



# **UNIVERSIDAD DE EXTREMADURA**

## **Escuela Politécnica**

### **Ingeniería Informática**

#### **Proyecto Fin de Carrera**

***“Sistema integral de control y videovigilancia  
remoto de viviendas bajo la plataforma .NET”***

**Autor: Alfredo Moreno Muñoz**

**Fdo.:**

**Autor: César Silgo Ortiz**

**Fdo.:**

**Director: Juan Hernández Núñez**

**Fdo.:**

#### **Tribunal calificador**

**Presidente:**

**Fdo.:**

**Secretario:**

**Fdo.:**

**Vocal:**

**Fdo.:**

**CALIFICACIÓN:**

**FECHA:**



## PRÓLOGO

Este proyecto se centra en dos de las áreas con mayor proyección en lo que a informática y bienestar se refiere: la domótica y la videovigilancia, tratadas desde el modelo de programación distribuida.

El objetivo principal es crear un conjunto de aplicaciones distribuidas que permitan el control y monitorización de forma remota de una instalación domótica junto a su sistema de videovigilancia. Es decir, una herramienta que permita al usuario encender y apagar electrodomésticos de su vivienda, verificar el estado de las luces de su vivienda, encender y apagar alarmas, comprobar si todas las puertas de su casa están cerradas, recibir avisos en caso de alarmas de seguridad, ver el estado de su vivienda mediante las cámaras de las que dispone... todo ello sin que el usuario tenga que estar físicamente en su vivienda, desde cualquier equipo conectado a Internet.

El software se desarrollará bajo los siguientes requerimientos:

- Debe permitir obtener toda la información posible acerca del estado de la instalación domótica, así como enviar órdenes a ésta.
- Debe desarrollarse mediante tecnologías de programación distribuida.
- Tiene que ser independiente de la instalación domótica sobre la que se ejecute la parte servidora.
- Debe de introducirse la licencia de las librerías Falcon de acceso al bus domótico.
- Debe garantizar la autenticación del usuario, es decir, debe asegurar que, cuando la vivienda recibe una orden, ésta proviene de un usuario autorizado y no de cualquier otro usuario.
- Tiene que permitir un mantenimiento de las cámaras presentes en la instalación domótica.
- Al ocurrir un evento detectado por las cámaras o instalación domótica, debe ser notificado al usuario de manera instantánea.

En base a estos requerimientos, al estado de la tecnología, tanto en el campo de la domótica como en el de videovigilancia y el de los componentes distribuidos, hemos elegido las siguientes tecnologías para el desarrollo del proyecto:

- EIB/KNX como instalación domótica. Este tipo de instalación es el estándar europeo con mayor futuro y además se adapta perfectamente a nuestras necesidades, debido a que permite control externo y a su estructura de sistema distribuido.
- ICE como middleware para sistemas distribuidos. ICE es la solución más flexible y ligera hoy en día en cuanto a sistemas distribuidos. Principalmente, proporciona una comunicación entre objetos transparente a la localización física de estos.
- Microsoft Visual Studio .NET. Plataforma de desarrollo, elaborada por Microsoft, que permite la conjunción de diferentes tecnologías de desarrollo, con la que ICE se integra a la perfección y con la que la instalación domótica obtiene mejores rendimientos en cuestiones de comunicación. Visual Studio .NET también aporta

muchas mejoras en lo que se refiere a facilidad de comunicación con cámaras de videovigilancia IP.

Basándonos en estos tres pilares se ha desarrollado todo el proyecto que se expone a continuación.

## ÍNDICE

ÍNDICE.....	5
TABLA DE ILUSTRACIONES .....	9
1    Introducción .....	11
1.1    Domótica .....	11
1.1.1    Áreas de aplicación.....	11
1.1.2    Componentes .....	12
1.1.3    Arquitecturas.....	12
1.1.4    Cableado de la instalación.....	13
1.1.5    Estándares domóticos .....	14
1.1.6    KNX/EIB .....	14
1.2    Videovigilancia .....	21
1.2.1    Introducción .....	21
1.2.2    Ventajas vídeo IP .....	22
1.3    Dispositivos móviles .....	24
1.4    Tecnologías distribuidas .....	25
1.4.1    Introducción .....	25
1.4.2    Características .....	25
1.4.3    Arquitectura Cliente/Servidor .....	26
2    Definición del problema .....	27
2.1    Sistema existente .....	27
2.1.1    Historia DomoUEX.....	27
2.1.2    Tecnologías.....	28
2.1.3    Análisis de debilidades .....	28
2.2    Especificación de requisitos .....	29
3    Estudio de viabilidad .....	31
3.1    Alternativas exploradas.....	31
3.1.1    OPC.....	31
3.1.2    MIDP.....	32
3.1.3    Java RMI .....	33
3.1.4    .NET .....	35
3.1.5    ICE.....	38
3.2    Toma de decisiones.....	38

3.2.1	Middleware .....	38
3.2.2	Plataforma de desarrollo.....	39
3.2.3	Instalación domótica .....	39
3.2.4	Videovigilancia .....	40
3.2.5	Dispositivos móviles .....	40
3.3	ICE.....	41
3.3.1	Introducción e historia .....	41
3.3.2	Arquitectura ICE .....	42
3.3.3	Servicios ICE.....	55
3.3.4	Beneficios de la arquitectura ICE .....	57
3.4	ICE-E .....	58
3.4.1	¿Qué es ICE-E?.....	58
3.4.2	Plataformas soportadas .....	59
3.4.3	Comparativa con ICE .....	60
3.5	ICE vs CORBA .....	61
3.5.1	Diferencias en el Modelo de Objetos.....	61
3.5.2	Diferencias en el soporte de Plataformas .....	62
3.5.3	Diferencias en la complejidad .....	63
4	Análisis y diseño .....	65
4.1	Mapa conceptual.....	65
4.2	Casos de uso.....	66
4.3	Arquitectura del sistema .....	68
4.3.1	Diagrama global.....	68
4.3.2	Componentes .....	69
4.3.3	Estructuras de datos.....	69
4.3.4	Relaciones .....	70
4.3.5	Datos persistentes.....	71
4.4	Diagrama de paquetes .....	73
4.5	Diagrama de clases.....	75
4.5.1	Servidor instalación .....	75
4.5.2	Servidor videovigilancia.....	78
4.5.3	Cliente .....	84
4.5.4	Cliente dispositivos móviles .....	88
4.5.5	DUConfig .....	91

4.6	Diagramas de secuencia .....	92
4.6.1	Acceder Instalación .....	92
4.6.2	Visualizar cámaras .....	92
4.6.3	Recibir evento instalación .....	93
4.6.4	Recibir alarma cámara .....	94
5	Manual de programador .....	95
5.1	Interfaces ICE .....	95
5.1.1	Servidor de instalación .....	95
5.1.2	Servidor de videovigilancia .....	96
5.1.3	Clientes .....	103
5.2	API .....	104
5.2.1	Ejemplo: DomoUEXCamera .....	105
6	Manual de usuario .....	108
6.1	Instalación .....	108
6.2	Configuración .....	108
6.3	Servidor instalación domótica .....	110
6.4	Servidor videovigilancia .....	111
6.4.1	Configuración .....	111
6.4.2	Datos persistentes .....	111
6.4.3	Interfaz .....	111
6.5	Cliente .....	113
6.5.1	Instalación domótica .....	113
6.5.2	Videovigilancia .....	114
6.6	Dispositivos móviles .....	115
7	De DomoUEX a QuercusDomoSystem .....	117
7.1	Arquitectura .....	117
7.2	Funcionalidades .....	118
8	Herramientas utilizadas .....	119
9	Trabajo futuro .....	120
10	Personal .....	121
10.1	Trabajo en equipo .....	121
10.2	Equipo .....	121
10.3	Investigación .....	121
10.1	Conocimientos adquiridos .....	122

10.2	Futuro .....	122
11	Agradecimientos .....	123
12	Referencias.....	124
ANEXO 1: ETS3.....		125
ANEXO 2: DIAGRAMA DE CLASES AMPLIADO .....		131



## TABLA DE ILUSTRACIONES

Ilustración 1. Arquitectura centralizada.....	13
Ilustración 2. Arquitectura distribuida .....	13
Ilustración 3. Distancias.....	16
Ilustración 4. Línea en EIB .....	16
Ilustración 5. Arquitectura instalación EIB .....	17
Ilustración 6. Información básica del telegrama.....	17
Ilustración 7. Dirección de grupo de nivel 2.....	18
Ilustración 8. Dirección de grupo de nivel 3.....	18
Ilustración 9. Transmisión eléctrica .....	18
Ilustración 10. Telegrama EIB.....	19
Ilustración 11. Sistema CCTV.....	21
Ilustración 12. Sistema híbrido.....	21
Ilustración 13. Sistema IP .....	22
Ilustración 14. Router y cámaras IP.....	23
Ilustración 15. Router y cámaras IP (II) .....	23
Ilustración 16. Dispositivos móviles .....	24
Ilustración 17. Arquitectura DomoUEx.....	27
Ilustración 18. Arquitectura MIDP .....	33
Ilustración 19. Esquema Java RMI.....	34
Ilustración 20. .NET Framework .....	36
Ilustración 21. .NET CLR.....	36
Ilustración 22. Ensamblado .NET.....	37
Ilustración 23. Arquitectura ICE .....	52
Ilustración 24. Mapa conceptual.....	65
Ilustración 25 Arquitectura QuercusDomoSystem .....	68
Ilustración 26. Diagrama de paquetes .....	74
Ilustración 27. Diagrama de clases servidor instalación .....	75
Ilustración 28. MServi .....	76
Ilustración 29. CBus.....	76
Ilustración 30. CDomoUexApp .....	77
Ilustración 31. CDomoUexDlg .....	77
Ilustración 32. Diagrama de clases videovigilancia .....	78
Ilustración 33. DomoUExModLessDlg.....	79
Ilustración 34. DomoUExCamera .....	79
Ilustración 35. DomoUExCamConfig.....	80
Ilustración 36. DomoUExMDSocket.....	81
Ilustración 37. DomoUExMDManager .....	81
Ilustración 38. DomoUExCamConfComl .....	82
Ilustración 39. CDomoUExCamServerDlg .....	82
Ilustración 40. Diagrama de clases cliente .....	84
Ilustración 41. funcion.....	85
Ilustración 42. instalacion .....	85

Ilustración 43. clientel .....	86
Ilustración 44. frmCamConfig.....	86
Ilustración 45. Form1 .....	87
Ilustración 46. Diagrama de clases cliente dispositivos móviles.....	88
Ilustración 47. DomoUEXCamEditDlg.....	88
Ilustración 48. DomoUEXVideoDialog .....	89
Ilustración 49. DomoUEXTabControl .....	89
Ilustración 50. clientel .....	89
Ilustración 51. CDomUEXPocketDlg .....	90
Ilustración 52. DUConfig.....	91
Ilustración 53. Diagrama de secuencia Acceder Instalación .....	92
Ilustración 54. Diagrama de secuencia Acceder Cámaras.....	92
Ilustración 55 Diagrama de secuencia Recibir Evento .....	93
Ilustración 56. DUConfig.....	109
Ilustración 57. Servidor instalación domótica.....	110
Ilustración 58. Servidor videovigilancia.....	112
Ilustración 59. Cliente - Instalación domótica.....	113
Ilustración 60. Cliente - Videovigilancia .....	114
Ilustración 61. Cliente - Configuración de cámara .....	115
Ilustración 62. Configuración servidores para dispositivos móviles .....	116
Ilustración 63. Interfaz dispositivos móviles .....	116
Ilustración 64. Arquitectura QuercusDomoSystem .....	117
Ilustración 65. Arquitectura DomoUEX.....	117
Ilustración 66. Diagrama de clases ampliado .....	131

## 1 Introducción

En este capítulo haremos un repaso, de forma general, de los diferentes campos en los que se basa el proyecto: domótica, videovigilancia, dispositivos móviles y tecnologías para la programación distribuida.

### 1.1 Domótica

La domótica es un concepto que se refiere a la integración de las distintas tecnologías en el hogar mediante el uso simultáneo de la electricidad, la electrónica, la informática y las telecomunicaciones. Su fin es mejorar la seguridad, el confort, la flexibilidad, las comunicaciones, el ahorro energético, facilitar el control integral de los sistemas para los usuarios y ofrecer nuevos servicios.

#### 1.1.1 Áreas de aplicación

Algunas de las áreas principales de la domótica son:

- **Automatización y Control:** incluye el control (abrir / cerrar, on / off y regulación) de la iluminación, climatización, persianas y toldos, puertas y ventanas, cerraduras, riego, electrodomésticos, suministro de agua y gas, etc.
- **Seguridad:** incluye alarmas de intrusión, alarmas personales y alarmas técnicas (incendio, humo, agua, gas, fallo de suministro eléctrico) y videovigilancia.
- **Telecomunicaciones:** incluye transmisión de voz y datos con redes locales (LAN) para compartir acceso de alta velocidad a Internet, recursos y el intercambio entre todos los equipos.
- **Audio y vídeo:** incluye la distribución de imágenes de vídeo capturadas con cámaras dentro y fuera de la casa a toda la casa y a través de Internet. Otra parte de audio / vídeo trata del entretenimiento como el multi-room y el "Cine En Casa".

Gracias a la aplicación de la domótica a las áreas anteriormente expuestas, se pueden crear otros campos de acción de la misma, como por ejemplo:

- **Programación de escenarios:** mediante los cuales, con una simple acción se puede adecuar toda la vivienda al gusto del usuario. Por ejemplo, pulsando un único botón se puede acondicionar la vivienda en los meses de verano, bajando las persianas hasta un punto determinado, encendiendo el aire acondicionado a una hora determinada, apertura y cierre de ventanas según temperaturas...
- **Avisos por teléfono, sms o email:** nuevos métodos de notificación de eventos hacia el usuario.

### 1.1.2 Componentes

Un sistema domótico está compuesto por tres tipos de dispositivos, sensores, actuadores y controladores.

- **Controlador:** es un programa informático que permite al sistema operativo interactuar con un periférico, haciendo una abstracción del hardware y proporcionando una interfaz (posiblemente estandarizada) para usarlo. Se puede esquematizar como un manual de instrucciones que indica cómo controlar y comunicarse con un dispositivo en particular. Por tanto, es una pieza esencial, sin la cual no se podría usar el hardware. Existen tantos tipos de controladores como tipos de periféricos y es frecuente encontrar más de un controlador posible para el mismo dispositivo, cada uno ofreciendo un nivel distinto de funcionalidades.
- **Sensor:** es un dispositivo que detecta manifestaciones de cualidades o fenómenos físicos, como la energía, velocidad, aceleración, tamaño, cantidad, etc. Podemos decir también que es un dispositivo que aprovecha una de sus propiedades con el fin de adaptar la señal que mide para que la pueda interpretar otro elemento. Muchos de los sensores son eléctricos o electrónicos, aunque existen otros tipos. Es un tipo de transductor que transforma la magnitud que se quiere medir en otra, que facilita su medida. Pueden ser de indicación directa (termómetro de mercurio) o pueden estar conectados a un indicador (posiblemente a través de un convertidor analógico a digital, un computador y un display) de modo que los valores detectados puedan ser leídos por un humano.
- **Actuador:** es el dispositivo de salida capaz de recibir una orden del controlador y realizar una acción. Los actuadores entran en acción cuando algún sensor detecta una acción en la vivienda (pulsar un interruptor).

### 1.1.3 Arquitecturas

La arquitectura de una instalación domótica es dependiente de donde reside el controlador de la misma. Existen dos tipos de arquitecturas posibles:

- **Centralizada:** el controlador de la instalación domótica se encuentra centralizado, es decir, recibe la información de todos los sensores, la procesa y genera la salida para el actuador.

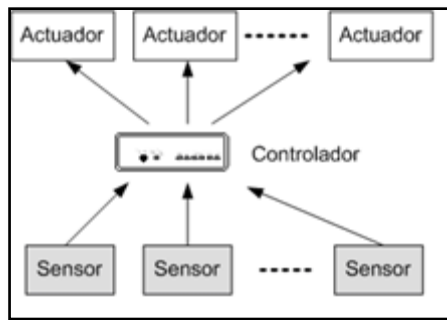


Ilustración 1. Arquitectura centralizada

- **Distribuida:** no existe la figura del controlador centralizado, sino que toda la inteligencia del sistema está distribuida por todos los módulos sean sensores o actuadores. Suele ser típico de los sistemas de cableado en bus.

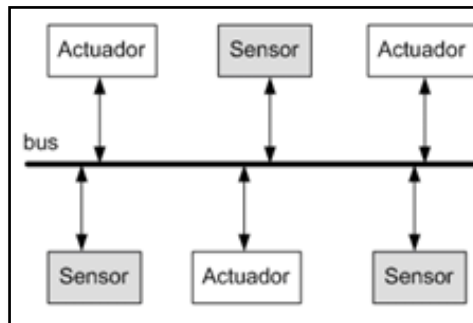


Ilustración 2. Arquitectura distribuida

- **Mixta:** mezcla las tecnologías anteriores. Son sistemas con arquitectura descentralizada en cuanto a que disponen de varios pequeños dispositivos capaces de adquirir y procesar la información de múltiples sensores y transmitirlos al resto de dispositivos distribuidos por la vivienda.

#### 1.1.4 Cableado de la instalación

Los dispositivos de la instalación domótica pueden estar comunicados mediante distintos medios: cable, comunicación inalámbrica y una versión mixta. A continuación se exponen los tipos:

- **Instalación cableada:** todos los sensores y actuadores están cableados a la central, o entre ellos, dependiendo del tipo de arquitectura que se esté utilizando. La arquitectura centralizada tiene normalmente una batería de respaldo, para, en caso de fallo del suministro eléctrico, poder alimentar a todos sus sensores y actuadores y así seguir funcionando normalmente durante unas horas.
- **Inalámbrica:** en este caso usan sensores inalámbricos alimentados por pilas o baterías y transmiten vía radio la información de los eventos a la central, o entre ellos. En el

sistema centralizado, la central funciona por red eléctrica y tiene una batería de respaldo para que el sistema siga funcionando si se corta el suministro.

- **Mixta:** combina las tecnologías anteriores en una única instalación.

### 1.1.5 Estándares domóticos

A continuación se presenta una breve descripción de los estándares domóticos más importantes a nivel mundial.

- **KNX-EIB:** Bus de Instalación Europeo con más de 20 años y más de 100 fabricantes de productos compatibles entre sí.
- **X10:** Protocolo de comunicaciones para el control remoto de dispositivos eléctricos, hace uso de los enchufes eléctricos, sin necesidad de nuevo cableado. Tiene poca fiabilidad frente a ruidos eléctricos, si los hay.
- **ZigBee:** Protocolo estándar, recogido en el IEEE 802.15.4, de comunicaciones inalámbrico.
- **OSGi:** Open Services Gateway Initiative. Especificaciones abiertas de software que permita diseñar plataformas compatibles que puedan proporcionar múltiples servicios. Ha sido pensada para su compatibilidad con Jini o UPnP.
- **LonWorks:** Plataforma estandarizada para el control de edificios, viviendas, industria y transporte.

### 1.1.6 KNX/EIB

#### 1.1.6.1 *Historia*

Originalmente conocido como Instabus, se fusionó en 1999 con otros dos estándares existentes en el mercado Europeo (BatiBUS y EHS), dando lugar a KNX que se establece como bus estándar Europeo. Esta unión se produjo por la irrupción en Europa de LON (Local Operating Networks) o LonWorks que de la mano de ECHELON y diferentes fabricantes, fue poco a poco “comiéndole” mercado a las anteriores, mostrándose como un protocolo más rápido, fiable y robusto.

#### 1.1.6.2 *Tecnología*

El bus EIB se puede definir como un sistema descentralizado en el que cada uno de los dispositivos conectados tiene control propio. Cada uno de los dispositivos tiene su propio

microprocesador y se pueden clasificar en sensores, que son los responsables de detectar actividad en el edificio, y en actuadores, que son capaces de modificar el entorno.

La EIBA propone una especificación abierta en la cual todos los dispositivos se conectan a través de la única línea de bus existente, sin precisar un control centralizado. Se basa en el protocolo CSMA/CA para solucionar el acceso al medio físico. Los sensores se comunican mandando telegramas a los actuadores que ejecutan los comandos apropiados.

El bus se adapta fácilmente a distintos tamaños y topologías pudiéndose conectar hasta 10000 dispositivos. El bus es independiente del medio físico que se utilice estando disponibles los siguientes:

- Par trenzado (9600bps).
- Red eléctrica (1200/2400bps, en un principio para 230V y 50Hz)
- EIB.net (10 Mbps sobre Ethernet)
- Radio Frecuencia.
- Infrarrojos.

Las instalaciones que existen en la actualidad están implementadas sobre par trenzado y, en menor medida, sobre red eléctrica, pudiendo tener elementos que se comunican mediante infrarrojos o radio frecuencia.

#### 1.1.6.3 *Topología*

La red del EIB se estructura de forma jerárquica. La unidad más pequeña se denomina línea, a la que se pueden conectar hasta un máximo de 64 dispositivos. La topología de la línea es libre siempre y cuando respete:

- que haya al menos una fuente de alimentación.
- que la longitud total no supere los 1000 m.
- que la distancia máxima entre la fuente de alimentación y un dispositivo sea menor de 350 m.
- que la distancia máxima entre dispositivos no supere los 750 m.
- que la distancia mínima entre dos fuentes de alimentación dentro de una misma línea sea mayor de 200 m.

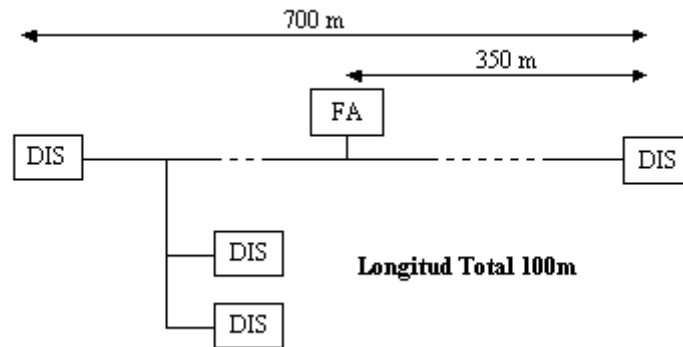


Ilustración 3. Distancias

Las líneas se agrupan en áreas. El área se compone de una línea principal de la que cuelgan hasta 15 líneas secundarias. Por tanto, un área podrá tener como máximo 960 dispositivos. Cada una de las líneas secundarias se conecta con la línea principal mediante un dispositivo llamado acoplador de línea. La línea principal deberá tener su propia fuente de alimentación.

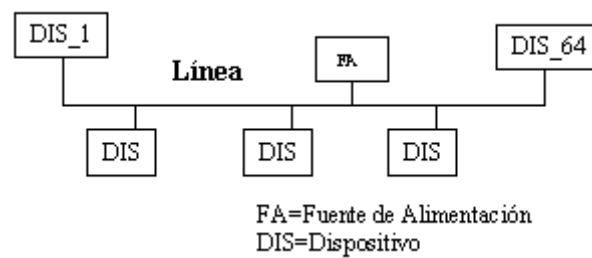


Ilustración 4. Línea en EIB

A su vez se puede disponer de hasta 15 áreas unidas mediante una línea principal denominada backbone. Como máximo se podrán conseguir hasta 14400 dispositivos. Las áreas se conectan al backbone mediante acopladores.



