede acceder desde el modo cliente. En la siguiente figura podemos ver dicho formulario.

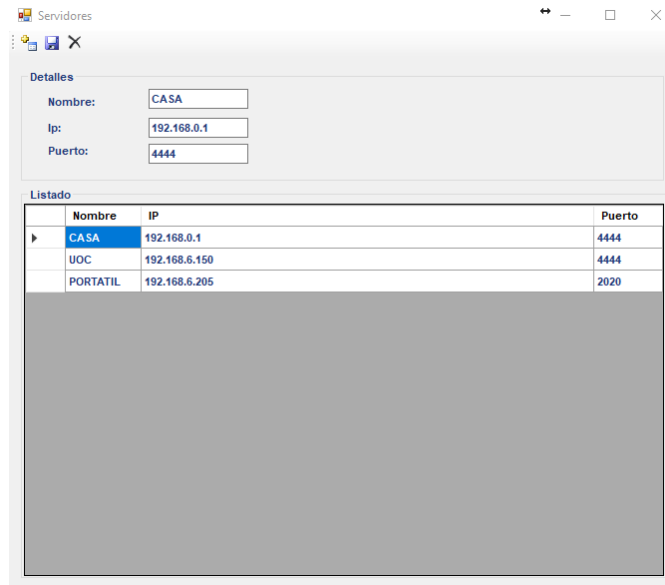


Figura 17. Formulario de edición de servidores.

- FrmConnect. Es un formulario que nos permite la conexión directa a un servidor o el guardado del mismo en el listado de servidores. Solo se puede acceder desde el modo cliente.



Figura 18. Formulario de conexión.

- FrmViewer. Este formulario nos permite visualizar la imagen del servidor remoto. Desde este formulario se solicita el usuario y la contraseña para acceder remotamente y se realiza la petición al servidor.

Si el usuario y la contraseña son validados se modifica el tamaño de este formulario adaptándose a la resolución de la pantalla del servidor. Las imágenes se reciben mediante eventos y se envían al control *PictureBox* del formulario. En la siguiente imagen podemos ver una captura de ese formulario realizada usando como cliente y servidor el mismo equipo, donde podemos ver como se produce un efecto espejo.

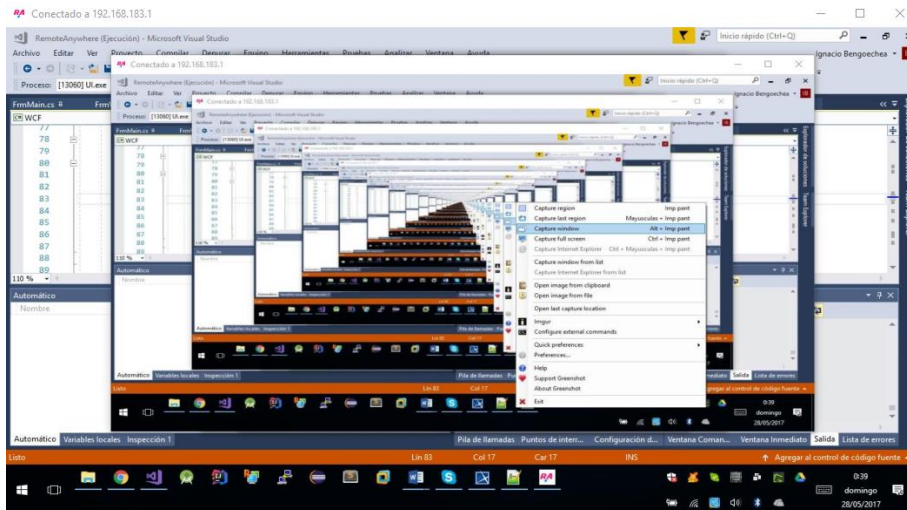


Figura 18. Conexión cliente/servidor desde el mismo ordenador.

Las pulsaciones del teclado, y las pulsaciones y movimientos del ratón son enviados mediante la capa de comunicación utilizando eventos.

En el caso del ratón se indica si hemos pulsado o liberado uno de los dos botones, o si lo hemos movido, en cualquiera de estos dos casos se debe determinar la posición de nuestro formulario donde se ha producido ese evento. Esas coordenadas deben ser trasladadas a la resolución real del servidor mediante un escalado. En la siguiente figura vemos el código relacionado.

```
public void sendMouseMove(int x, int y)
{
    if (connected)
    {
        try
        {
            svc.UpdateCursorMove(x, y);
        }
        catch (Exception ex)
        {
            Console.WriteLine("MouseMove exception: " + ex.Message);
        }
    }
}

public void sendMouseEvent(Win32Stuff.MouseEvents mouseEvent, int x, int y)
{
    if (connected)
    {
        try
        {
            svc.UpdateCursorEvent(mouseEvent, x, y);
        }
        catch (Exception ex)
        {
            Console.WriteLine("MouseMove exception: " + ex.Message);
        }
    }
}
```

En el servidor esta información se recupera y se usa la API de Windows para realizar el evento que corresponda, ya sea una pulsación del ratón o un movimiento. En el caso del teclado indicamos si la tecla ha sido pulsada o liberada, y la tecla que se ha pulsado. Esa información viaja mediante la capa de conexión hasta el servidor, donde se utiliza una API que nos permite simular pulsaciones del teclado.

Uso de *threads* y actualización en *WinForms*

Una característica peculiar de *WinForms* es que la interface no puede ser actualizada directamente desde un *thread*, ya que existe un controlador de acceso concurrente que impide la modificación de la interface de usuario para evitar problemas de concurrencia.