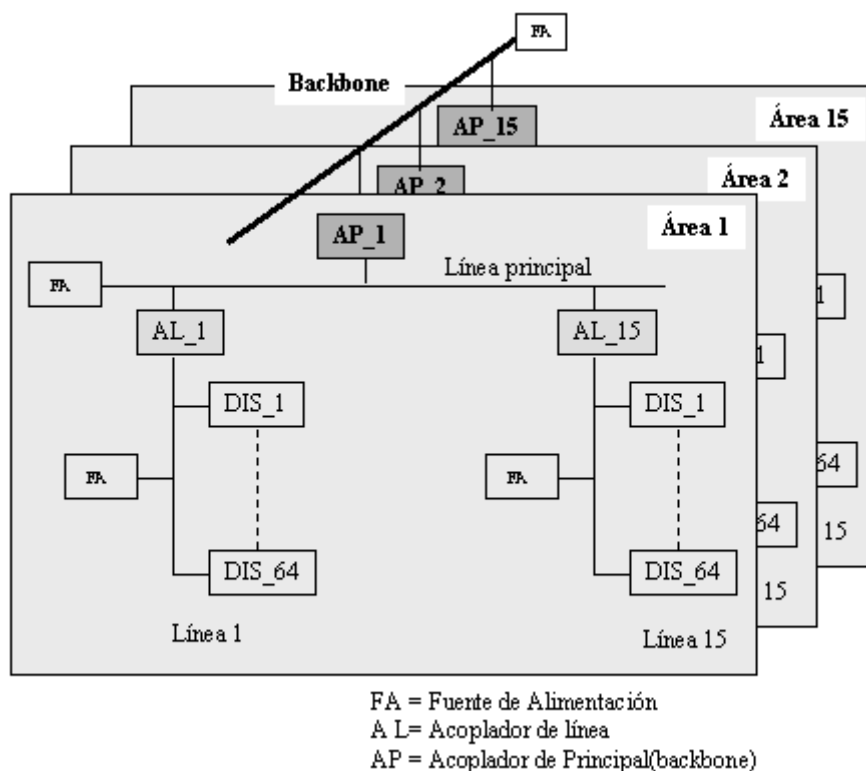


Ilustración 5. Arquitectura instalación EIB

Cada dispositivo tiene una dirección física de 16 bits asociada que le identifica unívocamente. La dirección de un dispositivo además define la localización de éste en la red. Cada dirección se divide en área, línea dentro del área, y número de dispositivo.

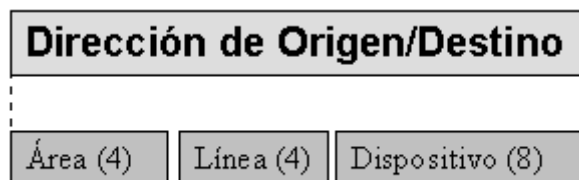
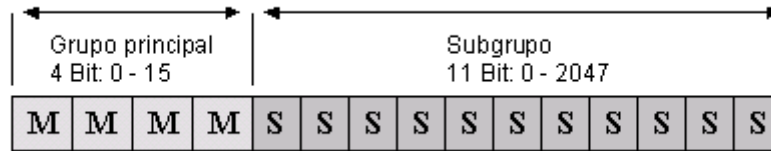


Ilustración 6. Información básica del telegrama

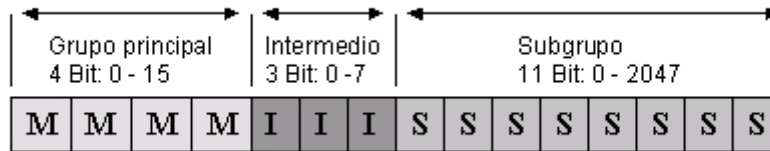
Las direcciones que empiezan por cero se reservan para los dispositivos acopladores.

Además de la dirección física, cada dispositivo puede tener una o más direcciones lógicas, denominadas direcciones de grupo. Las direcciones de grupo asocian funcionalmente dispositivos ya que todos los dispositivos que tengan la misma dirección de grupo reciben los mismos mensajes. Los sensores sólo pueden enviar telegramas a una dirección de grupo, mientras que los actuadores pueden tener varias direcciones de grupo, lo que les permite reaccionar a distintos sensores. Cualquier dispositivo de la red puede mandar telegramas a una dirección de grupo.

Dependiendo de la granularidad que el diseñador quiera dar a la red se pueden seleccionar direcciones de grupo de nivel 2 o de nivel 3. Las direcciones de grupo de nivel 2 dividen la dirección en dos campos: grupo principal y subgrupo, mientras que las de nivel 3 separan la dirección en: grupo principal, grupo intermedio y subgrupo. Con el nivel 2 se obtienen 15 grupos principales con 2047 subgrupos cada grupo. Para el nivel 3 la división queda en 15 grupos principales, cada uno con 7 grupos intermedios de 255 subgrupos cada uno.



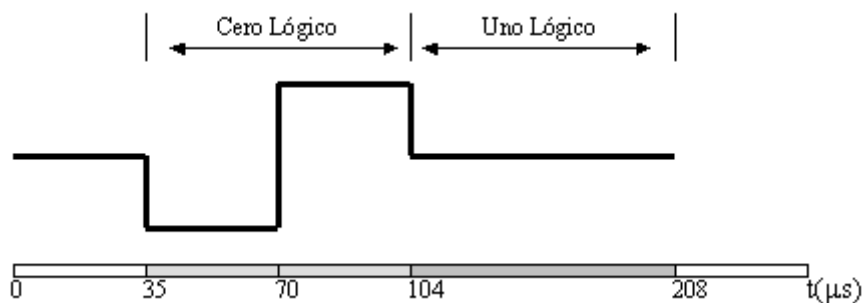
**Ilustración 7. Dirección de grupo de nivel 2**



**Ilustración 8. Dirección de grupo de nivel 3**

#### 1.1.6.4 **Telegramas**

Los dispositivos se comunican mediante señales binarias en banda base con una velocidad de transmisión de 9600 bps (en el caso de cable trenzado). Un cero lógico se representa mediante un flujo de corriente por el cable mientras que la ausencia de corriente significa un uno lógico.



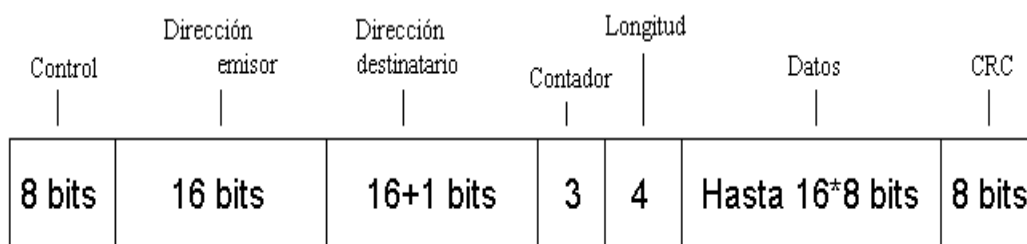
**Ilustración 9. Transmisión eléctrica**

Al tener que compartir el medio físico de transmisión, un dispositivo comenzará a transmitir siempre y cuando el bus esté desocupado. Cuando dos o más dispositivos transmiten simultáneamente se produce una colisión en el bus que debe ser resuelta mediante

un algoritmo CSMA/CD (Carrier Sense Multiple Access with Collision Avoidance). Los dispositivos se mantienen a la escucha mientras están transmitiendo. Tan pronto como detecten un cero cuando ellos están transmitiendo un uno, se pararán dejando el bus libre para el dispositivo de mayor prioridad.

El intercambio de información entre dos dispositivos se consigue mediante el envío de telegramas. Un telegrama se compone de un paquete de datos estructurado que el emisor envía y del correspondiente acuse de recibo con el que el receptor responde si no ha ocurrido ningún fallo. Cada paquete de datos se divide en los siguientes campos:

- Control (8 bits)
- Dirección del emisor (16 bits)
- Dirección del destinatario (16 bit +1 bit)
- Contador (3 bits)
- Longitud (4 bits)
- LSDU (Link Service Data Unit): datos que se van a transmitir (hasta 16x8 bits)
- Byte de comprobación (8 bits)



**Ilustración 10. Telegrama EIB**

El campo de control sirve para determinar la prioridad del mensaje, así como marca inicial del telegrama.

Tanto la dirección del emisor como la del receptor siguen el formato explicado en el apartado anterior, añadiendo un bit más en la dirección del destinatario que indica si se trata de una dirección física o de una dirección de grupo.

El contador se utiliza para funciones de enrutamiento, contando el número de saltos que ha dado el paquete. La longitud indica cuantos bytes ocupa la LSDU.

El último byte se utiliza para comprobar que los anteriores han sido transmitidos correctamente.

#### 1.1.6.5 *Ventajas KNX-EIB*

- **Independiente de cualquier tecnología tanto de hardware como de software:**

La tecnología KNX se ha convertido a nivel mundial en el primer estándar abierto libre de royalties e independiente de la plataforma hardware para Sistemas de Control de Viviendas y Edificios.

- **Interoperable:**

Asegura que los productos de distintos fabricantes, utilizados en distintas aplicaciones, funcionarán y se comunicarán unos con otros. Esto permite un alto grado de flexibilidad en la ampliación y modificación de las instalaciones.

- **Calidad del producto:**

La Asociación Konnex requiere un alto nivel de control de calidad durante todas las etapas de la vida del producto. Por esta razón, todos los miembros de Konnex que desarrollan productos KNX bajo la marca KNX, tienen que cumplir con la ISO 9001 antes que puedan solicitar la certificación de productos KNX. Adicionalmente a la ISO 9001, los productos deben cumplir con los requerimientos de la norma Europea para Sistemas Electrónicos en Viviendas y Edificios, es decir, EN 50090-2-2. En caso de duda, la Asociación Konnex está autorizada a testear de nuevo los productos para su certificación o bien puede requerir a los fabricantes una declaración de conformidad del hardware.

- **Funcionalidades independientes de los fabricantes:**

El estándar KNX contiene distintos perfiles de aplicación para diversas aplicaciones comunes en Viviendas y Edificios. Bajo la supervisión del Grupo Técnico varios grupos de trabajo de especificación de las aplicaciones realizan propuestas para estandarizar diversas funcionalidades (inputs, outputs, diagnóstico de datos y parámetros) en el dominio específico de aplicación. Para asegurar un alto grado de disciplina cruzada e interoperabilidad multi-vendedor el Task Force Interworking reevalúa estas propuestas antes de que se tome la decisión de incorporar un perfil de aplicación en el estándar KNX.

- **Herramienta Común de Software para la Ingeniería independiente del fabricante:**

La Asociación Konnex pone a disposición una herramienta de software para la ingeniería independiente del fabricante para planificar las uniones lógicas y configurar los productos certificados KNX.

## 1.2 Videovigilancia

### 1.2.1 Introducción

La historia de la vigilancia mediante cámaras comienza con la tecnología *Closed-circuit Television* (CCTV) donde, a diferencia de los sistemas *broadcast* (TV), la señal de vídeo no se transmite de forma abierta sino que se envía a una serie de monitores específicos y la grabación de vídeo se realiza mediante un sistema *Video Cassette Recoder* (VCR).

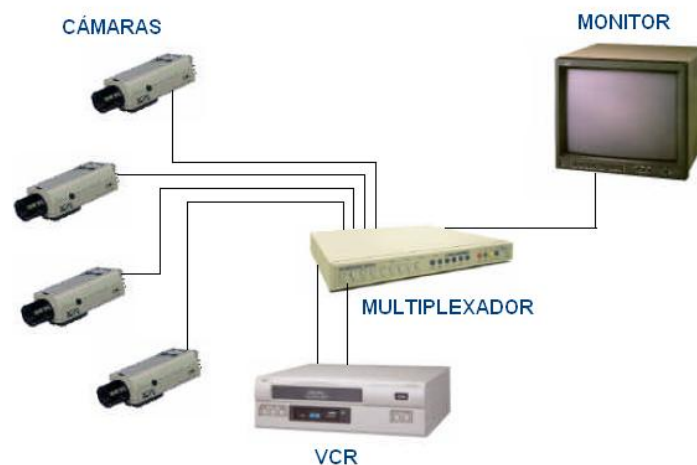


Ilustración 11. Sistema CCTV

Con la aparición de los primeros grabadores digitales de vídeo y el boom de las redes TCP/IP comienza a hablarse de sistemas híbridos donde la captura de vídeo sigue siendo analógica pero el almacenamiento pasa a ser digital (más efectivo) y los usuarios de la red tienen acceso al sistema.

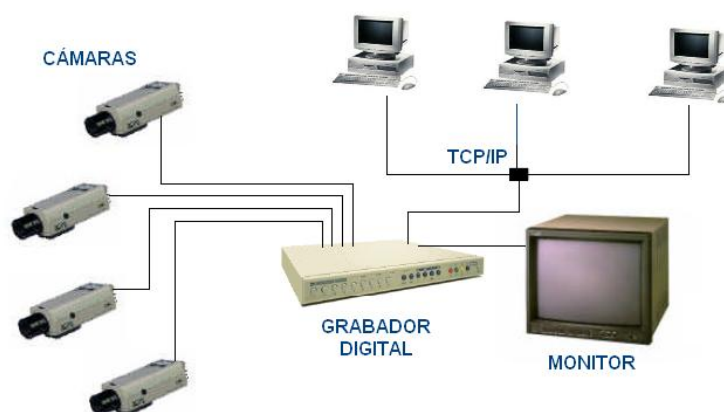


Ilustración 12. Sistema híbrido

Finalmente, en los últimos años estamos viviendo un enorme crecimiento en el uso de sistemas totalmente digitales basados en TCP/IP debido, en gran medida, a la posibilidad de realizar la fase de captura mediante mecanismos digitales sin caer en pérdidas de calidad significantes (que era el principal problema de este tipo de esquemas).

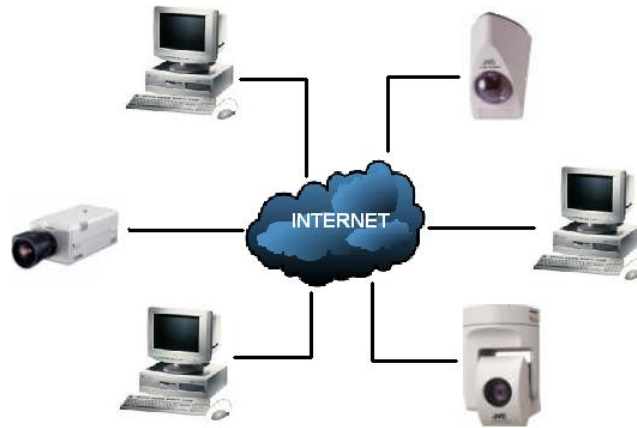


Ilustración 13. Sistema IP

A diferencia de las cámaras analógicas, las cámaras IP incluyen un software de administración más o menos potente (desde pequeños servidores web hasta sistemas operativos completos) que permiten descentralizar la arquitectura convirtiéndola en un esquema distribuido.

### 1.2.2 Ventajas vídeo IP

Los sistemas analógicos cuentan con una serie de ventajas importantes: mercado establecido, ampliamente testeados, costes reducidos y la idea de que un circuito cerrado es más seguro que uno abierto. Además, los sistemas híbridos suplen ciertas carencias de CCTV: grabación digital (ahorro en recursos), acceso multiusuario...

No obstante, el vídeo IP se está imponiendo sobre el vídeo analógico al ofrecer ciertas características únicas:

- La mayoría de procesos a los que se ve sometido el vídeo son procesos software y, por tanto, más flexibles para su tratamiento y modificación.
- La red abierta permite una mayor escalabilidad y actualmente cuentan con medidas de seguridad suficientemente potentes.
- Las cámaras IP pueden ser sistemas “inteligentes” donde se implementen funciones como la activación de alarmas, notificación vía e-mail, detección de movimiento, etc....
- CCTV funciona sobre cables coaxiales (costos de instalación grandes) mientras que vídeo IP puede funcionar sobre cualquier tipo de instalación TCP/IP. Un punto importante a tener en cuenta es qué ocurre si queremos disponer de varias cámaras en un recinto y situarlas detrás de un router doméstico. Puesto que el acceso a las cámaras se hace mediante una dirección IP y un número de puerto TCP (lo que comúnmente conocemos como *socket*), ¿habría que proporcionar una IP pública diferente para cada cámara? La respuesta es mucho más simple, únicamente es necesario configurar el router de tal forma que redirija el tráfico entrante a ciertos puertos TCP acordados previamente hacia sus correspondientes sockets.

Por ejemplo:

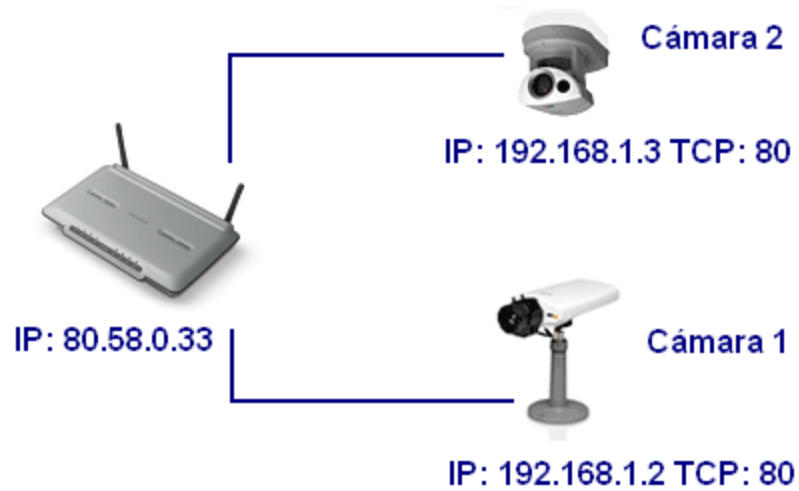


Ilustración 14. Router y cámaras IP

En este diagrama, la dirección pública del router es 80.58.0.33, las cámaras tienen asignadas direcciones IP internas (no accesibles desde el exterior) y ambas esperan las conexiones entrantes en el puerto TCP 80 (el más común para este tipo de cámaras). Para permitir el acceso a los usuarios desde el exterior de la red bastaría con indicar al router una tabla de redirección, por ejemplo:

SOCKET PÚBLICO	SOCKET PRIVADO
80.58.0.33:10001	192.168.1.2:80
80.58.0.33:10002	192.168.1.3:80

Con esta información, un usuario podría conectarse al socket destino 80.58.0.33:10002 y el router se encargaría de redireccionar los datos (línea roja) a la Cámara 2 de nuestro sistema.

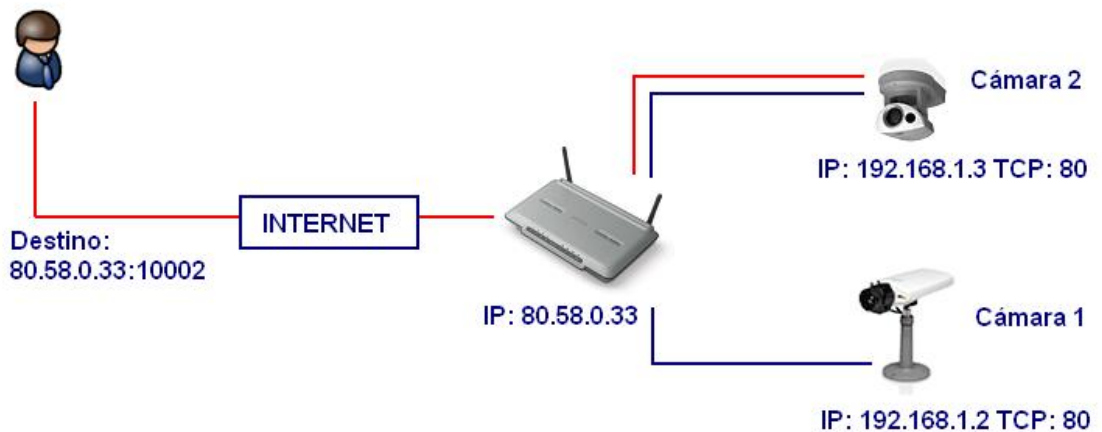


Ilustración 15. Router y cámaras IP (II)

Pero, sobre todo, el vídeo IP mejora de forma rápida y constante su calidad de imagen y reduce cada vez más su precio, principales obstáculos que frenaban esta tecnología frente a CCTV.

### 1.3 Dispositivos móviles

Desde los primeros teléfonos móviles de la década de los ochenta hasta las actuales *asistentes personales digitales* (PDA) la industria de los dispositivos móviles ha vivido un continuo crecimiento exponencial de las capacidades de sus productos.



Ilustración 16. Dispositivos móviles

Sin embargo, ese crecimiento de la potencia del hardware no se ha visto respaldado por un significativo aumento de las prestaciones software hasta hace poco, con la irrupción de los dispositivos móviles en Internet y el planteamiento de una nueva pregunta: *para qué queremos acceder a Internet desde un dispositivo móvil*. La respuesta a esta pregunta es sencilla: para cualquier actividad que podamos realizar desde cualquier computador convencional. Puesto que la tendencia actual es buscar la convergencia de las tecnologías en un único dispositivo multimedia potente y portable, no deberíamos limitar nuestras aspiraciones.

Como veremos más adelante, la posibilidad de implementar aplicaciones que sigan modelos complejos de programación distribuida en un dispositivo móvil es ya una realidad soportada por diversas tecnologías.

Hoy por hoy disponemos de la tecnología para embeber un ORB en un dispositivo móvil (algo impensable hasta hace no mucho tiempo), hoy por hoy disponemos de la tecnología para controlar una instalación domótica desde un dispositivo móvil.



## 1.4 Tecnologías distribuidas

### 1.4.1 Introducción

La computación, desde sus inicios, ha sufrido muchos cambios, de los grandes ordenadores que permitían realizar tareas en forma limitada y de uso un tanto exclusivo de organizaciones muy selectas, hasta los actuales ordenadores, ya sean personales o portátiles, que tienen las mismas e incluso mayores capacidades que los primeros y que están cada vez más introducidos en la vida cotidiana de una persona.

Los mayores cambios se atribuyen, principalmente, a dos causas que se dieron desde la década de los setenta:

- El desarrollo de los microprocesadores, que permitieron reducir en tamaño y costo los ordenadores y aumentar en gran medida las capacidades de los mismos y su acceso a más personas.
- El desarrollo de las redes de área local y de las comunicaciones, que permitieron conectar ordenadores con posibilidad de transferencia de datos a alta velocidad.

Es en este contexto donde aparece el concepto de "Sistemas Distribuidos" que se ha popularizado tanto en la actualidad y que tiene como ámbito de estudio las redes como por ejemplo: Internet, redes de teléfonos móviles, redes corporativas, redes de empresas, etc.

Una posible definición de sistema distribuido es la siguiente:

"Sistemas cuyos componentes hardware y software se comunican y coordinan sus acciones mediante el paso de mensajes para el logro de un objetivo. Se establece la comunicación mediante un protocolo prefijado por un esquema cliente-servidor".

### 1.4.2 Características

A continuación se muestran las características que debe poseer un sistema distribuido:

- **Concurrencia:** permite que los recursos disponibles en la red puedan ser utilizados simultáneamente por los usuarios y/o agentes que interactúan en la red.
- **Carencia de reloj global:** las coordinaciones para la transferencia de mensajes entre los diferentes componentes para la realización de una tarea, no tienen una temporización general, está más bien distribuida a los componentes.
- **Fallos independientes de los componentes:** cada componente del sistema puede fallar independientemente, con lo cual, los demás pueden continuar ejecutando sus acciones. Esto permite el logro de las tareas con mayor efectividad, pues el sistema en su conjunto continúa trabajando.

### 1.4.3 Arquitectura Cliente/Servidor

El concepto de cliente/servidor permite una forma eficiente de utilizar todos estos recursos de máquina de tal forma que la seguridad y fiabilidad que proporcionan los entornos mainframe se traspa a la red de área local. A esto hay que añadir la ventaja de la potencia y simplicidad de los ordenadores personales.

La arquitectura cliente/servidor es un modelo para el desarrollo de sistemas de información en el que las transacciones se dividen en procesos independientes que cooperan entre sí para intercambiar información, servicios o recursos. Se denomina cliente al proceso que inicia el diálogo o solicita los recursos y servidor al proceso que responde a las solicitudes.

En este modelo las aplicaciones se dividen de forma que el servidor contiene la parte que debe ser compartida por varios usuarios, y en el cliente permanece sólo lo particular de cada usuario.

Los clientes realizan generalmente funciones como:

- Manejo de la interfaz de usuario.
- Captura y validación de los datos de entrada.
- Generación de consultas e informes sobre las bases de datos.

Por su parte los servidores realizan, entre otras, las siguientes funciones:

- Gestión de periféricos compartidos.
- Control de accesos concurrentes a bases de datos compartidas.
- Enlaces de comunicaciones con otras redes de área local o extensa.

Siempre que un cliente requiere un servicio lo solicita al servidor correspondiente y éste le responde proporcionándolo. Normalmente, pero no necesariamente, el cliente y el servidor están ubicados en distintos procesadores. Los clientes se suelen situar en ordenadores personales y/o estaciones de trabajo y los servidores en procesadores departamentales o de grupo.

Entre las principales características de la arquitectura cliente/servidor se pueden destacar las siguientes:

- El servidor presenta a todos sus clientes una interfaz única y bien definida.
- El cliente no necesita conocer la lógica del servidor, sólo su interfaz externa.
- El cliente no depende de la ubicación física del servidor, ni del tipo de equipo físico en el que se encuentra, ni de su sistema operativo.
- Los cambios en el servidor implican pocos o ningún cambio en el cliente.

## 2 Definición del problema

QuercusDomoSystem nace a partir de varios proyectos anteriores que, unidos, dieron como resultado DomoUEx. En este capítulo nos centraremos en aquellos puntos de DomoUEx que requieren un remodelado y en las características que éste no incluye y que sería interesante contemplar en el nuevo sistema.

### 2.1 Sistema existente

#### 2.1.1 Historia DomoUEx

El proyecto domótico de la Universidad de Extremadura comenzó a gestarse en el año 2001, de la mano del profesor Juan Hernández Núñez y el alumno José Luis Huertas Fernández, bajo el título “Control de dispositivos domóticos EIB usando Java y Corba”.

Se trataba de un proyecto totalmente novedoso y muy ambicioso, ya que trataban de contemplar todos los aspectos posibles relacionados con la domótica. Lo consiguieron y, de ahí, nació DomoUEx.

DomoUEx se desarrolló a partir de la siguiente arquitectura:

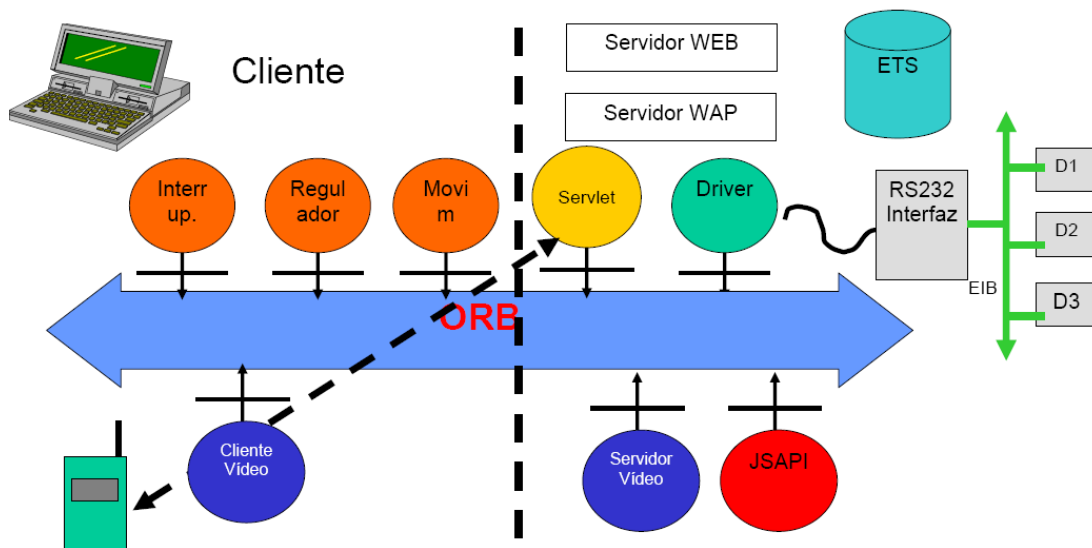


Ilustración 17. Arquitectura DomoUEx

El siguiente paso a llevar a cabo en el proyecto fue la realización de un interfaz basado en web, que permitiera las mismas funcionalidades que el comentado anteriormente, y fue realizado por Joaquín Recio.

José María Conejero Manzano llevó a cabo la siguiente ampliación del proyecto, estudiando la aplicación del modelo de componentes CORBA (CCM) al bus EIB.

En el año 2005 fue llevado a cabo uno de los cambios que permitirían una mejor comercialización del producto: sustituir las librerías EIBlet Engine por las desarrolladas por Benjamín Domínguez Iglesias. Aunque las librerías desarrolladas funcionaban correctamente contaban con algunas carencias frente a las anteriores.

En Noviembre de 2005, Juan Hernández Núñez y José María Conejero Manzano nos dieron la oportunidad de realizar este proyecto. La idea principal era ampliar algunos ámbitos donde DomoUX todavía no daba soporte.

Tras diversos estudios de posibilidades, vimos, entre los que formábamos parte del nuevo grupo de desarrollo del proyecto, que era una opción muy costosa. Por tanto, había que buscar otro camino.

Entre Juan Hernández Núñez, José María Conejero Manzano, César Silgo Ortiz y Alfredo Moreno Muñoz decidimos romper con DomoUX, y empezar el desarrollo de una nueva plataforma domótica, con arquitectura diferente y nuevo nombre: QuercusDomoSystem.

### 2.1.2 Tecnologías

Veamos una breve descripción de las tecnologías utilizadas para el desarrollo de DomoUX (ver *Ilustración 7. Arquitectura DomoUX*).

- RS-232: el sistema anterior se conectaba al bus a través del puerto serie.
- Falcon (Driver): son las librerías de acceso al bus. Mediante ellas se interactúa de forma directa con el mismo. Estas librerías son facilitadas por la *European Installation Bus Association* (EIBA).
- EIBlet-Engine: estas librerías permiten un nivel de abstracción superior al de las librerías Falcon. Se trata de unas librerías JNI (Java Native Interface) que implementan una capa de traducción para ofrecer una interfaz Java de las librerías Falcon, desarrolladas en C. Son librerías propietarias de la empresa JNetSystem.
- Java: lenguaje de programación utilizado para implementar las diferentes partes del proyecto.
- CORBA (ORB): middleware elegido para realizar las comunicaciones entre las distintas aplicaciones que componen el sistema.

### 2.1.3 Análisis de debilidades

En esta sección vamos a extraer las características de DomoUX que sería interesante mejorar o sustituir por otras alternativas.

- JNetSystem, la empresa desarrolladora de EIBlet-Engine, ya no da soporte para esas librerías y es conveniente eliminarlas del sistema.
- El acceso al bus se realiza mediante el puerto serie (RS-232). Puesto que este puerto cada vez es menos común, deberán estudiarse otras interfaces como USB, RJ45, etc.
- El código fuente de la aplicación es dependiente de la instalación con la que se trabaja por lo que habría que reprogramarlo cada vez que se utilice el sistema sobre una nueva instalación.
- El sistema de videovigilancia en red, mediante cámaras IP, no se encuentra totalmente integrado en el sistema. Habría que ahondar en las tecnologías de videovigilancia para hacer interactuar las cámaras con el resto de la instalación y contar con capacidad de transmisión de audio.
- No existe una versión para dispositivos móviles funcional.
- Sólo se permite la conexión de un único cliente a la vez. Solucionar esto replantea el uso de CORBA como middleware.
- No existe una representación gráfica real del recinto.

## 2.2 Especificación de requisitos

Teniendo en cuenta la información expuesta en las secciones anteriores, a continuación detallamos las características de funcionalidad que debe tener el sistema que vamos a desarrollar diferenciando requisitos mínimos, medios y máximos que queremos alcanzar.

### **Requisitos mínimos:**

- Nuevo método de conexión al bus mediante USB.
- Inserción de la licencia de las librerías Falcon de acceso al bus.
- Utilización de un middleware ligero que permita la utilización del sistema en dispositivos móviles.
- Inserción de audio en la aplicación.
- Integración vídeo IP en el sistema.
- Desarrollo de una versión para dispositivos móviles.
- Múltiples clientes simultáneos.

### **Requisitos medios:**

- Utilización de planos desarrollados en AutoCad.

- Inserción de la localización de los dispositivos de la instalación domótica de forma manual y del plano de su vivienda.
- Obtención de eventos a través de las cámaras disponibles.
- Audio bidireccional en el sistema.
- Mantenimiento, configuración e inserción de cualquier número de cámaras en el sistema a través de la aplicación.
- Conexiones seguras mediante contraseñas.

**Requisitos máximos:**

- Sistema de alertas a través de sms o correo electrónico.
- Grabación de vídeo en alarmas.

## 3 Estudio de viabilidad

### 3.1 Alternativas exploradas

Una vez definido el problema y teniendo en cuenta los requisitos planteados, en esta sección se presentan las diferentes tecnologías sobre las que se ha centrado la etapa de investigación mediante una descripción de cada una para, posteriormente, analizar sus ventajas e inconvenientes y justificar la selección de una de ellas como base para construir la solución al problema.

#### 3.1.1 OPC

OPC (**O**LE for **P**rocess **C**ontrol) es un conjunto específico de normas de comunicación en el campo del control y supervisión de procesos. El primer estándar (al principio llamado simplemente la Especificación OPC y ahora llamado la Especificación de Acceso de Datos) surgió como resultado de la colaboración de los mayores proveedores de automatización mundiales y de Microsoft. Al principio se basó en OLE de Microsoft COM (el modelo de objeto componente) y DCOM (el modelo de objeto distribuido componente) cuya especificación definió un juego estándar de objetos, interfaces y métodos para el empleo en el control de procedimiento y usos de automatización de fabricación para facilitar la interoperabilidad entre ellos. Las tecnologías COM/DCOM proporcionaron el marco de productos de software para ser desarrollados. Existen cientos de servidores de Acceso de Datos OPC y clientes disponibles.

La agregación de la especificación OPC a la tecnología OLE de Microsoft en Windows permitió la estandarización.

Los productos OPC son construidos una vez y reutilizados muchas veces, por lo tanto, tienen que llevarse a cabo continuos controles de calidad y mejora del producto.

En realidad OPC es un conjunto de protocolos entre los que podemos destacar los siguientes:

- OPC-DA (Data Access).- El original, sirve para el intercambio de datos a tiempo real entre servidores y clientes.
- OPC-AE (Alarms & Events).- Proporciona alarmas y notificaciones de eventos.
- OPC B (Batch).- Útil en procesos discontinuos.
- OPC DX (Data eXchange).- Proporciona interoperabilidad entre varios servidores.
- OPC HDA (Historical Data Access). - Acceso histórico a datos OPC.
- OPC S (Security).- Especifica cómo controlar el acceso de los clientes a los servidores.
- OPC XML-DA (XML Data Access). - Sirve para el intercambio de datos entre servidores y clientes como OPC-DA pero en vez de utilizar tecnología COM/DCOM utiliza mensajes SOAP (sobre HTTP) con documentos en XML
- OPC CD (Complex Data).- Permite a los servidores exponer y describir tipos de datos más complicados en forma de estructuras binarias y documentos XML.

### 3.1.2 MIDP

*Mobile Information Device Profile* (MIDP) es una especificación para el uso de Java en dispositivos móviles. Es parte del framework *Java Micro Edition* (Java ME) y está situado en el nivel más alto de la configuración *Connected Limited Device Configuration* (CLDC) que cumplen los dispositivos sujetos a las siguientes limitaciones:

- Procesador: 16 bit/16 MHz o más
- Memoria: 160-512 KB de memoria total disponible para la plataforma Java
- Alimentación: Alimentación limitada, a menudo basada en batería
- Trabajo en red: Conectividad a algún tipo de red, con ancho de banda limitado habitualmente.

Es decir, la gran mayoría de los dispositivos móviles actuales.



Los usuarios pueden seleccionar las aplicaciones MIDP situadas en un servidor web. El dispositivo comprueba si puede ejecutarla y la descarga, la verifica y compila a byte code para ponerla en funcionamiento. Las aplicaciones instaladas se pueden ejecutar, actualizar y borrar de forma sencilla.

Las aplicaciones MIDP permiten tener aplicaciones intuitivas y gráficas. La interfaz de usuario se ha optimizado para las pequeñas pantallas, mecanismos de introducción de datos y otras características de los dispositivos móviles. Las aplicaciones MIDP se pueden instalar y ejecutar en local, trabajar en red o sin conexión y pueden almacenar y gestionar de forma segura datos de forma local.

#### **Conectividad**

MIDP soporta los siguientes estándares: HTTP, HTTPS, datagramas, sockets, sockets de servidor, comunicación serie, servicio de mensajería, servicio de multidifusión.

#### **Provisión OTA**

MIDP permite desplegar y actualizar aplicaciones activamente *Over the air* (OTA). La especificación MIDP define cómo se localizan, instalan, actualizan y eliminan en dispositivos móviles. Esto se aplica también a los proveedores de servicios que hayan adoptado el modelo, que pueden acceder a los dispositivos para la consecuente actualización e instalación.



### Desarrollo con MIDP

La arquitectura de las aplicaciones desarrolladas sobre dispositivos que incorporan la arquitectura MIDP coexiste con terceras aplicaciones que se desarrollan sobre las distintas capas de aplicación que implementan estos dispositivos, dando lugar a esquemas con distintos tipos de aplicaciones sobre un mismo dispositivo que se aprovechan de la tecnología existente:

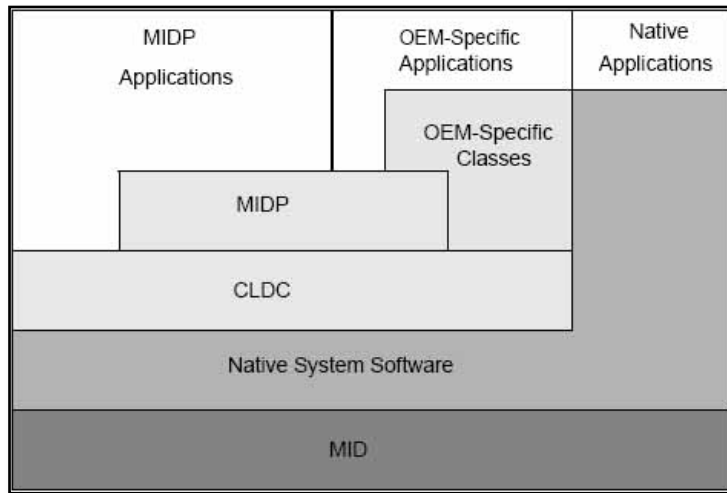


Ilustración 18. Arquitectura MIDP

- **MIDP:** Una aplicación MIDP o MIDlet es aquella que sólo utiliza las APIs definidas por la arquitectura MIDP o CLDC.
- **Específica OEM:** Estas aplicaciones dependen de clases ajenas a las especificación MIDP (dependen de clases específicas de fabricante de dispositivos o a terceros). Normalmente no son portables a otros dispositivos.
- **Nativa:** Las aplicaciones nativas no están escritas en Java sino sobre el software nativo del dispositivo.

### 3.1.3 Java RMI

*Java Remote Method Invocation API* (Java RMI) es la herramienta ofrecida por Sun para el desarrollo de aplicaciones Java distribuidas. Además, es parte del entorno de ejecución estándar.

Al estar pensado para entornos homogéneos permite ciertas operaciones no disponibles en otros sistemas: pasar objetos por referencia, recolección de basura distribuida o pasar tipos arbitrarios. La desventaja es, obviamente, que no permite la compatibilidad con lenguajes diferentes de Java.

Por medio de RMI, un programa Java puede exportar un objeto. A partir de esa operación este objeto está disponible en la red, esperando conexiones en un puerto TCP. Un cliente puede entonces conectarse e invocar métodos. La invocación consiste en el "marshaling" de los parámetros (utilizando la funcionalidad de "serialización" que provee

Java), luego se sigue con la invocación del método (cosa que sucede en el servidor). Mientras esto sucede el llamador se queda esperando por una respuesta. Una vez que termina la ejecución el valor de retorno (si lo hay) es serializado y enviado al cliente. El código cliente recibe este valor como si la invocación hubiera sido local.

RMI sigue un modelo de cuatro capas:

- **Aplicación:** se corresponde con la implementación real de las aplicaciones cliente y servidor. Aquí tienen lugar las llamadas a alto nivel para acceder y exportar objetos remotos.
- **Stub-Skeleton:** interactúa directamente con la capa de aplicación. Todas las llamadas a objetos remotos y acciones junto con sus parámetros y retorno de objetos tienen lugar en esta capa.
- **Referencia remota:** es responsable del manejo de la parte semántica de las invocaciones remotas. También es responsable de la gestión de la replicación de objetos y realización de tareas específicas de la implementación con los objetos remotos, como el establecimiento de las persistencias semánticas y estrategias adecuadas para la recuperación de conexiones perdidas. En esta capa se espera una conexión de tipo stream desde la capa de transporte.
- **Transporte:** es la responsable de realizar las conexiones necesarias y manejo del transporte de los datos de una máquina a otra. El protocolo de transporte subyacente para RMI es *Java Remote Method Protocol* (JRMP), que sólo es interpretado por programas Java.

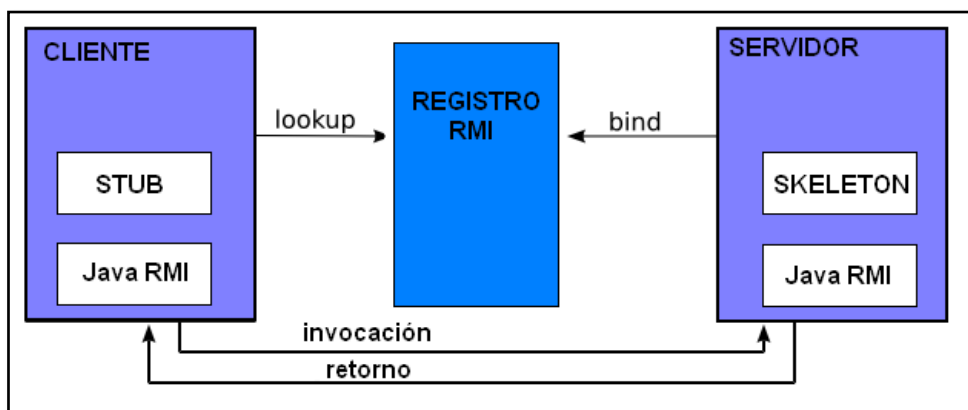


Ilustración 19. Esquema Java RMI

Toda aplicación RMI consta de dos partes:

- **Servidor:** crea algunos objetos remotos, referencias para hacerlos accesibles y espera a que el cliente los invoque.
- **Cliente:** obtiene una referencia a objetos remotos en el servidor y los invoca.

### 3.1.4 .NET

.NET no es una tecnología en sí misma, sino un conjunto de herramientas y tecnologías de Microsoft integradas en un único entorno y, en su mayoría, dependientes del *Microsoft .NET Framework* (componente de los sistemas operativos Windows).

Microsoft propone este entorno mostrando un especial cuidado en la comunicación transparente entre los distintos componentes permitiendo un desarrollo rápido de sistemas heterogéneos y buscando la integración de todos sus productos (desde el sistema operativo hasta las aplicaciones comerciales) reemplazando las antiguas APIs como Win32 por el nuevo framework.

Debido a las ventajas que la disponibilidad de una plataforma de este tipo puede darle a las empresas de tecnología y al público en general, muchas otras empresas e instituciones se han unido a Microsoft en el desarrollo y fortalecimiento de la plataforma .NET, ya sea por medio de la implementación de la plataforma para otros sistemas operativos aparte de Windows (Proyecto Mono de Ximian/Novell para Linux/MacOS X/BSD/Solaris), el desarrollo de lenguajes de programación adicionales o la creación de bloques (como controles, componentes y bibliotecas de clases adicionales) siendo algunas de ellos software libre, distribuidos bajo la licencia GPL.

Con esta plataforma, Microsoft irrumpe de lleno en el campo de los servicios web y establece XML como norma en el transporte de información en sus productos y lo promociona como tal en los sistemas desarrollados utilizando sus herramientas.

#### 3.1.4.1 .NET Framework

El marco de trabajo .Net (.Net Framework) constituye la base de la plataforma .NET y denota la infraestructura sobre la cual se reúnen un conjunto de lenguajes, herramientas y servicios que simplifican el desarrollo de aplicaciones en un entorno de ejecución distribuido. Los principales componentes del marco de trabajo son:

- El conjunto de lenguajes de programación, la especificación común de lenguajes (**CLS**) y la Infraestructura Común de Lenguajes (**CLI**).
- La Biblioteca de Clases Base (**BCL**).
- El Entorno Común de Ejecución para Lenguajes (**CLR**).

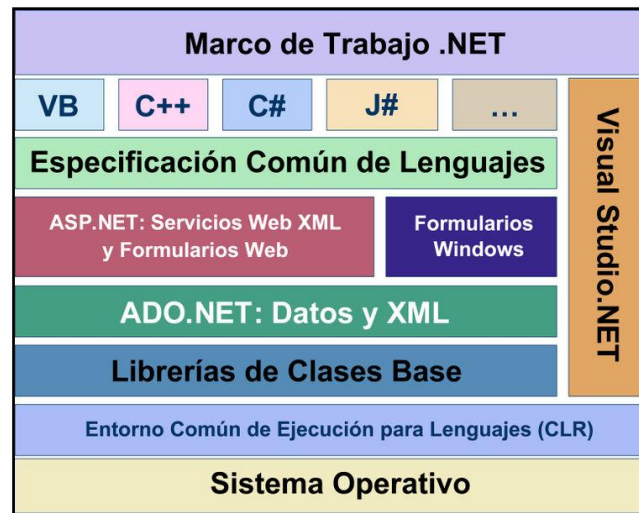


Ilustración 20. .NET Framework

#### 3.1.4.2 *CLI*

Denota el conjunto de normas que debe cumplir un lenguaje de programación para ser compatible con la plataforma. Por medio de estas normas se garantiza que todos los lenguajes desarrollados para la plataforma ofrezcan al programador un conjunto mínimo de funcionalidad y compatibilidad con todos los demás lenguajes.

.NET soporta actualmente más de veinte lenguajes de programación.

#### 3.1.4.3 *CLR*

Es el entorno de ejecución donde se cargan las soluciones desarrolladas en los distintos lenguajes de programación.



Ilustración 21. .NET CLR

La plataforma compila el código fuente de cualquiera de los lenguajes soportados por .NET en un código intermedio *Microsoft Intermediate Language* (MSIL) similar al BYTECODE de Java. Para generar dicho código el compilador se basa en el CLS que determina las reglas necesarias para crearlo.

Para ejecutarse, se necesita un segundo paso: un compilador *Just-In-Time* (JIT) es el que genera el código máquina real que se ejecuta. De esta forma se consigue con .NET independencia de la plataforma hardware.

La compilación JIT la realiza el CLR a medida que el programa invoca métodos, el código ejecutable obtenido se almacena en la memoria caché del ordenador, siendo recompilado de nuevo sólo en el caso de producirse algún cambio en el código fuente.

#### 3.1.4.4 BCL

Se encarga de gestionar las operaciones básicas de bajo nivel y está organizada mediante un sistema de espacios de nombres jerárquico. Se divide en tres grupos principales: ASP .NET y servicios web, Windows Forms y ADO .NET.