En este caso la opción que se maneja es informar al controlador de que existe un control delegado que debe ser ejecutado en el subproceso que corresponde al identificador de la ventana o del componente visual que se especifica.

En la siguiente porción de código podemos ver el código que se usa para actualizar el formulario de visualización en el cliente de la aplicación. Existen tres procedimientos delegados que podemos identificar con el termino *Invoke*.

En el primero se actualiza el encabezado del formulario, indicando si hemos conectado al servidor. En el segundo se actualiza el tamaño de la pantalla del cliente, ajustando la anchura y la altura de la misma en función de la resolución de la pantalla del servidor. En el tercero se recibe la imagen del servidor, mediante el uso de un evento que nos notifica si se ha recibido una imagen nueva, y se envía a un componente *PictureBox* que permite visualizarla en tiempo real gracias al control delegado.

```
private void clientScreenUpdated(Image salida)
{
    float ratio;
    if (!bConnected)
    {
        this.PConexion.Invoke((MethodInvoker)delegate
        {
            PConexion.Visible = false;
            this.Text = "Conectado a " + sIp;
            bConnected = true;
        });
        this.Invoke((MethodInvoker)delegate
        {
            if ((salida.Width != 0) && (salida.Height != 0) &&
(salida.Width!=outWidth) && (salida.Height!=outHeight))
            {
                outWidth = salida.Width;
                outHeight = salida.Height;
                ratio = (float)this.Height / ((float)salida.Height /
(float)salida.Width);
                this.Width = (int)(ratio);
                this.CenterToScreen();
                Console.WriteLine("UI updated:" + this.Width + " width, " +
this.Height + " height, " + ratio + " ratio");
            }
        });
        this.Imagen.Invoke((MethodInvoker)delegate
        {
            this.Imagen.Image = salida;
            Console.WriteLine("Image updated:" + salida.Width + " width,
" + salida.Height + " height, ");});});
```

Esto implica que el código que nos genera el evento de recepción de la imagen esta encapsulado en un *thread*. El mecanismo de recepción de imágenes está continuamente preguntando al servidor si tiene una nueva imagen, esto se debe a la configuración actual que se ha usado en WCF, que no nos permite recibir un evento en caso de recepción de imágenes.

Para evitar que este mecanismo sature la CPU del dispositivo la mejor solución es utilizarlo en un *thread* con baja prioridad. Cuando el *thread* recibe una nueva imagen envía un evento asíncrono al formulario del cliente y le notifica de que debe actualizar la imagen, como se ve en el código anterior.

3.5. CAPA DE COMUNICACIÓN

La capa de comunicación utiliza un servicio WCF para realizar la comunicación entre el cliente y el servidor. Para utilizar este servicio se debe indicar:

- El protocolo de comunicación. Podemos usar tres opciones. TCP, UDP, o HTTP. En nuestro hemos usado TCP porque nos permite una comunicación bidireccional, y además no tiene los requisitos de seguridad del protocolo HTTP, que requiere que registremos la dirección que vamos a usar o que usemos en modo administrador el programa. Más concretamente, el nombre de la opción es NetTcpBinding.
- El puerto y la ip de escucha, tanto en el servidor como en el cliente.
- La longitud máxima del paquete que se va a enviar. En WCF el tamaño por defecto es de 65536 bytes, el cual es fácil de superar al enviar imágenes, por lo que se ha ampliado a 2147483646 bytes, lo cual son cerca de 2MBytes.
- El método de compresión dinámica que vamos a usar. Tenemos 2 opciones que veremos posteriormente: Deflate y GZip.

En la siguiente porción de código podemos ver como se inicia la conexión WCF desde el cliente.

```
public void Start()
{ try
  { address = new EndpointAddress(new Uri("net.tcp://" + sIp + ":" + sPort
+ "/RemoteService"));
    binding = new NetTcpBinding();
    binding.MaxReceivedMessageSize = 2147483646;
    factory = new ChannelFactory<WCFServer.IRemoteService>(binding,
address);
    svc = factory.CreateChannel();
    connected=svc.SendLogin(sUser, sPassword);
    if (connected) { updateScreen();}
    else { Stop();}
  }
  catch (Exception e)
  { connected = false;
    MessageBox.Show(e.Message);
  }
}
```

Todos estos parámetros deben ser idénticos tanto en el cliente como en el servidor, sino no se establecerá la conexión. WCF generara una comunicación entre ambos extremos y permite que el cliente solicite métodos al servidor. En estas llamadas se pueden enviar y recibir información a través de los parámetros y a través de las salidas de las funciones.

Para ello, es necesario que se defina una interface con las operaciones que se utilizan en la comunicación. En nuestro caso son las siguientes.

- UpdateScreenImage. La inicia el cliente, y se le devuelva un array de bytes con la imagen del servidor codificada y comprimida en JPEG.
- UpdateCursorMove. La inicia el cliente, y envía al servidor las coordenadas escaladas de la posición del ratón en la imagen que muestra el cliente.
- UpdateCursorEvent. La inicia el cliente, y envía la pulsación o liberación del botón izquierdo o derecho del ratón y la posición del mismo.
- UpdateKeyboardEvent. La inicia el cliente, y envía la tecla que se ha pulsado o liberado en el teclado.
- SendLogin. La inicia el cliente, y envía el nombre y la contraseña del usuario que trata de autenticarse en el servidor. El servidor indica si se valida o no. En caso de que no se valida se anula la conexión.