#### 3.1.4.5 Ensamblados

Ficheros .EXE o .DLL que contienen toda la funcionalidad de la solución además de los recursos necesarios para su ejecución.



Ilustración 22. Ensamblado .NET

### 3.1.5 ICE

Ice es un middleware de comunicaciones a través de internet, desarrollado por Zeroc, de alto rendimiento y que posee licencia GPL. La característica más importante de Ice es que es muy ligero en comparación con otros middlewares existentes en el mercado.

Ice está disponible para la gran mayoría de plataformas existentes en el mercado, y, posee una versión para su utilización con dispositivos móviles. Esta versión elimina ciertas características imposibles de soportar por las limitaciones de los propios dispositivos. Sin embargo, sigue ofreciendo un modelo de ORB potente y completo.

## 3.2 Toma de decisiones

### 3.2.1 Middleware

Una vez analizadas las diferentes tecnologías de programación distribuida existentes llegamos a la conclusión de que sólo cuatro se adaptaban a las exigencias mínimas del proyecto: OPC, CORBA, Java RMI y ICE.

- **CORBA:** DomoUEX funcionaba bien sobre CORBA. Sin embargo, existen ciertos aspectos que la especificación no cubre. El primer problema es que el middleware es muy pesado y complejo, el segundo (quizás derivado del anterior) es la imposibilidad de utilizar una implementación libre de CORBA sobre dispositivos móviles. Como aspectos positivos cabe destacar que permite la heterogeneidad de lenguajes de programación en la aplicación y que ya contábamos con experiencia en esta tecnología.
- **OPC:** en el momento de comenzar este proyecto OPC era una propuesta experimental. Resultaba muy difícil y costoso encontrar cualquier documentación relacionada. No obstante, llegamos a las siguientes conclusiones: era dependiente de las tecnologías de Microsoft (DCOM) y no se especificaba nada acerca de su uso en dispositivos móviles.
- **Java RMI:** hasta muy avanzado el proceso de análisis aún no habíamos decidido el lenguaje de programación a utilizar y, por sus capacidades, Java RMI era uno de los middlewares a tener en cuenta. El principal inconveniente venía a la hora de integrar Java RMI en dispositivos móviles puesto que KVM (la máquina virtual utilizada por estos dispositivos) no lo soporta.
- **ICE:** aparentemente ICE cubría todas nuestras necesidades puesto que se trata de un middleware de fácil manejo, pensado para entornos heterogéneos donde pueden coexistir diferentes lenguajes, proporciona un middleware reducido para dispositivos móviles (ICE-E) lo suficientemente potente para cumplir con nuestros requisitos y todo esto bajo licencia GPL (libre).

FACTORES	CORBA	OPC	Java RMI	ICE
Complejidad	Alta	Alta	Baja	Baja
Heterogeneidad	Sí	No	No	Sí
Dispositivos móviles	No	No	No	Sí
Licencia	-	Propietario	Libre	Libre
Experiencia	Sí	No	Sí	No

El resultado es claro: para nuestro proyecto, ICE ofrece prestaciones muy superiores al resto de tecnologías por lo que decidimos trabajar con este middleware de programación distribuida. Sin embargo, no todo eran ventajas, puesto que no contábamos con experiencia previa con ICE (ni siquiera conocíamos su existencia) y la única fuente de información era la propia empresa desarrolladora.

### 3.2.2 Plataforma de desarrollo

A la hora de decidir que plataforma utilizar para el desarrollo del proyecto debíamos tener en cuenta las restricciones a las que estábamos sometidos: utilizar sistemas operativos Windows e implementar el acceso al bus en C++.

Teniendo en cuenta estas cuestiones decidimos que .NET era el entorno más adecuado para desarrollar nuestro proyecto, ya que encaja perfectamente con nuestras necesidades, además, nos permite disponer de C# (lenguaje de última generación) para aquellas partes del desarrollo en las que el uso de C++ no sea obligatorio y nos proporciona una plataforma donde se integran multitud de tecnologías que en algún momento podrían servirnos.

### 3.2.3 Instalación domótica

En cuanto al servidor y el cliente de la instalación domótica nos planteamos las siguientes cuestiones:

- Servidor: no tiene que estar cargado con mucha interfaz gráfica, ya que se mejora el rendimiento porque no necesita de esos recursos para su funcionamiento. Debe presentar un fácil manejo de sistemas relacionados con automatización y estar bien preparado para este fin, ya que, se precisa de ello para la comunicación con el bus domótico.
- Cliente: tiene que tener un interfaz gráfico amigable para el usuario y tener a nuestra disposición herramientas potentes para ese fin.

A partir de las premisas anteriores, tomamos las siguientes decisiones:

- Servidor: la implementación se hará utilizando MFC (**M**icrosoft **F**oundation **C**lasses), ya que, de todos los lenguajes disponibles en Visual Studio .NET, C++ es el mejor preparado de todos para encajar en nuestro proyecto y MFC proporciona un conjunto de librerías muy completas para el desarrollo software.
- Cliente: decidimos utilizar C#, ya que es un lenguaje relativamente nuevo y muy potente en lo que necesitamos: desarrollo de una interfaz amigable para el cliente.

### 3.2.4 Videovigilancia

Los principales requisitos a la hora de tratar el sistema de videovigilancia IP son: integración total en el sistema, disponibilidad de varias cámaras, posibilidad de gestionar eventos (como detección de movimiento) y grabar vídeo al darse alguna determinada circunstancia como la activación de una alarma.

Esto nos llevó a investigar en busca de cámaras IP que se amoldasen a nuestras exigencias. Tras evaluar diferentes marcas como *Ovislink*, *PHILIPS*, *RIMAX*... Llegamos hasta las cámaras de red *AXIS*.

Cada cámara *AXIS* cuenta con un sistema operativo UNIX embebido que la hace potente y flexible al contar con funciones de alto nivel muy interesantes (como activación de salidas binarias ante eventos, notificación de alarmas, definición de secuencias de movimiento...) y, a la vez, siendo reprogramable desde los niveles más bajos mediante scripts.

Además, *AXIS* proporciona una API de acceso a las funciones de la cámara vía HTTP (mediante parámetros CGI) y un control ActiveX *Axis Media Control* que nos permite integrar las funciones básicas de las cámaras en una aplicación desarrollada con .NET. Ambos funcionan de forma correcta, están totalmente testeados y ampliamente documentados.

Por estos motivos decidimos hacer uso de esa API para solucionar los problemas planteados en nuestra especificación.

### 3.2.5 Dispositivos móviles

Como ya se ha comentado anteriormente el middleware de distribución escogido para dispositivos móviles es ICE-E. Ya que éste únicamente está disponible para Java y C++ y el entorno a utilizar es .NET, parece lógico decantarse por la segunda opción para aprovechar las facilidades de integración de tecnologías de la plataforma en vez de utilizar otro entorno como *NetBeans* o *Eclipse* y realizar esta parte del proyecto en Java (que, por otro lado, sería una opción totalmente viable puesto que ICE y ICE-E están preparados para sistemas heterogéneos).



### 3.3 ICE

#### 3.3.1 Introducción e historia

A mediados de los 90, los middlewares empezaron a tener auge, ya que distintas empresas comenzaron a desarrollar productos que facilitaban la comunicación remota utilizando programación orientada a objetos.

A continuación se muestran distintas plataformas middlewares desarrolladas desde entonces, junto a una breve descripción de las mismas:

- DCOM: Desarrollado por Microsoft. Presenta problemas de escalado debido a que utiliza un recolector de basura distribuido. Es excesivamente complejo y tiene problemas de incompatibilidades con otros middlewares.
- .Net Remoting: Nueva especificación de DCOM desarrollada por Microsoft, se realizó en el año 2002.
- CORBA: Desarrollada por la OMG, es una especificación excesivamente compleja y que tiene presente ciertas características irrelevantes para el desarrollo de entornos distribuidos. Únicamente se puede utilizar en redes homogéneas y presenta problemas de interoperabilidad.
- JNDI y EJB: Elaborados por la compañía Sun Microsystems y únicamente válidos para su utilización en lenguaje Java. Tienen escaso soporte por parte de la compañía, ya que la JVM no los soporta.
- SOAP / Web Service: Tecnologías desarrolladas por Microsoft. Tienen escasa implantación debido a que poseen bajas prestaciones y provocan una alta sobrecarga en la red. SOAP aporta un nivel de abstracción muy bajo y los Webs Services están en proceso de desarrollo, tienen mucho camino que andar todavía.

Debido a que cada tecnología tenía una carencia en algún aspecto necesario para la programación en entornos distribuidos una empresa, llamada ZeroC, desarrolló Ice. Según sus desarrolladores, la idea inicial fue: “Construyamos un middleware igual de potente que CORBA sin cometer los errores de CORBA”. Sus características principales son:

- Multiplataforma y multilenguaje.
- Pensado para entornos heterogéneos
- Software libre (licencia GPL), abierto a la comunidad.
- Conjunto más completo de características del mercado.
- Fácil de aprender y usar.
- Buen soporte de seguridad.

### 3.3.2 Arquitectura ICE

#### 3.3.2.1 *Introducción a la arquitectura ICE*

ICE es una plataforma middleware orientada a objetos. Proporciona APIs y una serie de librerías que permiten construir sistemas cliente-servidor mediante objetos. El uso de ICE es conveniente en sistemas heterogéneos:

- Cliente y servidor escritos en lenguajes diferentes.
- Sistemas operativos diferentes.
- Arquitecturas hardware diferentes.
- Arquitecturas de red diferentes.

#### 3.3.2.2 *Terminología*

ICE no ha inventado nuevos términos en el ámbito de las comunicaciones mediante middlewares sino todo lo contrario, ha intentado usar toda la terminología existente en este campo.

##### 3.3.2.2.1 *Clientes y Servidores*

Los clientes y servidores no son posiciones fijas de cada extremo dentro de una comunicación, sino que, dependiendo de la finalidad de la comunicación, tanto cliente como servidor puede ser servidor y cliente respectivamente. Por lo tanto, se entiende cliente y servidor como:

- Clientes: entidades activas que solicitan servicios a los servidores.
- Servidores: entidades pasivas que proporcionan servicios a los clientes.

Es difícil encontrar servidores puros que únicamente proporcionen servicios y respondan a peticiones, lo normal es ver servidores como parte de clientes, que ofrecen servicios a ese mismo cliente y a otros que puedan existir. De esta forma, lo más común es que tampoco existan clientes puros, así, vamos a encontrar arquitecturas cliente-servidor híbridas, donde cada extremo solicita y proporciona servicios.

##### 3.3.2.2.2 *Objetos ICE*

Un objeto ICE es una entidad conceptual, que se caracteriza por:

- Es una entidad local o remota que puede responder a las solicitudes del cliente.

- Puede residir en un único servidor o en varios servidores. Si un objeto está replicado en varios servidores, se le considera un único objeto ICE.
- Cada objeto ICE puede tener uno o varios interfaces. Un interfaz es una colección de las operaciones que pueden ser llevadas a cabo por el servidor.
- Una operación puede tener cero o más parámetros. Éstos tienen un tipo específico, igual que el valor retornado por la misma. Los parámetros son inicializados por el cliente y pasados al servidor, y los valores de retorno son inicializados por el servidor y pasados al cliente.
- Cada objeto ICE tiene que tener un interfaz principal y puede proporcionar algún interfaz alternativo, en el que las operaciones que se proporcionan son distintas. El cliente indica qué operación de qué interfaz desea invocar.
- Un objeto ICE tiene un identificador que lo distingue del resto de objetos ICE existentes a escala mundial, por lo tanto, dos objetos nunca van a tener un identificador igual.

#### 3.3.2.2.3 Proxies

Para que un cliente sea capaz de ponerse en contacto con un objeto ICE, es necesaria la utilización de un proxy de comunicación. Un proxy es una representación del objeto ICE, que reside en el servidor, en el espacio de direcciones del cliente, por lo tanto, es la estructura que invoca operaciones en los servidores. Actúa de la siguiente manera:

1. Localiza el objeto ICE.
2. Activa el servidor del objeto, si estuviera desactivado.
3. Activa el objeto ICE en el servidor.
4. Transmite los parámetros.
5. Espera a que se complete la operación.
6. Devuelve los parámetros al cliente o lanza una excepción en caso de error.

El proxy encapsula toda la información necesaria para realizar estos pasos, es decir, contiene la siguiente información:

- Dirección del servidor.
- Identidad del objeto de petición del servicio.

- Identificador de la interfaz sobre la cual se invocará la petición.

Existen varios tipos de proxies, analizaremos a continuación en qué consiste cada uno.

#### *3.3.2.2.3.1 Stringfied proxy*

La información del proxy es pasada a la aplicación en forma de cadena de texto:

Ej: SimplePrinter: default -p 10000

ICE proporciona métodos para la obtención de proxies stringfied desde ficheros y para el almacenamiento de la información del proxy en ficheros.

La información existente en el proxy es transparente al cliente, ya que el cliente tiene que saber la identidad del objeto sobre el cual va a realizar peticiones y dónde se encuentra ese objeto.

#### *3.3.2.2.3.2 Direct proxy*

Un direct proxy se caracteriza porque se determina la identidad del objeto, la dirección de donde reside y el protocolo utilizado para la comunicación de manera directa, es decir, no se utilizan ficheros de texto para su configuración.

#### *3.3.2.2.3.3 Indirect proxy*

Existen dos formas de especificar un proxy indirecto:

1. Identidad del objeto:

Ej: SimplePrinter

2. Identidad del objeto y nombre del adaptador de objetos:

Ej: SimplePrinter@PrinterAdapter

El proxy indirecto no tiene ninguna información de donde se encuentra el objeto, por lo que es necesario un servicio de localización (servicio de nombres).

La ventaja de los proxies indirectos sobre los directos es que nos permiten trasladar objetos de un servidor a otro, ya que estos son encontrados mediante el servicio de

localización y, en los proxies directos, la información sobre la localización del objeto es parte del programa de aplicación.

#### 3.3.2.2.3.4 *Fixed proxy*

Son proxies destinados a una conexión en particular, es decir, en lugar de tener información sobre el adaptador de objetos y el objeto, poseen un manejador de conexión.

El manejador de conexión es válido únicamente mientras dura la conexión entre ambos cabos de la comunicación.

Los proxies fixed no pueden ser pasados como parámetros a invocaciones de operaciones en los servidores (no se permite su serialización). Se utilizan para permitir comunicaciones bidireccionales, así, un servidor no necesita abrir una nueva conexión para realizar callbacks.

#### 3.3.2.2.4 *Replicación*

En ICE, la replicación implica que existan múltiples adaptadores de objetos y objetos en distintas direcciones. Es utilizada para controlar posibles fallos que puedan producirse por la caída de algún servidor.

Debe de existir consistencia entre todos los servidores de réplica, es decir, una llamada a un método de un servidor determinado tiene que devolver lo mismo que la misma llamada a cualquier otro servidor.

A continuación se muestra un ejemplo de réplica con un proxy stringfied:

Ej: SimplePrinter: TCP-h server1-p 10001: TCP-h server2-p 10002

En este ejemplo, el objeto SimplePrinter está alojado en los servidores server1 y server2, y en los puertos 10001 y 10002 respectivamente.

#### 3.3.2.2.5 *Replica groups*

Existe otro método de replicación en ICE, los replica groups, que requieren la utilización de un servicio de localización.

En este método de replicación, un grupo consiste en un único identificador y un número indeterminado de adaptadores de objeto. A su vez, un adaptador puede ser miembro de varios replica groups.

Tras construir el replica group, su identificador puede ser utilizado por un indirect proxy en lugar del identificador de adaptador de objeto. Por ejemplo, un replica group identificado como PrinterAdapters puede ser utilizado por un proxy de la siguiente forma:

SimplePrinter@PrinterAdapters

El grupo es tratado por el servicio de localización como un “adaptador de objetos virtual”.

Cómo resuelve la dirección el servicio de localización es una cuestión de implementación. Por ejemplo, podría devolver la dirección de todos los adaptadores del grupo o sólo una en función de algún parámetro deseado.

#### 3.3.2.2.6 [Servants](#)

Como se mencionó anteriormente, un objeto ICE es una entidad conceptual que tiene un identificador, una dirección y un tipo. Para que un cliente pueda realizar peticiones a un servidor los servicios tienen que estar implementados en la parte del servidor, con lo que deben tener una estructura donde ser implementados, esta estructura es el servant.

En resumen, un servant, es una instancia de una clase implementada por el desarrollador en el servidor, donde los métodos son las operaciones existentes en el interfaz del objeto.

Un servant puede encarnar un único objeto o varios objetos simultáneamente. En el primer caso la identidad del objeto está implícita en el servant, en el segundo, se debe informar al servant en cada petición del cliente a qué objeto queremos acceder para que lo encarne durante la duración de la petición.

A la inversa, un objeto puede tener varios servants. Por ejemplo, podríamos crear un proxy para un objeto con dos direcciones de máquinas diferentes (tendríamos que implementar una política de decisión de qué servidor llamar primero y, en caso de error, a cuál llamar después). En ese caso, tendríamos dos servidores con un servant cada uno para el mismo objeto, por lo tanto, se obtiene replicación a partir de la existencia de diversos servants.

#### 3.3.2.2.7 [Como máximo una vez](#)

ICE proporciona la política de realizar una operación una sola vez como máximo. Cuando una petición es enviada a un servidor, si no tiene éxito, se devuelve una excepción al cliente, pero, en ningún caso, una petición es ejecutada dos veces a no ser que se tenga la certeza absoluta de que el anterior intento fue completamente fallido.

Existen operaciones que son del tipo idempotencia, es decir, que si ejecutamos la operación dos veces, el resultado final es el mismo que tras ejecutarlo una única vez. Por ejemplo, una operación que fuera  $x = 1$ , sería idempotente, pero una operación  $x = x+1$  no sería idempotente. La política de “como máximo una vez” garantiza que éstas últimas se ejecuten de forma segura.

La mayoría de sistemas que se construyen requieren operaciones no idempotentes por lo que esta política, aunque menos flexible, es necesaria para asegurar la fiabilidad.

ICE permite marcar operaciones como idempotentes, para estas operaciones el mecanismo de recuperación de errores es más agresivo que para las no idempotentes.

#### 3.3.2.2.8 [Invocación de métodos síncrona](#)

Por defecto, el modelo de petición usado por ICE es el de llamada a procedimiento síncrona, es decir, una invocación de operación se comporta como una llamada de procedimiento local, por lo que el hilo del cliente es suspendido durante la ejecución de la llamada y continúa cuando la llamada termina y todos sus resultados están disponibles.

#### 3.3.2.2.9 [Invocación de métodos asíncrona](#)

ICE también da soporte para la invocación de métodos asíncrona (AMI), en la que los clientes pueden invocar operaciones de forma asíncrona utilizando un proxy para invocar la operación, añadiendo un objeto callback a los parámetros y retornando inmediatamente. Una vez que la operación esté completada, el soporte de ejecución del servidor invoca un método en el objeto callback informando del resultado.

Los servidores no pueden distinguir entre invocaciones síncronas o asíncronas, ya que, simplemente ven que un cliente ha invocado una operación sobre un objeto que reside en ellos.

#### 3.3.2.2.10 [Expedición de métodos](#)

La expedición de métodos es el equivalente en el lado del servidor a la invocación de métodos por parte del cliente: hace referencia a cómo trata el servidor las peticiones realizadas por los clientes.

Las peticiones de los clientes pueden tratarse de dos maneras distintas:

- **Síncrona:** Cada vez que el servidor recibe una petición por parte de un cliente, se crea un hilo que ejecuta dicha petición. El hilo muere cuando se termina de atender la petición.

- **Asíncrona:** Cuando se recibe una petición de un cliente, el servidor decide si se retrasa la atención de dicha petición, es decir, el servidor no está obligado a tratar la petición nada más llegar. En el caso de que no se trate nada más llegar, se libera el hilo creado para su ejecución. Una vez que los resultados de la operación están disponibles, el servidor hace una llamada a la API del servidor de ICE para informar que una petición retrasada fue atendida y que informe al cliente.

La expedición de métodos asíncrona es útil cuando un servidor ofrece operaciones que bloquean a clientes durante un periodo amplio de tiempo, con lo que los hilos de las operaciones que se atenderán en un futuro, o sea, de las retrasadas, son eliminados y la carga del servidor disminuida.

Si se utilizase una expedición síncrona, cada cliente crearía un hilo en el servidor, con lo que el rendimiento en el servidor sufriría un bajón debido al número de hilos bloqueados.

Ambos métodos de expedición de métodos son transparentes al cliente, es decir, el cliente no puede saber si un servidor decidió tratar una operación sincrónicamente o asincrónicamente.

#### 3.3.2.2.11 Invocación de métodos unidireccional

Esta forma de invocación de métodos por parte del cliente tiene las siguientes características:

- La invocación se realiza en cuanto llega a la capa de transporte.
- No hay respuesta desde el servidor.
- Método válido únicamente si las operaciones no tienen valor de retorno y que no lancen excepciones al cliente.
- Sólo válido cuando se usan protocolos TCP y SSL.
- La ejecución puede ocurrir en un orden no deseado, ya que el orden de atención viene dado por el manejador de hilos del servidor, y no por el orden de llegada desde la red.

Las invocaciones de un solo sentido no son fiables, ya que al no retornar valores ni poder capturar excepciones no se sabe si la operación se ha podido llevar a cabo.

Este tipo de invocación es transparente al servidor, ya que no puede distinguir entre una invocación bidireccional y una unidireccional.



#### 3.3.2.2.12 Invocación de métodos unidireccional por lotes

Cada invocación unidireccional envía un mensaje separado al servidor, si se envían mensajes cortos en intervalos muy pequeños de tiempo se crea un tráfico alto en la red.

Mediante el método de lotes se permite enviar una serie de invocaciones unidireccionales como un único mensaje, es decir, al realizar una invocación unidireccional, ésta es almacenada en un buffer que, pasado un periodo de tiempo, se envía entero al cliente. El buffer puede enviarse realizando una llamada a la API de ICE.

Las invocaciones son atendidas por un solo hilo y, cuando el servidor recibe el paquete, las procesa individualmente en el orden en el que fueron introducidas en el buffer.

Este método es útil para servicios de mensajería, como es el servicio de notificación de eventos.

#### 3.3.2.2.13 Invocación por datagramas

Las invocaciones de datagramas tienen las mismas características que las invocaciones unidireccionales, no pueden retornar valores, no pueden existir parámetros de entrada/salida y no pueden ser transmitidas excepciones al cliente.

Requieren que el protocolo de transporte sea UDP, con lo que, al no ser orientado a conexión, no se garantiza que el paquete llegue a su destino pero se gana en velocidad. Este método de invocación es ideal para entornos donde la tasa de errores en la red es baja y en donde la velocidad es muy importante.

Las invocaciones basadas en datagramas no son fiables, además de los explicados en el párrafo anterior, hay que añadir estos casos:

- El objeto al que se hace la invocación puede no existir.
- El servidor puede que no esté activo.
- La llegada puede que no sea en orden. Esto es debido a la utilización del protocolo UDP como protocolo de transporte, ya que, los datagramas pueden tomar caminos distintos para llegar a su destino.

#### 3.3.2.2.14 Invocación por datagramas por lotes

Al igual que para invocaciones en un único sentido, ICE puede realizar las invocaciones basadas en datagramas por lotes, es decir, agrupar las invocaciones en un paquete y enviarlas todas juntas al servidor realizando una llamada a la API de ICE.

Gracias a este método, se reduce el elevado número de llamadas y permite a la red funcionar de manera más eficiente.

Realizar una invocación por lotes en sistemas basados en datagramas tiene una restricción, y es que el tamaño del paquete de invocaciones enviadas al servidor no puede exceder del tamaño máximo de paquetes de la red por la que se envía, ya que, si se fragmenta el paquete, aumenta la probabilidad de que se pierdan los fragmentos debido al uso de UDP.

Con este método se garantiza que o se realizan todas las invocaciones, o no se realiza ninguna, con lo cual, si se realizan todas, se garantiza que se haga en orden.

#### 3.3.2.2.15 Excepciones en tiempo de ejecución

Cualquier invocación de operación puede hacer saltar una excepción durante la ejecución de la aplicación. Las excepciones en tiempo de ejecución están predefinidas en ICE y cubren condiciones de error comunes, como el fracaso de conexión, la interrupción de conexión, o el fracaso de asignación de recurso.

Están implementadas para C++, Java y C#, y se integran fácilmente en la implementación de una aplicación gracias a las características de estos lenguajes.

#### 3.3.2.2.16 Excepciones de usuario

Las excepciones de usuario son usadas para indicar condiciones de error de aplicación. Además, pueden llevar una cantidad arbitraria de datos complejos y pueden delegar en jerarquías de herencia, que hacen fácil para los clientes poder manejar las categorías de errores, cogiendo una excepción que está por encima de la jerarquía de herencia.

#### 3.3.2.2.17 Propiedades

La mayor parte de ICE se configura vía propiedades. Las propiedades son pares de valor y nombre.

Ej: Protocol=tcp

Las propiedades frecuentemente son almacenadas en archivos de texto y analizadas por ICE en tiempo de ejecución.

#### 3.3.2.3 *SLICE*

Como se mencionó anteriormente, cada objeto ICE tiene un interfaz con una serie de operaciones. Los interfaces, operaciones y los tipos de los datos que son transmitidos entre el cliente y el servidor son definidos usando el lenguaje SLICE.

SLICE permite definir la interacción entre cliente y servidor de una forma independiente al lenguaje de programación utilizado. Las definiciones de SLICE son compiladas en una API para el lenguaje de programación específico.

#### 3.3.2.4 *Language Mappings*

Las reglas que gobiernan cómo cada archivo SLICE es traducido a un lenguaje de programación específico son conocidas como language mappings. Por ejemplo, para el mapeado de C++, una secuencia SLICE aparece como un vector STL, mientras que, para Java, lo hace como un Java array.

Para determinar cómo es la API generada desde un determinado archivo SLICE, sólo se necesita la definición SLICE y un cierto conocimiento sobre las reglas de mapeado. Las reglas son tan simples y regulares que hacen innecesario leer el código generado para saber cómo usar la API.

Este código generado es bastante complejo y no es recomendable gastar tiempo en analizarlo (aunque, desde luego, cualquiera es libre para poder hacerlo). La mejor opción es familiarizarse con las reglas de mapeo, así, se podrá ignorar el código generado automáticamente.

Actualmente ICE provee language mappings para C++, Java, C#, Visual Basic .NET, Python y, sólo para el lado cliente, PHP y Ruby.

## 3.3.2.5 Estructura Cliente-Servidor

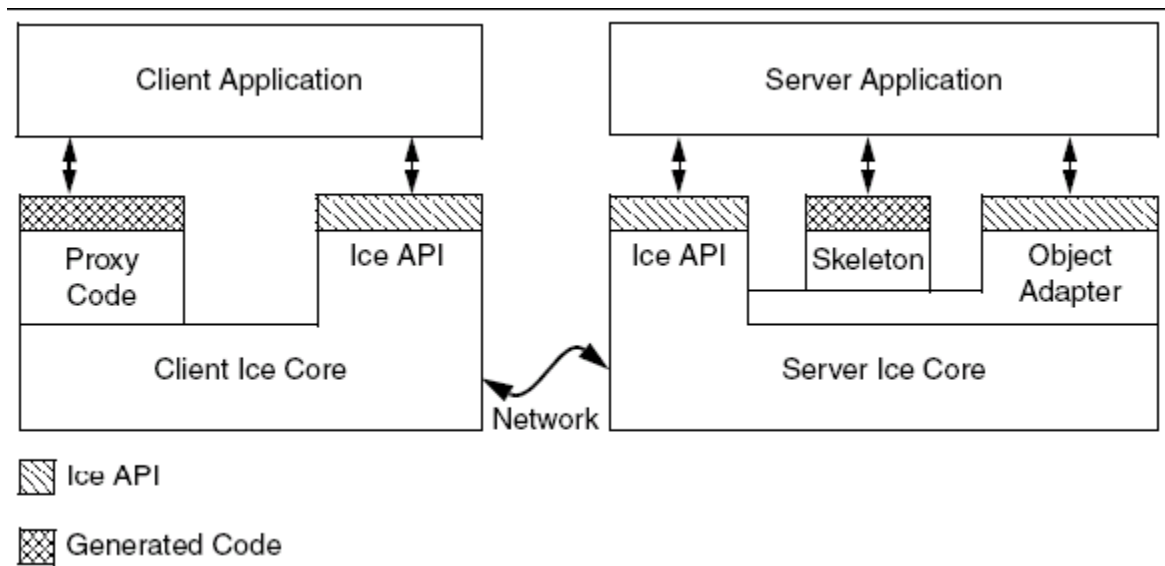


Ilustración 23. Arquitectura ICE

Tanto el cliente como el servidor consisten en una mezcla de código de aplicación, de biblioteca, y código generado por las definiciones SLICE:

- El núcleo de ICE contiene soporte en ejecución para comunicaciones remotas tanto para el cliente como para el servidor. La mayor parte de este código se encarga de los detalles de networking, hilos, ordenación de bytes y otras cuestiones relacionadas con la interconexión que queremos mantener lejos del código de aplicación. El núcleo de ICE se distribuye como una serie de bibliotecas que pueden ser utilizadas por el cliente y por el servidor.
- La parte genérica del núcleo de ICE (es decir, la parte que es independiente de los tipos específicos definidos en SLICE) es accesible desde la API de ICE. Esta API se utiliza para labores de administración como inicialización y finalización del soporte de ejecución. Es idéntica para cliente y servidor (aunque los servidores utilizarán un mayor número de funciones).
- El código del proxy es generado a partir de las definiciones SLICE y es específico a los tipos y objetos definidos. Tiene dos funciones principales:
  - Proporciona un interfaz de bajo nivel para el cliente. La llamada a una función de la API del proxy generado termina enviando un mensaje RPC al servidor que invoca una función determinada sobre el objeto deseado.
  - Proporciona marshaling y unmarshaling: Marshaling es el proceso de serializar una estructura de datos compleja, para la transmisión sobre el cable.

Convierte datos en una forma estandarizada para la transmisión e independiente de la representación y la reglas de relleno de la máquina local. Unmarshaling es el proceso inverso, es decir, deserializar los datos que llegan desde la red y reconstruir una representación local de los datos en tipos adecuados al lenguaje de programación utilizado.

- El código de skeleton también es generado a partir de la definición SLICE y, por tanto, específico a los tipos de objetos y datos definidos. Es el equivalente del lado servidor al código de proxy del lado cliente: proporciona un interfaz de alto nivel que permite al soporte de ejecución ICE transferir el hilo de control al código de aplicación. El skeleton también contiene marshaling y unmarshaling, así, el servidor puede recibir parámetros enviados por el cliente, y devolver parámetros y excepciones.
- El adaptador de objeto es una parte de la API de ICE específica del lado servidor: sólo los servidores usan adaptadores de objeto. Tiene varias funciones:
  - Asigna peticiones entrantes de los clientes a métodos específicos de objetos. En otras palabras, rastrea qué servants hay en memoria y qué identificadores de objeto tienen.
  - Se asocia a uno o varios endpoints de transporte. Si más que un endpoint es asociado a un adaptador, los servants que encarnan objetos dentro del adaptador pueden ser alcanzados desde cualquiera de ellos. Por ejemplo, se puede asociar un endpoint tanto TCP/IP como UDP con un adaptador, para proporcionar diferentes calidades de servicio y características de funcionamiento.
  - Es responsable de la creación de los proxies que pueden ser pasados a clientes. El adaptador de objeto conoce el tipo, la identidad, y los detalles de transporte de cada uno de sus objetos e integra todos estos detalles cuando el código de aplicación del lado servidor solicita la creación de un proxy.

En definitiva, desde el punto de vista de la aplicación, sólo existen dos procesos: cliente y servidor. Todo lo demás es proporcionado por las librerías de ICE y el código generado por la definición SLICE.

#### 3.3.2.6 *El protocolo ICE*

ICE proporciona un protocolo RPC que puede usar TCP/IP o UDP como transporte subyacente.

Además, ICE también permite usar el protocolo de transporte seguro SSL (toda la comunicación entre el cliente y el servidor es cifrada).

El protocolo ICE define:

- Tipos de mensaje, como petición y tipos de mensaje de respuesta.
- Una máquina de estados que determina en que secuencia son intercambiados los diferentes tipos de mensaje entre cliente y servidor, junto con la semántica de establecimiento de conexión y desconexión para el protocolo de transporte utilizado.
- Las reglas de codificación que determinan como cada tipo de datos es representado en el cable.
- Una cabecera para cada mensaje que contiene detalles como el tipo, el tamaño, el protocolo y la versión de codificación empleados.

ICE también soporta compresión de datos en el cable: utilizando un parámetro de configuración, se puede pedir que todo el tráfico de red sea comprimido para conservar el ancho de banda. Esto es útil si la aplicación genera un volumen de tráfico alto.

ICE es una herramienta perfecta para construir mecanismos eficientes de expedición de eventos porque permite enviar mensajes sin tener que analizar la información que va dentro. Es decir, los conmutadores de mensajes no realizan marshaling ni unmarshaling, tratando los mensajes como un simple buffer de bytes opaco y ahorrando bastante tiempo.

El protocolo ICE también soporta el modo de operación bidireccional: si un servidor quiere enviar un mensaje a un objeto callback proporcionado por el cliente, el callback puede ser realizado sobre la conexión que al principio fue creada por el cliente. Esto es sobre todo importante cuando el cliente está detrás de un firewall que permite conexiones salientes, pero no conexiones entrantes.

#### 3.3.2.7 *Persistencia de objetos*

ICE tiene un servicio de persistencia de objetos llamado Freeze. Freeze hace fácil almacenar el estado de un objeto en una base de datos: se define el estado almacenado por los objetos en SLICE, y el compilador de Freeze genera el código que almacena y recupera el estado del objeto en y desde una base de datos.

Por defecto, Freeze usa Berkeley DB como base de datos. No obstante, si se prefiere almacenar el estado del objeto en otra base de datos, puede hacerse sin problemas. ICE también ofrece una serie de instrumentos que hacen más fácil mantener bases de datos y migrar el contenido de bases de datos existentes a un nuevo esquema si las definiciones de tipo de objetos se cambian.

### 3.3.3 Servicios ICE

El núcleo de ICE proporciona una sofisticada plataforma cliente-servidor para el desarrollo de aplicaciones distribuidas. Sin embargo, las aplicaciones reales por lo general requieren más que una capacidad remoting: normalmente, también se necesita un modo de levantar servidores en demanda, distribuir proxies a clientes, distribuir eventos asincrónicos, configurar su uso, distribuir parches...

ICE incluye una serie de servicios que proporcionan estas y otras características. Los servicios se implementan como servidores de ICE a los que la aplicación accederá como cliente.

Ninguno de los servicios usa las funciones internas de ICE, que son opacas a los desarrolladores, es decir, cualquiera podría desarrollar servicios equivalentes. Sin embargo, tener estos servicios disponibles como parte de la plataforma permite centrarse en el desarrollo de la propia aplicación sin necesidad de construir mucha infraestructura antes. Además, construir estos servicios no es un esfuerzo trivial.

#### 3.3.3.1 *IceGrid*

IceGrid es una implementación de un servicio de localización de ICE que resuelve la información simbólica en un proxy a una dirección de protocolo para binding indirecto. Pero esto es sólo el principio de las posibilidades de IceGrid:

- Permite registrar servidores para arranque automático: en vez tener un servidor levantado continuamente, cuando un cliente emita una primera petición, IceGrid despierta a los servidores necesarios.
- Proporciona herramientas que hacen más fácil configurar aplicaciones complejas que contienen varios servidores.
- Soporta la replicación y el equilibrio de carga.
- Automatiza la distribución y actualización de ejecutables y ficheros dependientes.
- Proporciona un servicio de pregunta simple que permite a los clientes obtener proxies para objetos en los que están interesados.

#### 3.3.3.2 *IceBox*

IceBox es un servidor de aplicaciones que puede orquestar el comienzo y la finalización de un conjunto de componentes de aplicación. Los componentes de aplicación pueden ser desplegados como una librería dinámica en vez de como un proceso. Esto reduce la carga total

del sistema, por ejemplo, permitiendo controlar varios componentes de aplicación en una única máquina virtual Java en vez de tener múltiples procesos, cada uno con su propia máquina virtual.

#### 3.3.3.3 *IceStorm*

IceStorm es un servicio de publicación-suscripción para clientes y servidores. Fundamentalmente, IceStorm actúa como un conmutador de distribución de eventos.

Los publicantes envían eventos al servicio que pasa los acontecimientos a los suscriptores. De este modo, un sólo evento publicado puede ser enviado a múltiples suscriptores. Los eventos son clasificados por tema, y los suscriptores especifican los temas en los que están interesados. Sólo los eventos cuyo tema coincida con sus intereses son enviados a los suscriptores. El servicio permite la selección de distintos criterios de calidad de servicio para permitir a las aplicaciones escoger la medida adecuada entre fiabilidad y rapidez.

IceStorm es en particular útil si se tiene la necesidad de distribuir información a muchos componentes de aplicación (un ejemplo típico es el de una tele impresora con un número grande de suscriptores). Además, IceStorm puede ser ejecutado como un servicio federado, es decir, múltiples instancias del servicio pueden ejecutarse sobre máquinas diferentes para repartir la carga de trabajo.

#### 3.3.3.4 *IcePatch2*

IcePatch2 es un servicio de actualización de software. Permite distribuir fácilmente actualizaciones entre los clientes que sólo tienen que conectarse al servidor de IcePatch2 para solicitar actualizaciones de un aplicación concreta.

El servicio automáticamente comprueba la versión del software del cliente y transfiere cualquier componente de aplicación actualizado en un formato comprimido para aprovechar mejor el ancho de banda. Los parches de software pueden ser asegurados usando el servicio de Glacier2, entonces sólo los clientes autorizados pueden acceder a las actualizaciones.

#### 3.3.3.5 *Glacier2*

Glacier2 es el servicio de firewall de Ice: permite a clientes y servidores comunicarse por un firewall sin comprometer la seguridad. El tráfico entre servidor y cliente es totalmente cifrado usando certificados de clave pública y es bidireccional. También ofrece apoyo a la autenticación mutua y el control de sesión seguro.



### 3.3.4 Beneficios de la arquitectura ICE

La arquitectura ICE proporciona una serie de ventajas a los desarrolladores:

- Semántica Orientada a Objeto:

ICE conserva totalmente el paradigma orientado a objeto.

- Soporte para comunicación síncrona y asíncrona:

Esto permite escoger un modelo de comunicación según las necesidades de uso.

- Soporte para múltiples interfaces:

Mediante facets, los objetos pueden proporcionar múltiples interfaces conservando una identidad de objeto única. Esto proporciona una gran flexibilidad, en particular cuando una aplicación evoluciona pero tiene que seguir manteniendo la compatibilidad con clientes antiguos.

- Independencia de la máquina:

Los clientes y servidores son independientes de la arquitectura subyacente. Factores como la ordenación de bytes o la técnica de relleno son transparentes al código de la aplicación.

- Independencia del lenguaje de programación:

Clientes y servidores puede ser desarrollados en cualquier lenguaje, incluso no tienen por qué compartirlo. La definición de SLICE usada tanto por el cliente como por el servidor establece el interfaz entre ellos y es la única cosa sobre la que tienen que estar de acuerdo.

- Independencia de la implementación:

Los clientes no se preocupan de cómo los servidores implementan sus objetos. Esto quiere decir que una vez construidos los clientes, el servidor puede modificarse siempre que siga respetando el interfaz acordado (por ejemplo para usar un sistema de persistencia diferente o incluso para construir el servidor en un nuevo lenguaje de programación).

- Independencia del sistema operativo:

ICE es totalmente portátil, el código fuente compila sobre Windows y UNIX.

- Soporte para hilos:

El run-time de ICE está compuesto por hilos y la API ofrece soporte para éstos.

- Independencia del transporte:

Actualmente ofrece tanto TCP/IP como UDP como protocolos de transporte. Ni el cliente ni el servidor son conscientes del transporte subyacente (el transporte deseado puede ser escogido mediante un parámetro de configuración).

- Localización y servidores transparentes:

El run-time de ICE se encarga de localizar objetos y gestionar el mecanismo de transporte. Las interacciones cliente-servidor parecen independientes de la conexión. Mediante IceGrid los servidores pueden migrar a diferentes direcciones físicas sin afectar a los proxies de los clientes y estos no deben preocuparse de como se distribuyen las implementaciones de los objetos en los que están interesados.

- Seguridad:

Las comunicaciones entre cliente y servidor pueden asegurarse totalmente con cifrado fuerte sobre SSL, así, las aplicaciones pueden utilizar redes públicas sin garantías para comunicarse de forma segura. Además, como ya hemos visto anteriormente podemos usar Glacier2 como herramienta firewall.

- Persistencia:

Mediante Freeze.

- Código fuente público:

El código fuente de ICE es público. Aunque no es necesario conocerlo para construir aplicaciones, es accesible a todo el mundo y puede ser modificado.

En general, ICE proporciona una plataforma para el desarrollo de aplicaciones distribuidas más completa que cualquier otra que conozcamos.

## 3.4 ICE-E

### 3.4.1 ¿Qué es ICE-E?

Ice-E ("Embedded Ice") es un middleware de comunicaciones diseñado específicamente para entornos con recursos limitados como móviles Smartphone, PDAs y controladores embebidos.

Está disponible para Java y C++ y, básicamente, se trata de un subconjunto de las opciones proporcionadas por ICE.

### 3.4.2 Plataformas soportadas

La siguiente tabla indica las plataformas y compiladores donde puede utilizarse ICE-E.

## C++

Plataforma	Compilador
Fedora Core 4 Linux i386	GCC 4.0.2
Fedora Core 4Linux x86_64	GCC 4.0.2
Red Hat Enterprise Linux 4.2 i386	GCC 3.4.4
Red Hat Enterprise Linux 4.2 x86_64	GCC 3.4.4
Windows 2000/XP	Visual C++ 6.0 SP5 y STLport 4.5.3 Visual Studio .NET 2003 Visual Studio 2005
Pocket PC 2003 / Smartphone 2003 (Windows CE 4.2)	Visual Studio 2005 con Smart Device embedded VC++ 4.0 SP4 y STLport 5.0.2

## Java

Plataforma	Compilador
Linux Fedora Core 4	JDK 1.1.8 JDK 1.4.2 Sun WTK 2.2
Windows 2000/XP	JDK 1.1.8 JDK 1.4.2 Sun WTK 2.2
Pocket PC 2003 (Windows CE 4.2)	CrEme 3.27
Palm OS 5	WebSphere Micro Environment
Nokia Series 40 2nd edition	Series 40 Platform 2.0 SDK

Nokia Series 60 2nd edition

Series 60 Platform 2.0 SDK

Nokia Series 80 2nd edition

Series 80 Platform 2.0 SDK

### 3.4.3 Comparativa con ICE

Para reducir las librerías, algunas de las opciones de ICE han sido eliminadas y otras son opcionales.

**Modelos de concurrencia:** ICE-E sólo soporta el modelo de concurrencia de un hilo por conexión en el lado servidor. Así, aplicaciones que utilicen callbacks anidados deben tener en cuenta que:

- Para conexiones bidireccionales, los callbacks anidados pueden ser bidireccionales.
- Sólo se permite un nivel de anidamiento de callbacks.

Respecto al cliente, ICE-E también soporta el modelo de concurrencia bloqueante, pero si utilizamos este modelo la comunicación bidireccional no está permitida.

**Transporte:** ICE-E sólo dispone de TCP como protocolo de transporte. No es posible utilizar SSL o UDP.

**Objetos por valor:** No se permite el paso de clases SLICE por valor de forma remota.

**Servicio de localización:** No se permiten servicios de localización de servant en ICE-E.

**Otras características eliminadas:** invocación y expedición de métodos asíncronos, APIs de streaming, compresión de datos...

**Opcionales:** soporte para routers (necesario para usar Glacier2), localizadores (necesarios para proxies indirectos) y comunicación por lotes.

A pesar de estas diferencias, ICE y ICE-E son totalmente compatibles, una aplicación desarrollada en ICE puede comunicarse sin problemas con una desarrollada en ICE-E o, incluso, podemos desarrollar parte de nuestra aplicación en uno y parte en otro.

## 3.5 ICE vs CORBA

Obviamente, ICE usa muchas ideas que pueden ser encontradas en CORBA y en plataformas de programación distribuida anteriores como DCE. En algunas áreas, ICE es notablemente parecido a CORBA mientras que, en otras, las diferencias son profundas y tienen grandes implicaciones. Es importante ser consciente de estas diferencias.

### 3.5.1 Diferencias en el Modelo de Objetos

El modelo de objetos ICE, aún cuando superficialmente parece el mismo, se diferencia en un número de puntos importantes del modelo de objeto de CORBA.

#### 3.5.1.1 *Tipo de Sistema*

Un objeto ICE, como un objeto CORBA, tiene exactamente un interfaz principal.

Sin embargo, un objeto ICE puede proporcionar otros interfaces como facets. Es importante hacer notar que todas las facets de un objeto ICE comparten la misma identidad de objeto, es decir el cliente ve un único objeto con múltiples interfaces en vez de varios objetos, cada uno con un interfaz diferente.

Las facets proporcionan gran flexibilidad arquitectónica. En particular, ofrecen un acercamiento al problema de versionado: es fácil ampliar la funcionalidad en un servidor sin romper clientes ya desplegados simplemente añadiendo una nueva faceta a un objeto ya existente.

#### 3.5.1.2 *Semántica Proxy*

Los proxies de ICE (el equivalente de referencias de objeto de CORBA) no son opacos. Los clientes siempre pueden crear un proxy sin el soporte de cualquier otro componente de sistema, mientras ellos conozcan el tipo y la identidad del objeto. Permitir a los clientes crear proxies en demanda tiene una serie de ventajas:

- Los clientes pueden crear proxies sin necesidad de consultar un look-up externo. En efecto, la identidad de objeto y el nombre del objeto, se consideran uno mismo e idénticos. Esto elimina los problemas que pueden provenir de perder el servicio de nombres por algún motivo y reduce la cantidad de componentes necesarios para el correcto funcionamiento del cliente y el servidor.
- No es necesario disponer de diferentes codificaciones de proxies stringified. Una única representación es suficiente y, además, esta es legible. Esto evita la complejidad introducida por las referencias a objetos de CORBA (IOR, corbaloc y corbaname).

La experiencia durante muchos años con CORBA ha revelado que, pragmáticamente, la opacidad de referencias de objeto es problemática: no sólo requiere APIs más complejas y el apoyo de soporte en ejecución, también afecta a la hora de construir sistemas.

Por eso, los mecanismos como corbaloc y corbaname fueron añadidos, así como `is_equivalent` (mal definido) y operaciones de hash para la comparación de referencias. Todos estos mecanismos comprometen la opacidad de referencias de objeto, pero otras partes de la plataforma CORBA todavía tratan de mantener la ilusión de referencias opacas. Así, el desarrollador se encuentra con lo peor de ambas opciones: las referencias son, ni totalmente opacas, ni totalmente transparentes, la confusión resultante y la complejidad son considerables.