Este es el código que establece la interface.

```
[ServiceContract()]
public interface IRemoteService
{
    // Capture the screen data.
    [OperationContract]
    byte[] UpdateScreenImage();

    // Capture the cursor data.
    [OperationContract]
    void UpdateCursorMove(int x, int y);

    // Capture the cursor event data.
    [OperationContract]
    void UpdateCursorEvent(Win32Stuff.MouseEvents mouseEvent, int x, int y);

    // Capture the keyboard event data.
    [OperationContract]
    void UpdateKeyboardEvent(Win32Stuff.KeyboardEvents keyEvent,
    virtualKeyCode vkcode);

    // Login
    [OperationContract]
    bool SendLogin(string user, string password);
}
```

Las clases que conforman la capa de conexión son las siguientes:

- IServiceContainer. Es la interface sobre el que se definen las operaciones remotas que se pueden realizar sobre el servidor.
- ServiceContainer. Es la implementación de la interface sobre la que se definen las

operaciones remotas. Es en esta clase donde se estructura toda la lógica del servidor WCF de la aplicación. Tiene varias funciones:

- Crea un servicio que está a la escucha de posibles peticiones, en la ip interna de la aplicación y en el puerto que se ha configurado.
- Cada vez que se abre un canal espera a recibir un comando de autenticación del mismo. En caso de que la autenticación sea válida el canal se mantiene, sino se cierra.
- Cuando recibe un comando para enviar la imagen del servidor realiza una captura de la pantalla mediante la clase *ScreenCapture*. Esa imagen capturada en JPEG se convierte a un array de bytes y se envía por el canal de comunicación.
- Cuando recibe un comando para recrear el evento o el movimiento de un ratón utiliza la API de Windows para generar ese evento.
- Cuando recibe un comando para recrear el evento de un teclado utiliza una API externa llamada *InputSimulator* para implementarlo.
- Si deja de recibir información por el canal se cierra automáticamente.
- ClientThread. Esta es la clase del cliente que se conecta al servicio WCF. Esta clase tiene varias funciones:
 - Establece el canal de comunicación por el que se envían comandos y datos.
 - Realiza el proceso de login y desconecta la conexión en caso de que no sea válido.
 - Le envía continuamente al servidor el comando para recibir imágenes remotas. Al ser un proceso de interrogación continuo puede incrementar el uso de la CPU, por ello esta clase se ejecuta en un *thread* independiente de la misma, con baja prioridad, de forma que no afecte al uso de la CPU.
 - Cada vez que se produce en el cliente, sobre la imagen que representa al servidor, un evento el teclado o en el ratón se envía al servidor la información de dicho evento para replicarlo.
 - Cuando se cierra la pantalla de visor del cliente se cierra el canal de comunicación.
- ScreenCapture. Se encarga de capturar la imagen de la pantalla, y de convertirla a un array de bytes que se envía a través del canal que ha creado el servicio WCF.
- MyServiceHost. Es la clase desde la que se lanza el servicio WCF. En caso de que se cierre la aplicación de servidor se solicita que la clase se cierre para no utilizar recursos y cerrar realmente la aplicación.
- GDIStuff. Es un *wrapper*, es una función que encapsula a otra. En C# se utilizan para poder encapsular funciones de librerías de otros lenguajes, normalmente C o C++. En este caso se usa para poder usar llamadas al sistema que permiten capturar la pantalla. Es usada por *ScreenCapture*
- Utils. Es usada como modulo para albergar funciones auxiliares.
- WIN32Stuff. Es otro *wrapper* que se usa para encapsular llamadas al sistema relacionadas con los eventos del teclado y del ratón.

CALIDAD DE IMAGEN

Existen dos sistemas de compresión que debemos tener en cuenta. Por un lado, tenemos el sistema de compresión sin pérdidas es aquel en el que permite minimizar la cantidad de información enviada, utilizando un espacio menor, siendo posible recuperar la información original. Se utiliza para comprimir datos que no deben ser degradados, como documentos, imágenes o sonidos. Ejemplos de imágenes con este sistema de compresión son las que usan el formato PNG o RLE.

Por otro lado, tenemos el sistema de compresión con pérdidas, el cual nos permite enviar minimizar la información utilizado usando una aproximación, con una cantidad reducida de datos, que no nos permite reconstruir exactamente los datos originales. El formato PNG es un ejemplo formato que usa este sistema.

En esta aplicación el uso de sistemas de compresión con pérdidas es adecuado y recomendado, ya que no es necesario reconstruir la imagen original de manera exacta, por lo que podemos usar JPEG o cualquier variante. En el caso de JPEG tenemos un parámetro denominado *compression quality* que nos permite ajustar la compresión de la imagen resultante.

Podemos ver un ejemplo en las siguientes imágenes, donde comparamos una imagen comprimida en JPEG, con un 50% de *compression quality*, la cual ocupa 100 KB, contra otra imagen comprimida en PNG, la cual ocupa 513 KB.

Ambas imágenes tienen una calidad similar. Es necesario ampliarlas bastante para poder apreciar artefactos en la imagen comprimida en JPEG, debidos a la compresión con pérdidas. En cualquier caso, el objetivo de la aplicación queda cubierto con el uso del formato JPEG.



Figura 19. Imagen comprimida en JPEG. Tamaño 100 KB. Fuente uoc.edu.