#### 3.5.1.3 *Identidad de Objeto*

El modelo de objeto ICE asume que las identidades de objeto son mundialmente únicas (pero sin imponer esta exigencia al desarrollador de aplicaciones). La ventaja principal de identidades de objeto mundialmente únicas consiste en que permiten migrar servidores y combinar los objetos en múltiples servidores separados en un sólo servidor sin preocupaciones sobre colisiones de nombre: si cada objeto ICE tiene una identidad única, es imposible para ésta chocar con la identidad de otro objeto en un dominio diferente.

El modelo de objeto ICE también usa la identidad de objeto fuerte: es posible determinar si dos proxies denotan el mismo objeto como una operación local del lado cliente (con CORBA, se deben invocar operaciones sobre los objetos remotos para conseguir la comparación de identidad fiable). La comparación de identidad local es mucho más eficiente y crucial para algunos dominios de aplicación, como un servicio de transacción distribuido.

### 3.5.2 *Diferencias en el soporte de Plataformas*

CORBA, dependiendo de qué especificación escojamos, proporciona muchos de los servicios proporcionados por ICE. Por ejemplo, CORBA soporta la invocación de métodos asíncronos y, con el modelo de componentes, una forma de múltiples interfaces.

Sin embargo, el problema es que, normalmente, es imposible encontrar estas características en una única implementación. Demasiadas especificaciones CORBA son opcionales o no extensamente implementadas, como desarrolladores, podemos enfrentarnos con no disponer de un servicio concreto.

Otras características de ICE que no tienen equivalentes directos en CORBA:

- Expedición de métodos asíncrona (AMD): CORBA no proporciona ningún mecanismo para suspender el tratamiento de una operación en el servidor, liberar el hilo de control y continuar la operación más tarde.
- Seguridad: hay muchas páginas de especificaciones relacionadas con la seguridad, pero muchas permanecen sin poner en práctica hasta el momento. CORBA, actualmente, no ofrece ninguna solución para coexistir con un firewall.
- Características de Protocolo: el protocolo ICE ofrece soporte bidireccional, que es una exigencia fundamental para permitir callbacks a través de firewall (CORBA especificó un protocolo bidireccional una vez, pero la especificación técnicamente no fue válida y, hasta donde sabemos, nunca fue puesta en práctica). Además, ICE permite utilizar UDP, así, la distribución de eventos sobre redes confiables (locales) puede ser ligera y eficiente. CORBA no proporciona ningún soporte para UDP. Otro rasgo importante de los protocolos ICE es que todos los mensajes y datos son totalmente encapsulados en el cable. Esto permite a ICE implementar servicios, como IceStorm, extremadamente eficientes porque, para transmitir datos, no es necesario el proceso de marshalling y unmarshalling. También es importante para el despliegue de protocolos de puente como Glacier2, porque el puente no tiene por qué ser configurado con un tipo definido de información.
- Language mappings: CORBA no especifica language mappings para C#, Visual Basic o PHP.

### 3.5.3 Diferencias en la complejidad

CORBA es conocido como una plataforma grande y compleja. Esto es en gran parte resultado de que CORBA es estandarizado: las decisiones son alcanzadas según el acuerdo general y el voto de la mayoría.

En la práctica, esto significa que, cuando una nueva tecnología está siendo estandarizada, el único modo de alcanzar el acuerdo es acomodar las características favoritas de todas las partes interesadas. El resultado es que las especificaciones son grandes y complejas.

Toda esta complejidad conduce a implementaciones que son grandes e ineficaces. La complejidad de las especificaciones es reflejada en la complejidad de la API de CORBA: hasta los expertos con años de experiencia todavía tienen que trabajar con un manual de referencia cerca y, debido a esta complejidad, las aplicaciones con frecuencia son amenazadas por fallos latentes que no se descubren hasta el final del despliegue.

El modelo de objetos de CORBA añade complejidad. Por ejemplo, referencias de objeto opacas fuerzan la especificación de un servicio de nombres porque los clientes deben tener algún modo de tener acceso a las referencias de objeto. Esto requiere que el desarrollador aprenda otra API, y que configure y despliegue otro servicio cuando, con el modelo de objeto de ICE, ningún servicio de nombres es necesario.

Una de las áreas más infames de complejidad en CORBA es el mapeo de C++. El API de CORBA para C++ es misterioso en extremo, en particular, las cuestiones de direccionamiento de memoria es más de lo que muchos desarrolladores están dispuestos a soportar. Es más, el código requerido para implementar el mapeo C++ no es ni pequeño ni eficiente, llevándonos a binarios más grandes de lo necesario que consumen memoria en exceso. Si has usado CORBA con anterioridad apreciarás la simplicidad, eficiencia e integración con STL del mapeado de ICE para C++.

En contraste con CORBA, ICE es ante todo una plataforma simple. Los diseñadores de ICE tuvieron gran cuidado para escoger una serie de características que es suficiente y mínima: se puede hacer todo lo que queramos con la API más simple posible. Esto hace más fácil comprender y utilizar la plataforma y conduce a acortar la duración del desarrollo y los errores.



## 4 Análisis y diseño

Una vez identificados los problemas a solucionar, podemos centrarnos en cómo afrontar su resolución de forma general. En este capítulo abordaremos esta cuestión a través del análisis del nuevo sistema.

### 4.1 Mapa conceptual

Llegados a este punto, estamos en condiciones de definir una primera aproximación conceptual de cómo debe ser el sistema a desarrollar:

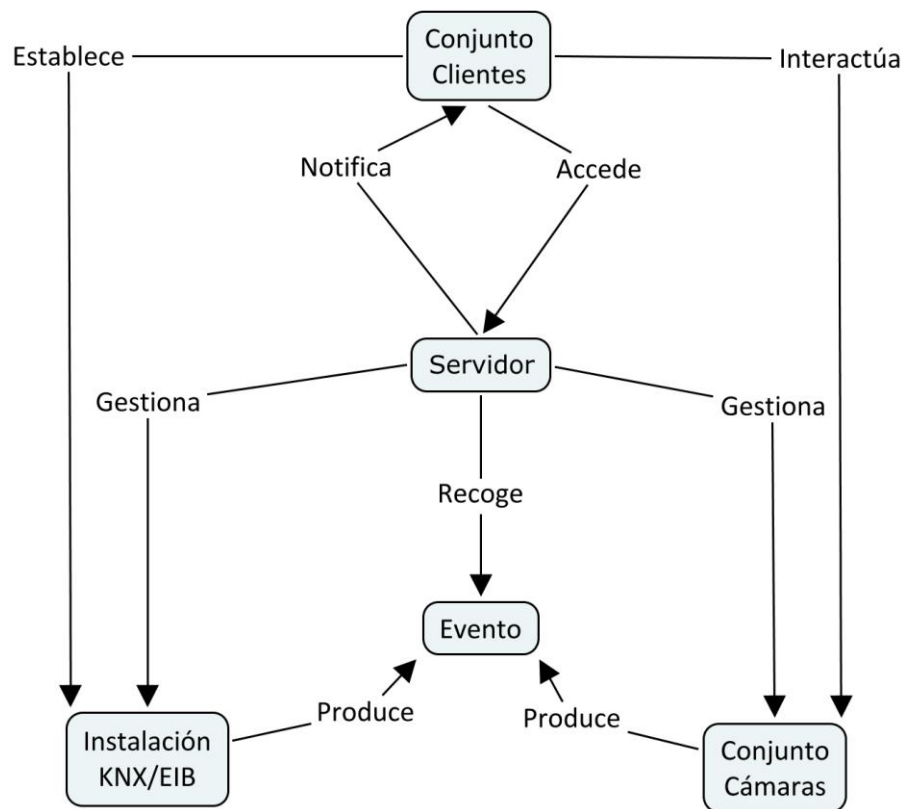
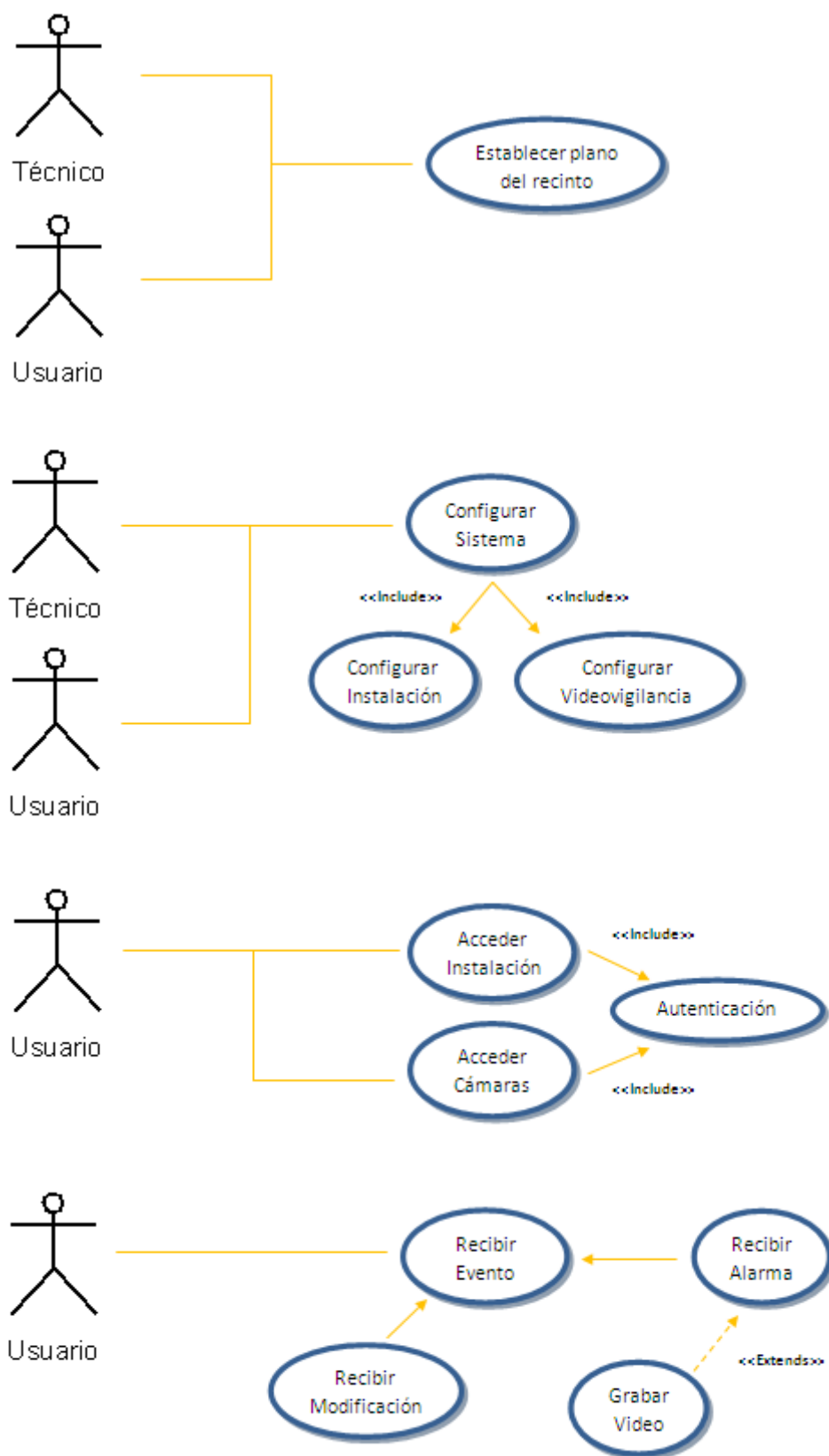


Ilustración 24. Mapa conceptual

## 4.2 Casos de uso



<b>Nombre</b>	<b>Establecer plano del recinto</b>
<b>Descripción</b>	Establece el plano (exportado desde AutoCad) del recinto donde se encuentra la instalación domótica.
<b>Precondiciones</b>	Plano disponible.
<b>Flujo</b>	El actor selecciona el plano que quiere establecer, valida la operación y el sistema almacena la información.
<b>Poscondiciones</b>	Plano almacenado en el sistema.

<b>Nombre</b>	<b>Configurar instalación</b>
<b>Descripción</b>	Especifica la configuración de los dispositivos domóticos de la instalación.
<b>Precondiciones</b>	Plano almacenado en el sistema.
<b>Flujo</b>	El actor modifica la configuración según sus necesidades, valida la operación y el sistema almacena los datos.
<b>Poscondiciones</b>	Configuración almacenada en el sistema.

<b>Nombre</b>	<b>Configurar cámaras</b>
<b>Descripción</b>	Especifica la configuración de las cámaras IP de la instalación.
<b>Precondiciones</b>	-
<b>Flujo</b>	El actor modifica la configuración según sus necesidades, valida la operación y el sistema almacena los datos.
<b>Poscondiciones</b>	Configuración almacenada en el sistema.

<b>Nombre</b>	<b>Autenticación</b>
<b>Descripción</b>	Autentica al usuario en el sistema.
<b>Precondiciones</b>	-
<b>Flujo</b>	El actor introduce los datos de identificación y espera el resultado.
<b>Poscondiciones</b>	Resultado de la operación enviado al usuario.

<b>Nombre</b>	<b>Acceder instalación</b>
<b>Descripción</b>	Permite al actor conectarse al servidor para interactuar con la instalación.
<b>Precondiciones</b>	Actor autenticado.
<b>Flujo</b>	El actor realiza las operaciones deseadas en la instalación.
<b>Poscondiciones</b>	-

<b>Nombre</b>	<b>Acceder cámaras</b>
<b>Descripción</b>	Permite al actor conectarse al servidor para interactuar con las cámaras IP.
<b>Precondiciones</b>	Actor autenticado.
<b>Flujo</b>	El actor realiza las operaciones deseadas en las cámaras.
<b>Poscondiciones</b>	-

<b>Nombre</b>	<b>Recibir alarma</b>
<b>Descripción</b>	Notifica al actor de una alarma activada en el sistema.
<b>Precondiciones</b>	Alarma disparada.
<b>Flujo</b>	El sistema notifica al usuario de una alarma producida dentro de éste, opcionalmente puede activar la grabación de vídeo.
<b>Poscondiciones</b>	Notificación enviada.

<b>Nombre</b>	<b>Recibir modificación</b>
<b>Descripción</b>	Notifica al actor de una modificación en el sistema.
<b>Precondiciones</b>	Modificación realizada
<b>Flujo</b>	El sistema notifica al usuario de una modificación en el estado de la instalación o de las cámaras.
<b>Poscondiciones</b>	Notificación enviada.

### 4.3 Arquitectura del sistema

En esta sección veremos cómo debe estructurarse el sistema que vamos a desarrollar: cuáles son sus componentes y cómo se relacionan entre sí.

#### 4.3.1 Diagrama global

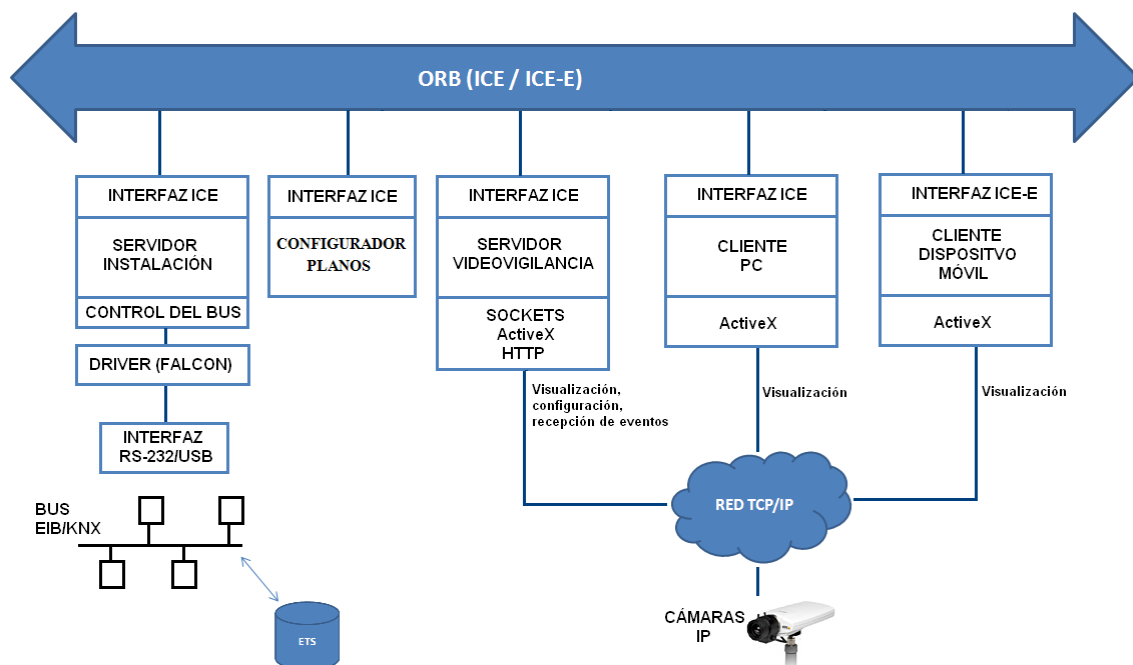


Ilustración 25 Arquitectura QuercusDomoSystem

### 4.3.2 Componentes

- Servidor de la instalación: permite la introducción de los componentes que presenta la instalación domótica junto a las relaciones entre ellos. Además, lleva a cabo la gestión de la instalación domótica, es decir, interactúa con la instalación de forma directa, enviándole órdenes y recibiendo eventos. Se codificará en lenguaje C++ utilizando las librerías MFC de .NET.
- Configurador de planos: da la posibilidad al usuario de configurar los dispositivos domóticos en un plano de su vivienda, además, permite la selección de imágenes para cada uno de los dispositivos. Se codificará en C#.
- Servidor de videovigilancia: permite gestionar la configuración de las cámaras y tratar los eventos que provengan de estas. Se codificará en lenguaje C++ utilizando las librerías MFC de .NET.
- Cliente PC: proporciona al usuario un interfaz que le permita conectarse a los servidores para acceder a la instalación domótica y al sistema de videovigilancia. Se codificará utilizando C#.
- Cliente dispositivos móviles: ofrecerá una funcionalidad similar al cliente de PC adaptada a dispositivos móviles. Puesto que ICE-E no soporta C#, será codificado en C++ utilizando las librerías MFC para dispositivos móviles de .NET.

### 4.3.3 Estructuras de datos

#### 4.3.3.1 *Instalación domótica*

La información de la instalación domótica en el cliente se almacena en una lista de funcionalidades donde cada funcionalidad tiene almacenado los siguientes datos:

- Dirección de grupo de la funcionalidad
- Imagen para el estado activo
- Imagen para el estado inactivo
- Posición en el eje de las X dentro del plano
- Posición en el eje de las Y dentro del plano
- Estado en el que se encuentra

El cliente se descarga la información desde el servidor de la instalación domótica y la almacena en los distintos campos.

El servidor almacena la información de la instalación domótica y contiene un vector con el estado en el que se encuentran las distintas funcionalidades de la instalación domótica.

#### 4.3.3.2 *Videovigilancia*

La principal estructura de datos consiste en una lista de objetos *cámara* (cada uno de los cuales contendrá toda la información relevante acerca de una cámara) gestionada por el servidor de videovigilancia.

Esta lista de cámaras se mantendrá en la memoria del servidor de videovigilancia y será almacenada en disco cuando éste deje de funcionar o cuando se produzca un cambio en la misma.

Puesto que vamos a encontrarlos en un entorno distribuido multiusuario será vital mantenerla actualizada sin incurrir en errores e informar a los usuarios cuando se produzca un cambio desde cualquier punto.

#### 4.3.4 *Relaciones*

##### 4.3.4.1 *Instalación domótica*

- **Cliente-Servidor, Servidor-Cliente:** esta comunicación será realizada mediante ICE, ambas partes realizan funciones de cliente y de servidor, ya que, el cliente accederá al servidor para realizar funciones en la instalación, y el servidor accederá a los clientes para notificar cambios producidos en la instalación.
- **Servidor-Instalación domótica:** gracias a las librerías de acceso al bus (Falcon) se puede llevar a cabo la comunicación de la aplicación con los dispositivos instalados en la vivienda. La comunicación se puede realizar a través del puerto serie RS-232 y a través de los puertos USB. En la capa más cercana a la instalación domótica corre un bucle que recibe las órdenes a través de sockets TCP.
- **Instalación domótica-Servidor:** el servidor llevará incorporado la parte de obtención de eventos de la instalación, con lo que la comunicación entre ambos se realiza de forma directa, ya que pertenecen a la misma aplicación.

#### 4.3.4.2 *Videovigilancia*

- **Cámaras-Clientes:** el acceso a las cámaras desde los clientes, para su visualización, se realizará a través del control ActiveX proporcionado por AXIS.
- **Cámaras-Servidor de videovigilancia:** en este apartado se diferencian tres aspectos.
  1. Visualización: al igual que con los clientes, se hará mediante el control ActiveX.
  2. Configuración: algunas opciones de configuración de cámaras pueden realizarse mediante el control ActiveX, sin embargo, otras requerirán de la API de acceso HTTP que proporcionan las cámaras de AXIS (ver documentación de la API para más detalles).
  3. Eventos: las cámaras generan eventos como la detección de movimiento. Estos eventos se envían como segmentos TCP/IP a un socket (en nuestro caso será un puerto asignado en el servidor). Una vez enviados será el servidor el que se encargue de gestionar la respuesta a dichos eventos.
- **Servidor de videovigilancia-Clientes:** en este punto se sitúa el middleware ICE y su versión para dispositivos móviles: ICE-E. A través de éstos se permitirá el acceso a la configuración de las cámaras y su modificación, además de informar a los clientes de los eventos ocurridos en las cámaras desde el servidor.

#### 4.3.5 *Datos persistentes*

##### 4.3.5.1 *Instalación domótica*

El servidor de la instalación domótica mantiene cuatro tipos de ficheros de forma persistente, son los presentados a continuación.

##### 4.3.5.1.1 *Datos instalación domótica*

Este fichero almacena la configuración de la instalación domótica, sus dispositivos y funcionalidades. A continuación se muestra un breve ejemplo:

DISPOSITIVOS

Luz :: 1/1/2

Pulsador :: 1/1/1

DIRECCIONES

Pulsador :: 1/1/1 || Luz :: 1/1/2 || 0/0/1

En la primera parte del fichero se almacenan los dispositivos junto a su dirección física, y posteriormente se almacenan las direcciones de grupo o funcionalidades del sistema que se han configurado a partir de los dispositivos. Las direcciones de grupo almacenan los dispositivos que la componen juntos a su dirección física y la dirección de grupo de dicha funcionalidad.

En el ejemplo, nuestra instalación está compuesta por una luz y un pulsador, con sus respectivas direcciones físicas, y una única funcionalidad: encender/apagar la luz a partir del pulsador.

#### 4.3.5.1.2 Estado de la instalación domótica

Este fichero almacena el vector comentado en el apartado de estructuras de datos que contiene el estado en el que se encuentra la instalación domótica.

#### 4.3.5.1.3 Gestión de usuarios

Los usuarios que tienen acceso al servidor son registrados en el mismo y en un fichero se almacena información relativa a ellos siguiendo este formato:

- Nombre de usuario
- Contraseña
- Path de localización del fichero de información de la instalación de dicho usuario

#### 4.3.5.1.4 Información instalación de cada usuario

Cada usuario que accede a la instalación domótica tiene la posibilidad de configurar los dispositivos a su gusto, por lo que, para cada usuario registrado en el sistema, se almacena la siguiente información:

- Número de funcionalidades del sistema.
- Funcionalidades:
  - Dirección de grupo
  - Imagen para el estado activo
  - Imagen para el estado inactivo
  - Posición en el eje X en el plano
  - Posición en el eje Y en el plano



#### 4.3.5.2 *Videovigilancia*

Se mantendrá un fichero log en el servidor que registre la actividad de éste. El formato de los datos guardados será:

*dd/MM/AAAA – hh:mm:ss Mensaje registrado*

Por ejemplo:

23/05/2007 - 17:17:17 Servidor inicializado

Además, al producirse una detección de movimiento en una cámara preparada para ello se activará la grabación de vídeo y éste quedará almacenado en el servidor.