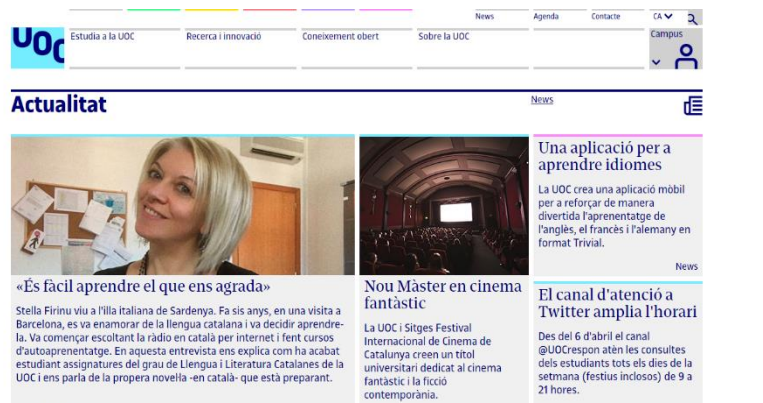


Figura 20. Imagen comprimida en PNG. Tamaño 513 KB. Fuente uoc.edu.

COMPRESIÓN DE DATOS

Los sistemas de compresión de datos nos permiten minimizar las necesidades de ancho de banda requeridas para el envío de imágenes. Existen varias opciones de configuración de WCF que nos permite seleccionar el tipo de compresión de datos que queremos usar:

- *CompressionFormat.Deflate*. Es una librería que nos permite aplicar el algoritmo Deflate, el cual usa una combinación del algoritmo LZ77 y de codificación Huffman. Se utilizó y se patentó para desarrollar PKZIP, y se permite usar en aquellas maneras que no están cubiertas por la patente. Por lo que se puede usar para mandar datos comprimidos por internet.
- *CompressionFormat.GZip*. Es una librería que nos permite comprimir un stream mediante el formato GZip, el cual se basa parcialmente en Deflate mejorando el rendimiento del mismo.

Al seleccionar una de las dos opciones el intercambio de mensajes entre cliente y servidor se realiza usando la opción seleccionada. En nuestro caso dejamos configurada por defecto la opción GZip, ya que es una extensión de Deflate.

ENCRIPCIÓN

Para poder asegurar las encriptaciones existen varias opciones de configuración en WCF que nos permiten establecer el tipo de seguridad que se desea usar

- Transporte. En este tipo se encripta la comunicación del canal de transporte. Si se usase HTTP se usaría SSL para asegurarlo.
- Mensaje. En este tipo se encripta la información de cada mensaje que se envía.

En el modo de comunicación que estamos utilizando, NetTCPBinding, solo se permite que usemos la encriptación por mensaje la cual se deja configurada por defecto.

CAPÍTULO 4. PRUEBAS DE USO Y DE RENDIMIENTO

La realización de pruebas nos permite estudiar y determinar el correcto funcionamiento de la solución, o de determinar los posibles fallos sobre la misma. Para poder establecer las pruebas que se deben realizar se ha determinado una directiva de pruebas.

- Cumplimiento de los requisitos y funcionalidades establecidos para la solución.
- Revisión de las buenas prácticas en el desarrollo del software, no son exigidas en los requisitos, pero son de uso habitual.
- Desarrollo de las pruebas según los módulos que se han planteado en la aplicación.
- Valoración general de las pruebas y del proceso de las mismas.

Indicador	Capa	Estado
Creación de usuarios en la base de datos	Acceso a datos	OK
Edición de usuarios en la base de datos	Acceso a datos	OK
Eliminación de usuarios en la base de datos	Acceso a datos	OK
Creación de servidores en la base de datos	Acceso a datos	OK
Edición de servidores en la base de datos	Acceso a datos	OK
Eliminación de servidores en la base de datos	Acceso a datos	OK
Guardado de la configuración en la base de datos	Acceso a datos	OK
Modo servidor solo al arrancar	Lógica de Negocios	OK
Modo cliente solo al arrancar	Lógica de Negocios	OK
Modo mixto cliente/servidor al arrancar	Lógica de Negocios	OK
Solicitud de recepción de imágenes de servidor	Capa de conexión	OK
Respuesta con la imagen capturada del servidor	Capa de conexión	OK
Envío de eventos del teclado	Capa de conexión	OK
Recepción y generación de eventos del teclado	Capa de conexión	OK
Envío de eventos del ratón	Capa de conexión	OK
Recepción y generación de eventos del ratón	Capa de conexión	OK
Envío de movimientos del ratón	Capa de conexión	OK
Recepción y generación de movimientos del ratón	Capa de conexión	OK
Envío de usuario y contraseña	Capa de conexión	OK
Recepción de usuario y contraseña	Capa de conexión	OK
Desconexión si el usuario y contraseña no son validos	Capa de conexión	OK
Conexión si el usuario y la contraseña no son validos	Capa de conexión	OK
Desconexión del cliente en caso de caída del servidor	Capa de conexión	OK
Guardado del registro de conexión en el servidor	Capa de conexión	OK
Guardado del registro de desconexión en el servidor	Capa de conexión	OK
Guardado del registro de fallo de autenticación en el servidor	Capa de conexión	OK

PRUEBAS DE RENDIMIENTO

Con respecto a las pruebas de rendimiento el funcionamiento de la aplicación se ha mostrado muy constante. Para poder establecer el rendimiento de la aplicación se ha creado en la capa de lógica de negocios una función que monitoriza el rendimiento de la CPU y de la memoria.

En el siguiente código podemos ver como se solicita los datos sobre la CPU y la memoria. Cada vez que se reciben datos de rendimiento se genera un evento que actualiza la capa de presentación. De esta forma, evitamos que se produzca un bloqueo en la interfaz gráfica.

```
public void GetCounter()
{
    String appName="UI";
    PerformanceCounter process_cpu = new PerformanceCounter("Process", "%
Processor Time", appName);
    PerformanceCounter ramCounter = new PerformanceCounter("Process",
"Working Set", appName);
    float cpu, ram;
    String salida;
    while (!(end))
    {
        cpu =process_cpu.NextValue();
        ram= ramCounter.NextValue();
        ram = ram / 1024 / 1024;
        ram = (float)(Math.Truncate((double)ram * 100.0) / 100.0);
        cpu =(float)(Math.Truncate((double)cpu * 100.0) / 100.0);
        salida= "CPU: "+(cpu)+" %; RAM: "+(ram)+" MB ";
        Thread.sleep(1000);
        if (cpuUpdated != null)
        {
            cpuUpdated(salida);
        }
    }
}
```

El uso de este código nos permite determinar qué porcentaje de CPU y de memoria se están consumiendo en cada momento. Esos valores se actualizan cada segundo y se muestran en la capa de presentación.

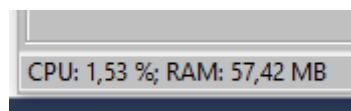


Figura 21. Rendimiento de la aplicación.

Durante el desarrollo de las pruebas se chequea estos valores y vemos que se mantienen constantes a lo largo de las mismas.

CAPÍTULO 5. CONCLUSIONES DEL PROYECTO

La realización de este proyecto ha sido muy estimulante. Desde el punto de vista del aprendizaje me ha permitido familiarizarme con técnicas de desarrollo en Net Framework y en Visual Studio. También me ha permitido desarrollar un proyecto distribuido con comunicación cliente/servidor.

Asimismo, es un proyecto que aún se puede ampliar y mejorar. El uso de la arquitectura basada en capas nos permite extenderlo a otras plataformas en un futuro próximo. Sin embargo, es en este punto en el de las posibles ampliaciones donde se han visto las limitaciones que tenemos al usar la tecnología de Net Framework.

Esta no es una solución multiplataforma. Un objetivo opcional del proyecto era realizar un cliente en Android o en iOS del mismo, para lo cual era necesario que la implementación en Xamarin para Android e iOS de WCF estuviera completado. Sin embargo, esta implementación aún se está realizando por el equipo de desarrollo de Xamarin, por lo que hay que esperar a que la tengan operativa para que se pueda llegar a desarrollar.

Con respecto a la planificación y la metodología, me gustaría indicar que la estructura de descomposición de trabajo ha sido muy útil para poder establecer posteriormente la integración del mismo. El seguimiento de la evaluación continua, la elaboración de informes de actualización para la consultora, María Isabel, y las respuestas que he recibido de ella han permitido el enfoque actual de este proyecto.

En un futuro se plantean líneas de trabajo que se deberán desarrollar en torno a esta solución, la más inmediata como he comentado anteriormente es el desarrollo de un cliente para Android e iOS. Pero también existen otras ideas que permiten aprovechar el potencial de WCF y combinarlo con nuevos servicios.

Una posibilidad que se plantea es el tratar de extender las limitaciones de la necesidad de redirección en los puertos, mediante el uso de una capa de comunicación basada en UDP.

CAPÍTULO 6. GLOSARIO

- Android: Sistema operativo móvil basado en el núcleo Linux.
- EDT: Estructura de descomposición del trabajo.
- Framework: Estructura conceptual y tecnológica de asistencia definida, normalmente, con artefactos o módulos concretos de software, que puede servir de base para la organización y desarrollo de software.
- HTML: Hace referencia al lenguaje de marcado para la elaboración de páginas web.
- iOS: Sistema operativo móvil de la multinacional Apple.
- JSON: Formato de texto ligero para el intercambio de datos.
- Mono: Implementación libre de NET Framework de Microsoft.
- Net Core: Framework de código libre desarrollado por Microsoft.
- Net Framework: Framework de Microsoft que hace un énfasis en la transparencia de redes, con independencia de plataforma de hardware y que permita un rápido desarrollo de aplicaciones.
- Open source: Software de código abierto.
- PictureBox: Control de WinForms que muestra gráficos en formato de mapa de bits, GIF, JPEG o icono.
- REST: Estilo de arquitectura software para sistemas hipermedia distribuidos
- SOAP: Protocolo estándar que define cómo dos objetos en diferentes procesos pueden comunicarse por medio de intercambio de datos XML
- SSL: Protocolos criptográfico que proporciona comunicaciones seguras por una red.
- Stream: Transmisión de datos.
- TCP: Protocolo de red de control de transporte.
- Thread: Es una tarea que puede ser ejecutada al mismo tiempo que otra tarea.
- UDP: Protocolo mínimo de transporte orientado a mensaje.
- WCF: Windows Communication Foundation. Permite usar objetos remotos.
- WinForm: Librería gráfica de Net Framework.
- Xamarin: Implementación libre de la plataforma de desarrollo .NET para dispositivos Android, iOS y GNU/Linux.
- XML: Lenguaje de marcado extensible.