### 4.4 Diagrama de paquetes

Exponemos, a continuación, el diagrama de paquetes del sistema donde se presentan los proyectos que lo componen y las relaciones de acceso entre ellos.

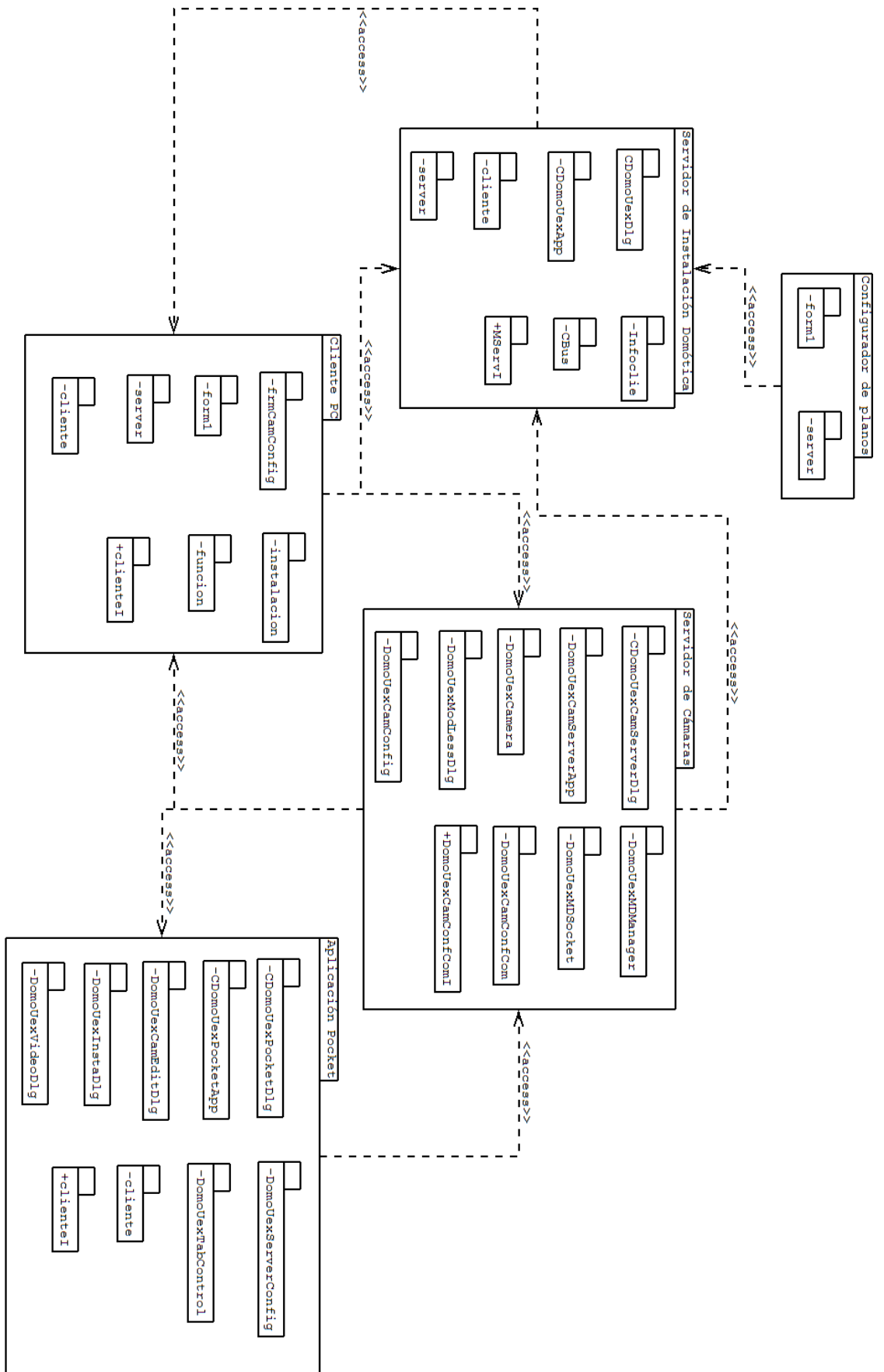


Ilustración 26. Diagrama de paquetes

## 4.5 Diagrama de clases

En esta sección mostramos los diagramas de clases de cada una de las partes de la aplicación así como una descripción de cada y las relaciones entre ellas. Al final del documento se incluye un anexo con el diagrama de clases completo (ANEXO 2: DIAGRAMA DE CLASES AMPLIADO).

### 4.5.1 Servidor instalación

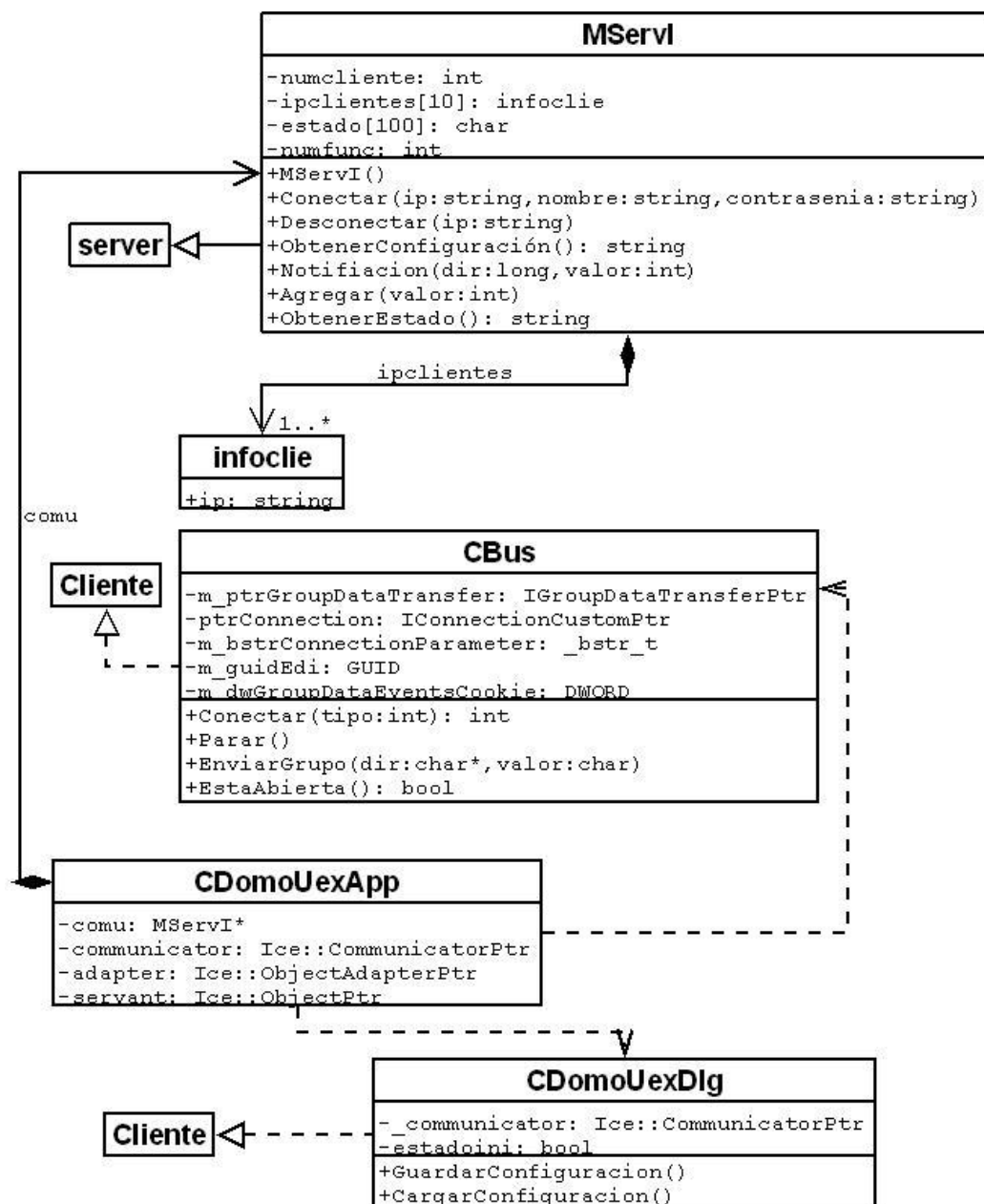


Ilustración 27. Diagrama de clases servidor instalación

4.5.1.1 *MServi*

<b>MServi</b>
<pre> - numcliente: int - ipclientes[10]: infoclie - estado[100]: char - numfunc: int + MServi() + Conectar(ip: string, nombre: string, contrasenia: string) + Desconectar(ip: string) + ObtenerConfiguración(): string + Notifiacion(dir: long, valor: int) + Agregar(valor: int) + ObtenerEstado(): string </pre>

Ilustración 28. *MServi*

Esta clase implementa los métodos que pueden ser invocados por los clientes en el servidor. Los atributos más importantes son los siguientes:

- numclientes: Indica el número de clientes conectados al servidor.
- ipclientes: Almacena las direcciones IP de los clientes conectados al servidor.
- estado: Almacena el estado en el que se encuentran las funcionalidades de la instalación domótica.
- numfunc: Indica el número de funcionalidades que posee la instalación.

Los métodos se corresponden con los presentes en los ficheros slice, que se detallarán más adelante.

4.5.1.2 *CBus*

<b>CBus</b>
<pre> -m_ptrGroupDataTransfer: IGroupDataTransferPtr -ptrConnection: IConnectionCustomPtr -m_bstrConnectionParameter: _bstr_t -m_guidEdi: GUID -m_dwGroupDataEventsCookie: DWORD + Conectar(tipo: int): int + Parar() + EnviarGrupo(dir: char*, valor: char) + EstaAbierta(): bool </pre>

Ilustración 29. *CBus*

Esta clase se encarga de interactuar con el bus EIB. Los atributos que posee son utilizados para llevar a cabo la comunicación con el mismo. Presenta los siguientes métodos:

- Conectar: Conecta la aplicación al bus EIB, el parámetro *tipo* indica el tipo de dispositivo a utilizar para la conexión, USB o RS-232.
- Parar: Desconecta la aplicación del bus EIB.
- EnviarGrupo: Lleva a cabo una operación en el bus EIB, los parámetros indica la dirección (*dir*) sobre la que se realizará la operación, y su valor.
- EstaAbierta: Comprueba si está conectada la aplicación con el servidor.

#### 4.5.1.3 *CDomoUexApp*

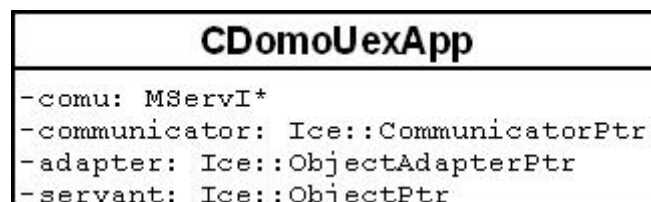


Ilustración 30. *CDomoUexApp*

Clase principal de la aplicación, posee los atributos necesarios para ofrecer servicios a los clientes.

#### 4.5.1.4 *CDomoUexDlg*

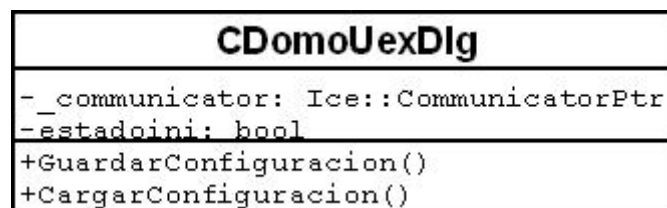


Ilustración 31. *CDomoUexDlg*

Clase que proporciona un interfaz gráfico para la ejecución de la aplicación. Los atributos se utilizan para notificar eventos en los clientes. Hay que destacar los siguientes métodos:

- GuardarConfiguracion: Almacena el estado en el que se encuentra el servidor en un fichero de texto, el estado incluye, configuración de la instalación domótica y el estado de las funcionalidades.
- CargarConfiguracion: Carga el fichero de configuración en el servidor.

## 4.5.2 Servidor videovigilancia

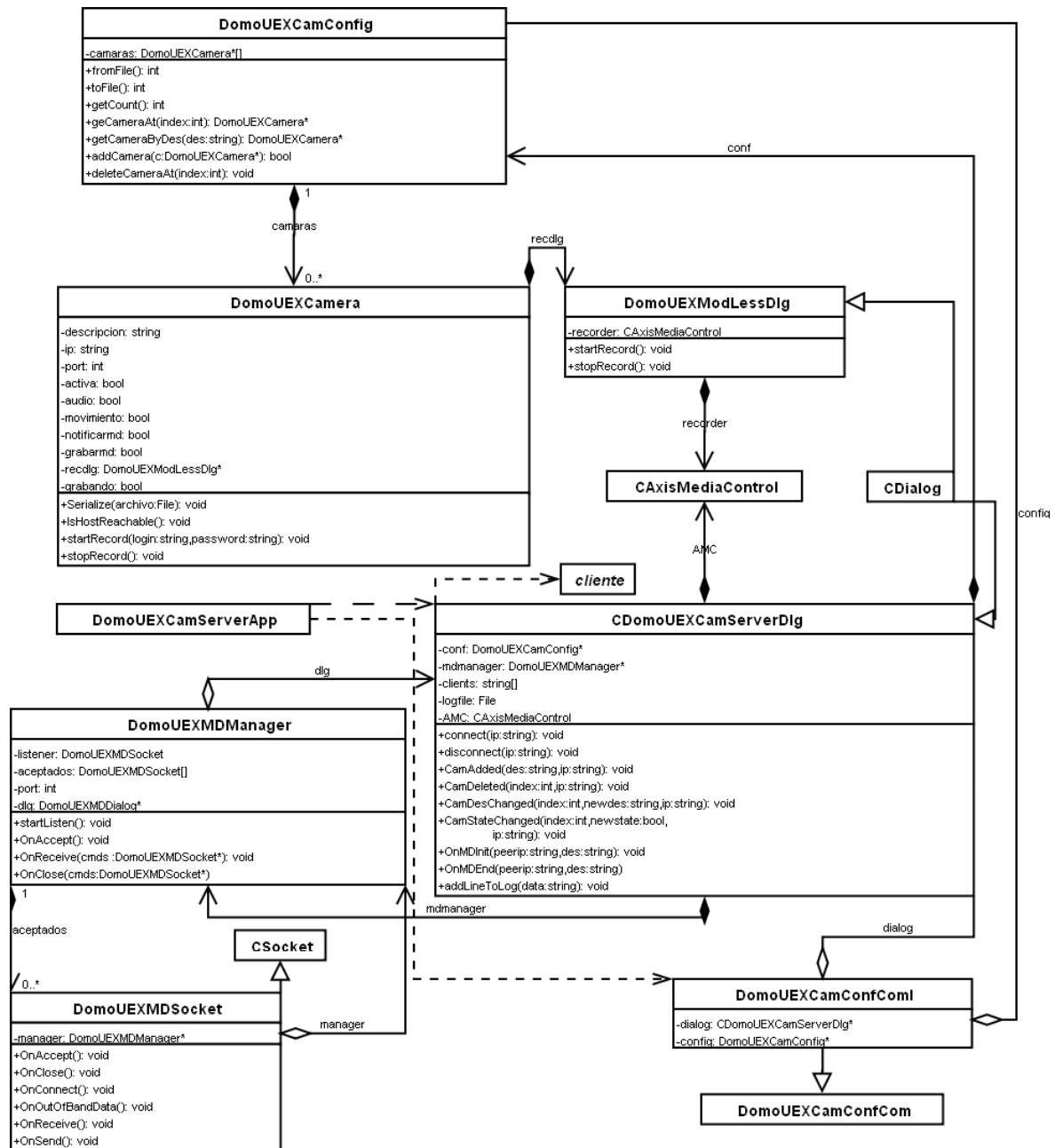


Ilustración 32. Diagrama de clases videovigilancia

4.5.2.1 *DomoUEXModLessDlg*

DomoUEXModLessDlg
-recorder: CAxisMediaControl
+startRecord(): void
+stopRecord(): void

Ilustración 33. DomoUEXModLessDlg

Hereda de CDialog (la clase de diálogo básica de MFC) y es la ventana encargada de grabar vídeo para una determinada cámara. Para ello cuenta con un atributo *recorder*, instancia de la clase control ActiveX CAxisMediaControl. Los métodos startRecord y stopRecord indican cuando debe comenzarse a grabar vídeo y cuando debe terminar la grabación.

4.5.2.2 *DomoUEXCamera*

DomoUEXCamera
-descripcion: string
-ip: string
-port: int
-activa: bool
-audio: bool
-movimiento: bool
-notificarmd: bool
-grabarmd: bool
-recdlg: DomoUEXModLessDlg*
-grabando: bool
+Serialize(archivo:File): void
+IsHostReachable(): void
+startRecord(login:string,password:string): void
+stopRecord(): void

Ilustración 34. DomoUEXCamera

Esta clase representa una cámara dentro de nuestro sistema. El objeto incluye, como atributos, información relevante acerca de ésta:

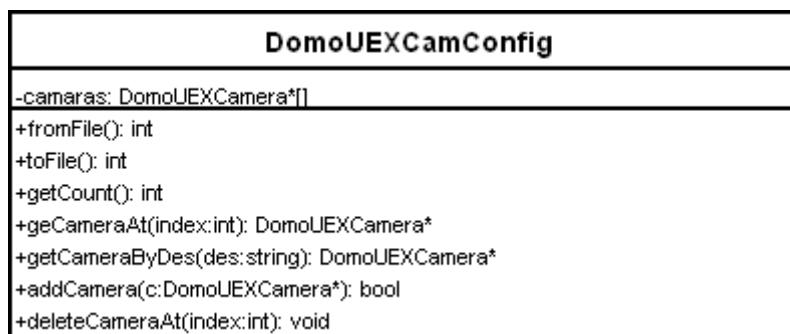
- descripción: Nombre asignado a la cámara para diferenciarla del resto (por ejemplo: cocina, salón, cámara1...)
- ip: dirección IP de la cámara.
- port: puerto TCP donde la cámara acepta conexiones (normalmente 80).
- activa: indica si la cámara se encuentra activa (true) o apagada (false).
- audio: indica si la cámara puede transmitir audio (true) o no (false).

- movimiento: indica si la cámara puede detectar movimientos (true) o no (false).
- notificarmd: indica si se ha de notificar a los clientes la detección de movimiento (true) o no (false).
- grabarmd: indica si se debe grabar movimiento (true) o no (false).
- recdlg: referencia a la ventana de grabación de vídeo (clase DomoUEXModLessDlg) asociada a esta cámara.
- grabando: indica si la cámara está grabando vídeo actualmente (true) o no (false).

Los métodos más importantes son:

- Serialize: guarda la información de la cámara en el fichero pasado por parámetro.
- IsHostReachable: actualiza el estado de la cámara (atributo activa) averiguando si ésta se encuentra disponible.
- startRecord: comienza a grabar vídeo desde la ventana asociada a la cámara con el nombre de usuario y password pasados como parámetros.
- stopRecord: finaliza la grabación de vídeo.

#### 4.5.2.3 *DomoUEXCamConfig*



**Ilustración 35.** *DomoUEXCamConfig*

Esta clase es la encargada de organizar las cámaras del sistema en una estructura de lista contenida en el atributo *camaras*. Los métodos disponibles son:

- fromFile: carga la colección de cámaras desde un fichero.
- toFile: almacena la colección de cámaras en un fichero.
- getCount: devuelve el número de cámaras del sistema.
- getCameraAt: devuelve una referencia a la cámara situada en la posición pasada por parámetro.
- getCameraByDes: devuelve una referencia a la cámara cuya descripción coincida con la que se pasa por parámetro.
- addCamera: añade la cámara pasada por parámetro a la lista de cámaras del sistema.
- deleteCameraAt: elimina la cámara situada en la posición indicada por parámetro.



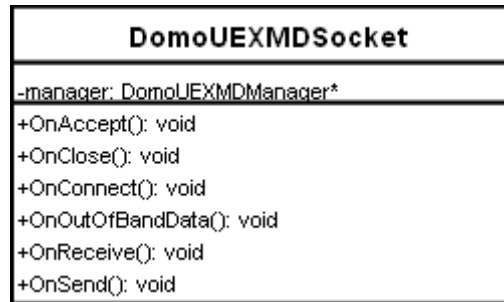
4.5.2.4 *DomoUEXMDSocket*

Ilustración 36. DomoUEXMDSocket

Hereda de la clase CSocket (abstracta) y redefine los métodos relacionados con las conexiones TCP/IP establecidas con las cámaras. Cuenta con un atributo *manager* que apuntará al gestor de sockets del sistema.

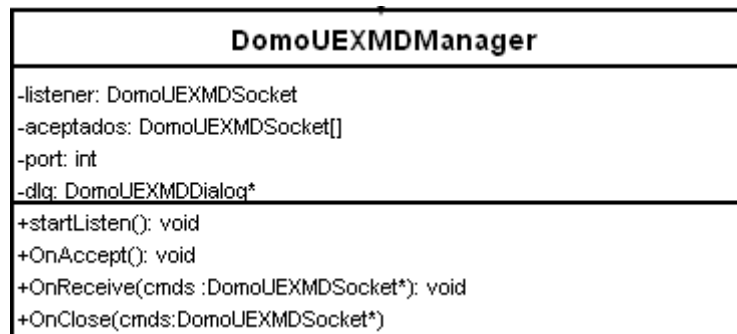
4.5.2.5 *DomoUEXMDManager*

Ilustración 37. DomoUEXMDManager

Gestiona los sockets del sistema e informa a la ventana principal de los eventos recibidos. Sus atributos son:

- listener: socket que se encargará de aceptar las conexiones entrantes e informar al gestor.
- aceptados: vector dinámico de sockets actualmente abiertos con conexión activa.
- port: número de puesto TCP donde se aceptan las conexiones.
- dlg: referencia a la ventana principal.

Los métodos principales son:

- startListen: indica que debe comenzarse a realizar escuchas en el puerto seleccionado.

- OnAccept: conexión aceptada desde listener.
- OnReceive: datos recibidos en el socket pasado por parámetro.
- OnClose: conexión cerrada en el socket pasado por parámetro.

#### 4.5.2.6 *DomoUEXCamConfComI*



Ilustración 38. *DomoUEXCamConfComI*

Implementa los métodos ofrecidos por el interfaz de comunicación ICE para la gestión de cámaras. Este interfaz será comentado con mayor detalle en capítulos posteriores, basta comentar ahora que cuenta con dos atributos de referencia a la ventana principal (*dialog*) y al gestor de cámaras (*config*).