CAPÍTULO 7. BIBLIOGRAFÍA

- **SHERIFF, Paul D.** (2006). *Fundamental of N-Tier with examples in VB.Net and C#*. PDSA Inc.
- **Esposito, Saltarello** (2015). *Microsoft NET: Architecting Applications for the Enterprise*, Second Edition. Microsoft Press.
- **Levinson, Jeff** (2003). *Building Client/Server Applications with VB .NET. An example-driven approach*. Apress.
- **Sharp, Jagger** (2002). *Microsoft Visual C# .Net step by step*. Microsoft Press.
- **Net Framework** (2017) [en línea]. Wikipedia.
[Consulta 10 de abril de 2017]
<https://en.wikipedia.org/wiki/.NET_Framework>
- **Deflate** (2016) [en línea] Wikipedia.
[Consulta 12 de abril de 2017]
<<https://en.wikipedia.org/wiki/DEFLATE>>
- **Accesing services using a WCF client** (2017). Microsoft.
[Consulta 16 de abril de 2017]
<<https://docs.microsoft.com/en-us/dotnet/framework/wcf/accessing-services-using-a-wcf-client>>
- **How to use a WCF client** (2017). Microsoft.
[Consulta 17 de abril de 2017]
< <https://docs.microsoft.com/en-us/dotnet/framework/wcf/how-to-use-a-wcf-client>>
- **Input Simulator** (2017). Codeplex.com.
[Consulta 10 de mayo de 2017]
<<http://inputsimulator.codeplex.com>>
- **Using and disposing of WCF clients** (2012). Abel.
[Consulta 18 de abril de 2017]
<<https://coding.abel.nu/2012/02/using-and-disposing-of-wcf-clients>>
- **Mouse events in C#** (2010). Microsoft.com
[Consulta 2 de mayo de 2016]
<[https://msdn.microsoft.com/es-es/library/ms171542\(v=vs.110\).aspx](https://msdn.microsoft.com/es-es/library/ms171542(v=vs.110).aspx)>
- **Reading JPEG into a byte array** (2013). Perpetual Enigma.
[Consulta 3 de mayo de 2016]
<<https://prateekvjoshi.com/2013/12/28/reading-jpeg-into-a-byte-array/>>
- **Consuming a WCF Web Service** (2016). Xamarin.
[Consulta 14 de mayo de 2016]
<<https://developer.xamarin.com/guides/xamarin-forms/cloud-services/consuming/wcf/>>

ANEXO. MANUAL DE INSTALACIÓN, USO Y COMPILACIÓN

En este anexo se indica los requisitos necesarios para poder utilizar el instalador del programa, las instrucciones para usar la aplicación y los pasos necesarios para poder realizar la compilación del programa,

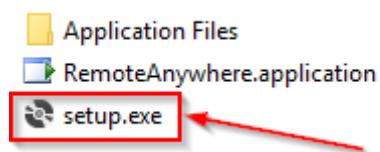
INSTALACIÓN DE LA APLICACIÓN

Es un requisito del programa que esté instalada la plataforma Net Framework versión 4.5.2. Si se usa el instalador entonces este comprobará si se ha instalado la plataforma o no, en cuyo caso la descargaría automáticamente de internet y la instalaría, solicitando confirmación del usuario.

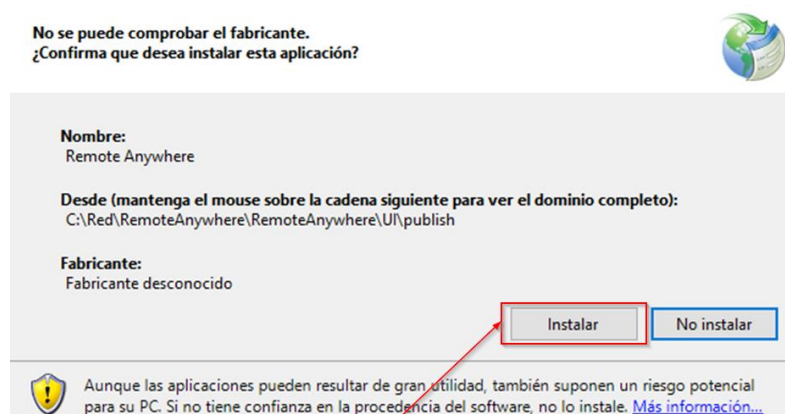
Si hubiese algún problema con la red que impidiera la descarga de esta plataforma se puede descargar e instalar manualmente desde esta dirección.

<https://www.microsoft.com/es-es/download/details.aspx?id=42642>

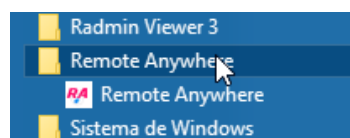
El instalador está incluido en la carpeta *inst* y su ejecutable es el archivo setup.exe.



Al pulsar el archivo setup.exe el proceso de instalación nos pide confirmación.

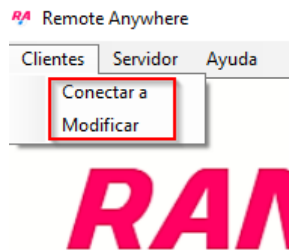


Tras instalarlo, el programa se ejecuta y además queda instalado en las carpetas del menú inicio.

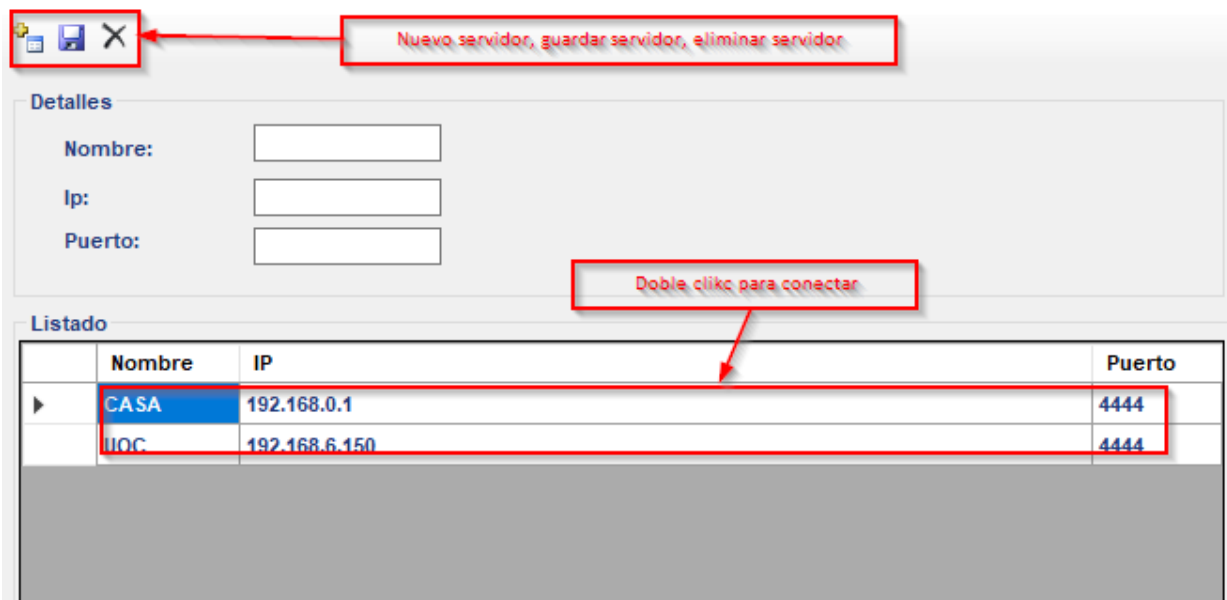


USO DE LA APLICACIÓN

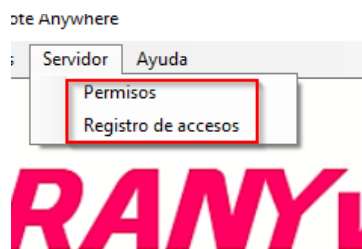
Desde el menú “Cliente” tenemos la opción de conectarnos a un servidor o de modificar el listado de servidores.



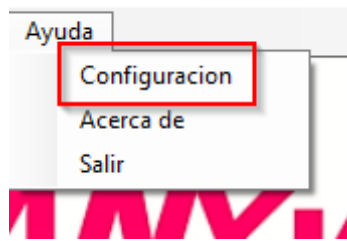
Desde la opción de “Modificar” podemos dar de alta un servidor, editarlo, eliminarlo, o conectarnos a ese servidor.



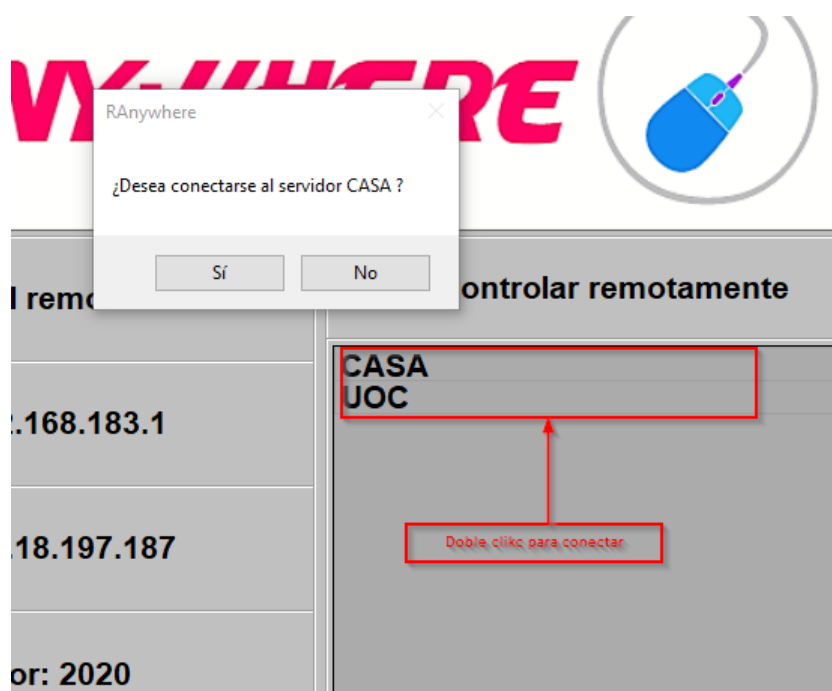
Desde el menú “Servidor” tenemos la opción de editar los permisos de los usuarios y de ver los registros de acceso.



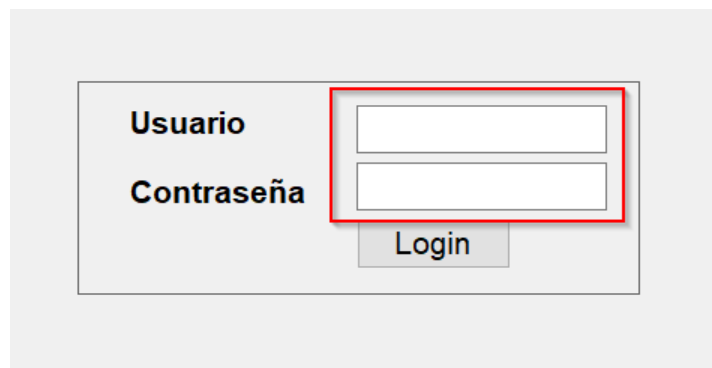
Desde el menú “Ayuda” tenemos la opción de configurar el modo de la aplicación, que puede ser mixto, solo servidor o solo cliente, también podemos indicar el puerto.



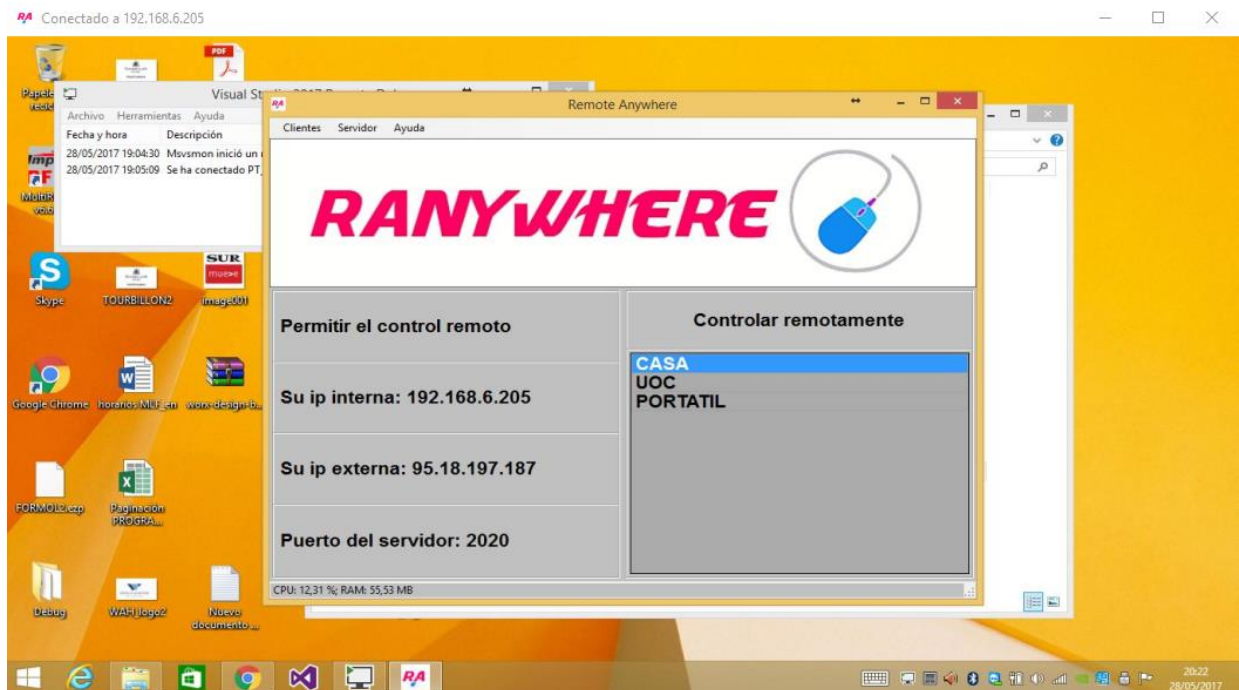
Desde la pantalla principal podemos acceder a los servidores que se han dado de alta previamente, mediante doble click.



Al intentar entrar en un servidor remoto se nos solicitara el usuario y la contraseña del mismo.



Una vez conectados podemos manejar remotamente el ordenador. El formulario del visor se adaptará a la resolución del servidor remoto para que la imagen no se vea distorsionada.



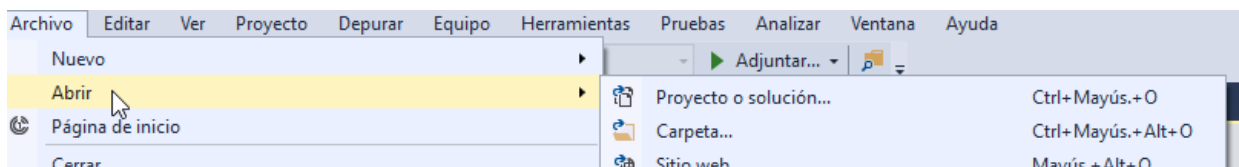
INSTRUCCIONES DE COMPILACIÓN

El programa ha sido realizado mediante el lenguaje C# en Visual Studio. Es necesario un ordenador con Windows 7 o superior para poder compilarlo y usarlo.

Código fuente. El código fuente del desarrollo está en la carpeta *src*. Para poder visualizarlo se recomienda *Visual Studio 2017 Community Edition*, la cual se puede descargar gratuitamente desde esta dirección:

<https://www.visualstudio.com/es/downloads>

En Visual Studio vamos al menú “Archivo” y seleccionamos la opción “Abrir/Proyecto o solución...”



La solución se cargará y podremos ver los cuatro proyectos que la componen, uno por cada capa:

- UI. Es la capa de presentación y el proyecto principal. Genera un archivo ejecutable llamado “RemoteAnywhere.exe”.
- BOL. Es la capa de lógica de negocios. Genera una librería con el nombre BOL.dll.

- DAL. Es el proyecto de la capa de accesos de datos. Genera una librería con el nombre DAL.dll.
- WCF. Es el proyecto de la capa de conexión. Genera una librería con el nombre WCF.dll que usamos como servicio.

Tras cargar la solución, la seleccionamos en el explorador de soluciones y pulsamos la combinación de teclas CTRL + F5 para iniciar la aplicación sin depurar.