#### 4.5.2.7 *CDomoUEXCamServerDlg*

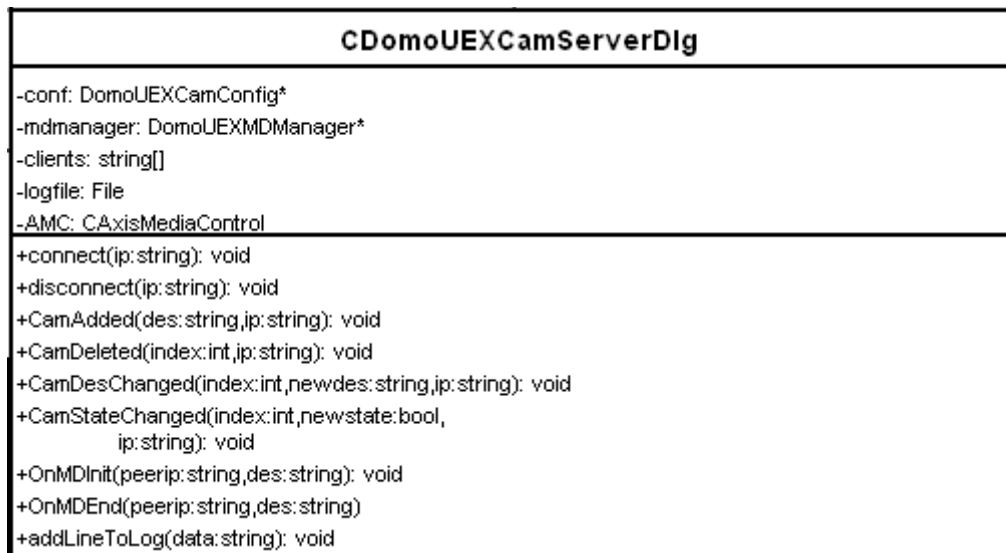


Ilustración 39. *CDomoUEXCamServerDlg*

Ventana de diálogo principal de la aplicación. Para actualizar su información cuenta con atributos referencia al gestor de cámaras y al gestor de sockets (*conf* y *mdmanager* respectivamente). También contiene una lista con las direcciones IP de los clientes conectados

al sistema (*clients*), un manejador de log *logfile* para las excepciones producidas en las comunicaciones ICE y un control ActiveX *AMC* para interactuar con las cámaras configuradas.

Los principales métodos son:

- connect: registra una nueva IP de usuario en la lista de usuarios conectados.
- disconnect: elimina una IP de cliente registrado.
- addLineToLog: inserta la línea pasada por parámetro en el log del servidor.

Los siguientes métodos se activan desde el interfaz de comunicación DomoUEXCamConfComI al recibir peticiones de los clientes:

- CamAdded: nueva cámara insertada desde el cliente *ip* con descripción *des*.
- CamDeleted: cámara posicionada en *index* eliminada desde el cliente *ip*.
- CamDesChanged: descripción de cámara modificada desde un cliente.
- CamStateChanged: estado de una cámara modificada desde un cliente.
- OnMDInit: movimiento detectado en la cámara *des* desde la IP *peerip*.
- OnMDEnd: movimiento finalizado en la cámara *des* desde la IP *peerip*.

4.5.3 Cliente

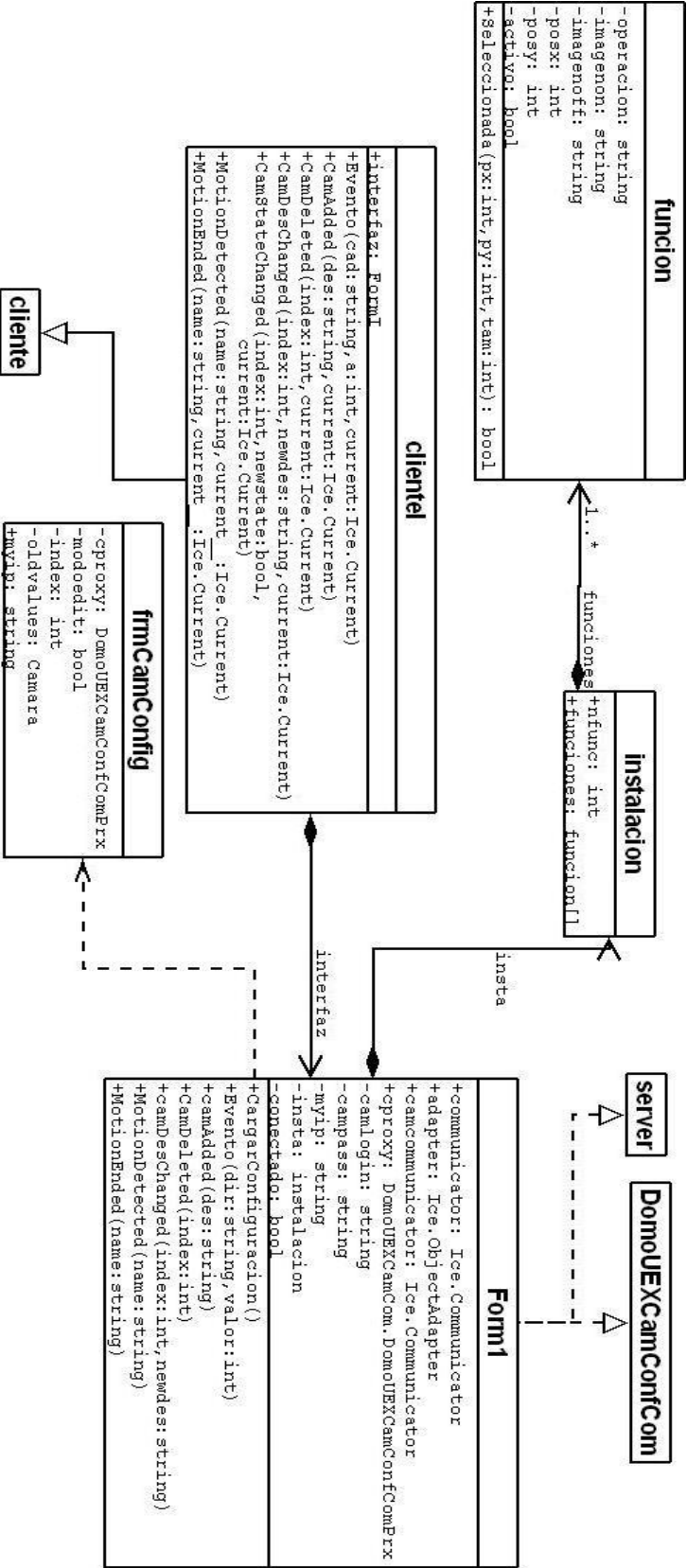


Ilustración 40. Diagrama de clases cliente

4.5.3.1 *funcion*

funcion
<pre>-operacion: string -imagenon: string -imagenoff: string -posx: int -posy: int -activo: bool +Seleccionada(px:int,py:int,tam:int): bool</pre>

Ilustración 41. funcion

Esta clase almacena toda la información acerca de una de las funcionalidades de la instalación domótica. Hay que destacar los siguientes atributos:

- Operación: Almacena la dirección de grupo de la funcionalidad.
- imagenon: Almacena la ruta donde se encuentra la imagen para cuando la funcionalidad se encuentra activa.
- imagenoff: Almacena la ruta donde se encuentra la imagen para cuando la funcionalidad se encuentra inactiva.
- posx: Coordenada en el eje x, dentro del plano, de la funcionalidad.
- posy: Coordenada en el eje y, dentro del plano, de la funcionalidad.
- activo: Indica el estado en el que se encuentra la funcionalidad.

De los métodos cabe destacar el siguiente:

- Seleccionada: Indica si, al haber hecho clic el cliente en el plano de la instalación domótica, la funcionalidad seleccionada es esta.

4.5.3.2 *instalacion*

instalacion
<pre>+nfunc: int +funciones: funcion[]</pre>

Ilustración 42. instalacion

La clase instalación almacena toda la información acerca de la instalación domótica sobre la que se está ejecutando el cliente. Sus atributos más importantes son los siguientes:

- nfunc: Indica el número de funcionalidades que posee la instalación domótica.
- funciones: Almacena la información acerca de todas las funcionalidades que posee la instalación domótica.

4.5.3.3 *clienteI*

<b>clienteI</b>
<pre> +interfaz: FormI +Evento(cad: string, a: int, current: Ice.Current) +CamAdded(des: string, current: Ice.Current) +CamDeleted(index: int, current: Ice.Current) +CamDesChanged(index: int, newdes: string, current: Ice.Current) +CamStateChanged(index: int, newstate: bool,                   current: Ice.Current) +MotionDetected(name: string, current__ : Ice.Current) +MotionEnded(name: string, current__ : Ice.Current) </pre>

Ilustración 43. clienteI

Esta clase implementa los métodos ofrecidos por el cliente al conjunto servidores. Como atributo posee el interfaz gráfico, para que, cuando ocurren eventos, tanto en el servidor de vídeo, como en el servidor de la instalación domótica, se actualice en tiempo real el interfaz gráfico.

Los métodos se comentarán con más detalle en capítulos posteriores.

4.5.3.4 *frmCamConfig*

<b>frmCamConfig</b>
<pre> -cproxy: DomoUEXCamConfComPrx -modeedit: bool -index: int -oldvalues: Camara +myip: string </pre>

Ilustración 44. frmCamConfig

Este diálogo será el que permita al cliente editar la información sobre cada una de las cámaras o crear una nueva. Para ello cuenta con una referencia al proxy de gestión de cámaras (*cproxy*), un valor booleano que indica si el cuadro se ha abierto en modo edición o creación (*modeedit*), un valor entero *index* que identifica la cámara que se está editando y una estructura de tipo *Camara* (veremos más adelante en qué consiste al hablar de los interfaces) que almacena la antigua configuración de la cámara en caso de que se haya abierto el diálogo en modo edición.

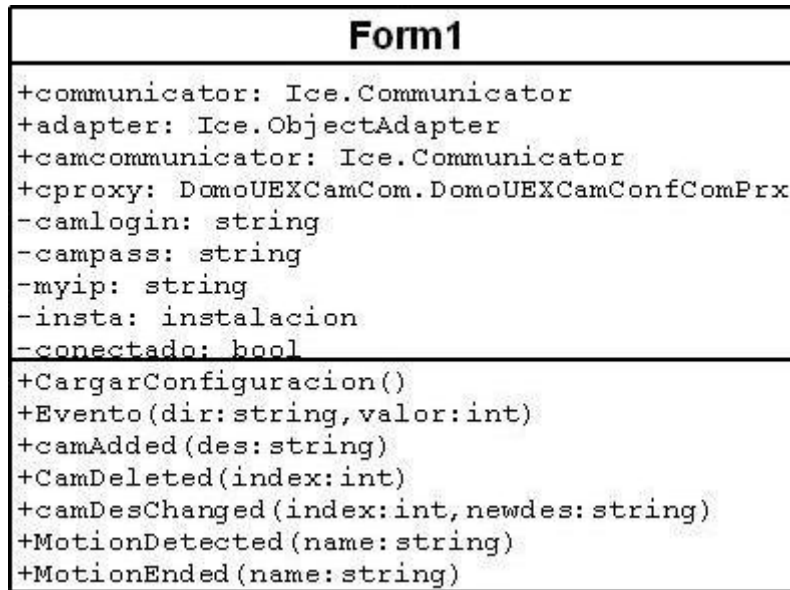
4.5.3.5 *Form1*

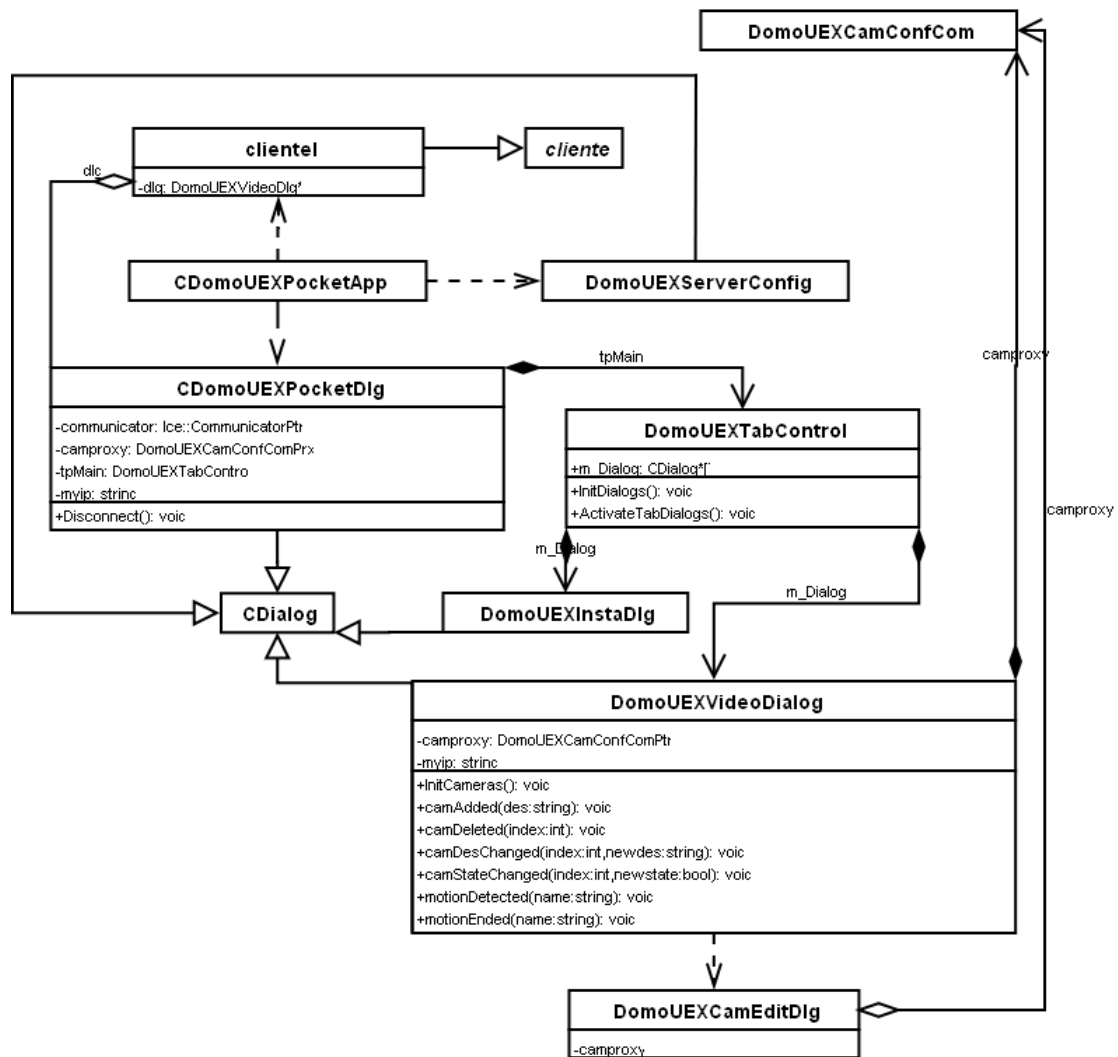
Ilustración 45. Form1

Clase que contiene el interfaz gráfico de la aplicación servidora. Mediante los atributos de carácter público se lleva a cabo la comunicación con el servidor. Los atributos con ámbito privado son utilizados para:

- camlogin: Almacena el nombre de usuario para el módulo de videovigilancia.
- campass: Almacena el password para el módulo de videovigilancia.
- myip: Almacena la dirección IP del cliente donde se ejecuta la aplicación.
- insta: Almacena la información acerca de la instalación domótica.
- conectado: Indica si se encuentra conectado al servidor el cliente.

Los métodos llevan a cabo las operaciones existentes en los ficheros slice sobre el interfaz gráfico.

#### 4.5.4 Cliente dispositivos móviles



#### Ilustración 46. Diagrama de clases cliente dispositivos móviles

#### 4.5.4.1 *DomoUEXCamEditDlg*

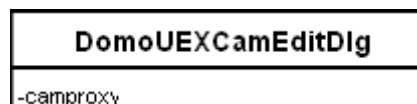


Ilustración 47. DomoUEXCamEditDlg

En esta ventana se editarán las propiedades de las cámaras que se deseen modificar o crear mediante controles visuales. Cuenta con un atributo *camproxy* que apunta al proxy de acceso al servant de configuración de cámaras.



#### 4.5.4.2 *DomoUEXVideoDlg*

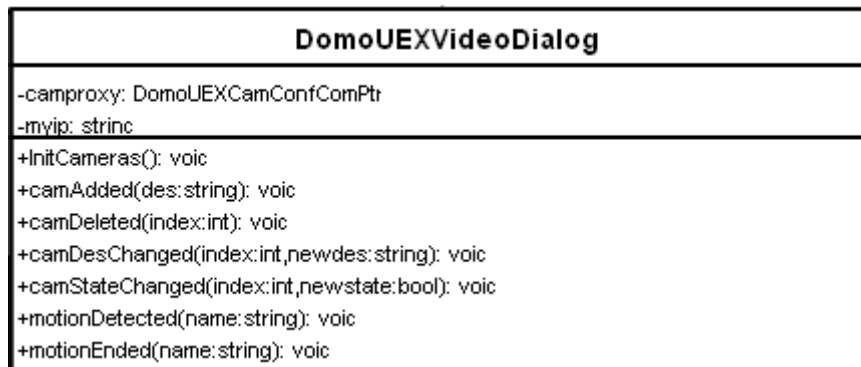


Ilustración 48. DomoUEXVideoDialog

Esta ventana proporciona acceso a la lista de cámaras además recibe los cambios realizados en éstas a través de ICE. Como atributos cuenta con una referencia al servant de configuración de cámaras (*camproxy*) y la IP del host local (*ip*).

#### 4.5.4.3 *DomoUEXTabControl*

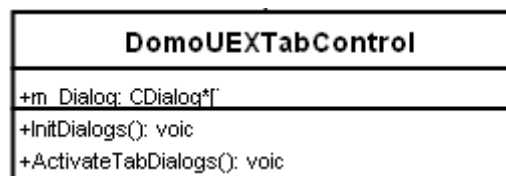


Ilustración 49. DomoUEXTabControl

Puesto que en la versión para dispositivos móviles no se ofrecen controles con pestañas funcionales debemos implementarlo nosotros. Para ello este control cuenta con un vector de cuadros de diálogo (cada una de las pestañas) y métodos para inicializar y activar estos diálogos.

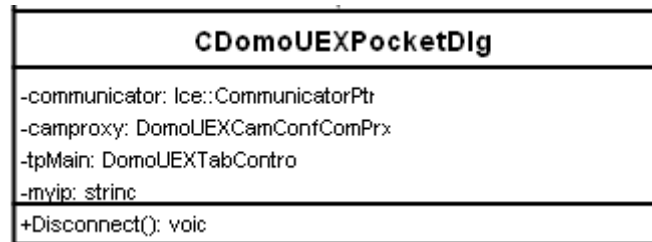
#### 4.5.4.4 *clientel*



Ilustración 50. clientel

Interfaz de comunicación ICE que ofrece el cliente. Cuenta con un atributo *dlg* de referencia al diálogo de control de vídeo para actualizar el interfaz al producirse llamadas a alguno de sus métodos.

#### 4.5.4.5 *CDomoUEXPocketDlg*



**Ilustración 51.** *CDomoUEXPocketDlg*

Ventana principal de la aplicación, cuenta con los siguientes atributos:

- communicator: referencia a la estructura de comunicaciones de ICE.
- camproxy: referencia al proxy de acceso al servant de configuración de cámaras.
- tpMain: panel de pestañas principal.
- myip: IP del host local.

El método Disconnect tiene como objetivo informar al servidor de vídeo cuando el cliente sale del sistema.

#### 4.5.5 DUConfig

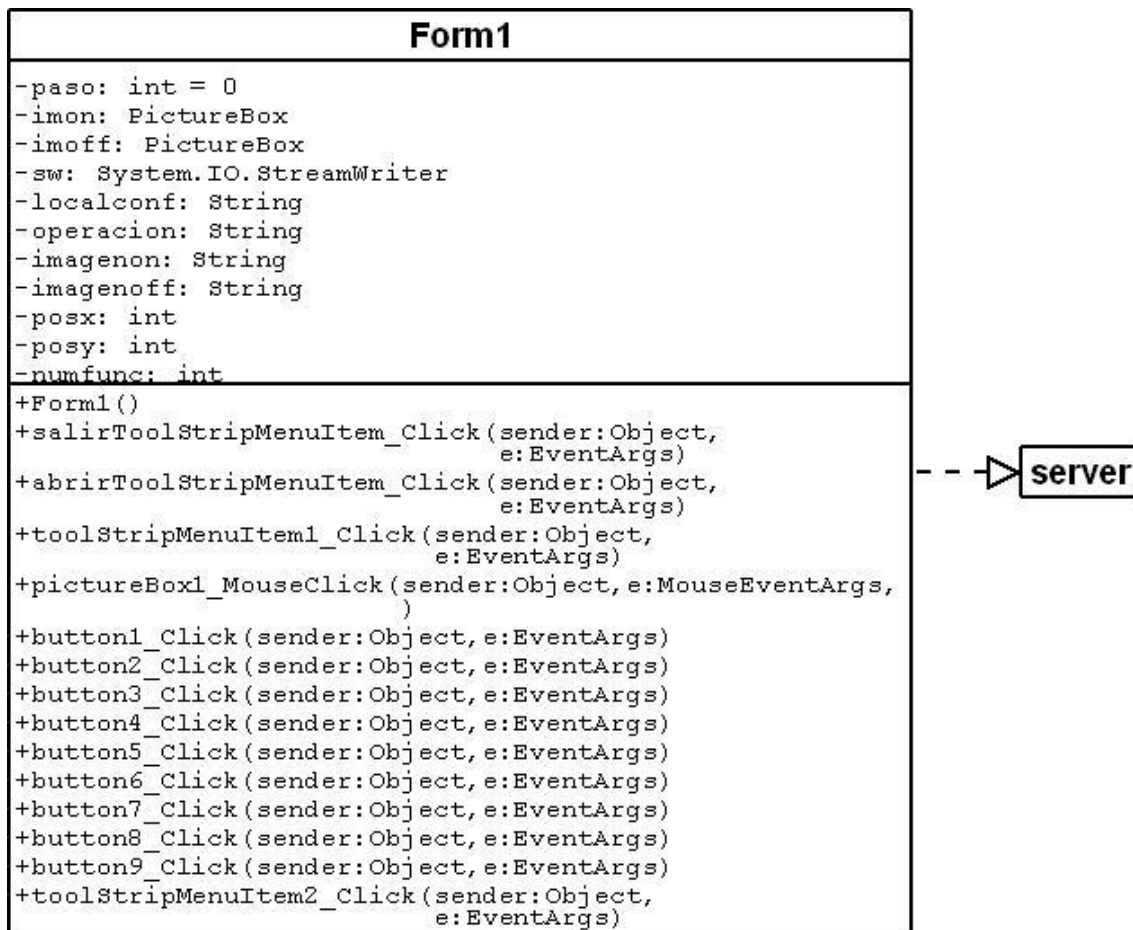


Ilustración 52. DUConfig

Mediante esta aplicación, se lleva a cabo la configuración de las funcionalidades existentes en la instalación domótica sobre el plano de la vivienda. Hay que destacar los siguientes atributos:

- **paso:** Indica en el punto de configuración en el que se encuentra la aplicación.
- **imon:** Indica la imagen seleccionada para el estado activo de la funcionalidad que se está configurando.
- **imoff:** Indica la imagen seleccionada para el estado inactivo de la funcionalidad que se está configurando.
- **sw:** Flujo de escritura a fichero.
- **localconf:** Ruta donde se almacenará la configuración que se está realizando.
- **operacion:** Dirección de grupo de la funcionalidad que se está configurando.
- **imagenon:** Ruta donde se encuentra la imagen seleccionada para el estado activo de la funcionalidad.
- **imagenoff:** Ruta donde se encuentra la imagen seleccionada para el estado inactivo de la funcionalidad.
- **posx:** Posición de la funcionalidad en el plano, eje x.
- **posy:** Posición de la funcionalidad en el plano, eje y.
- **numfunc:** Lleva el número de funcionalidades que se han configurado.

## 4.6 Diagramas de secuencia

A continuación se muestran cuatro diagramas de secuencia que ilustran las cuatro interacciones más importantes del usuario con el sistema: acceso a la instalación domótica, visualización del sistema de videovigilancia, recepción de un evento desde la instalación y recepción de una alarma desde una cámara de red.

### 4.6.1 Acceder Instalación

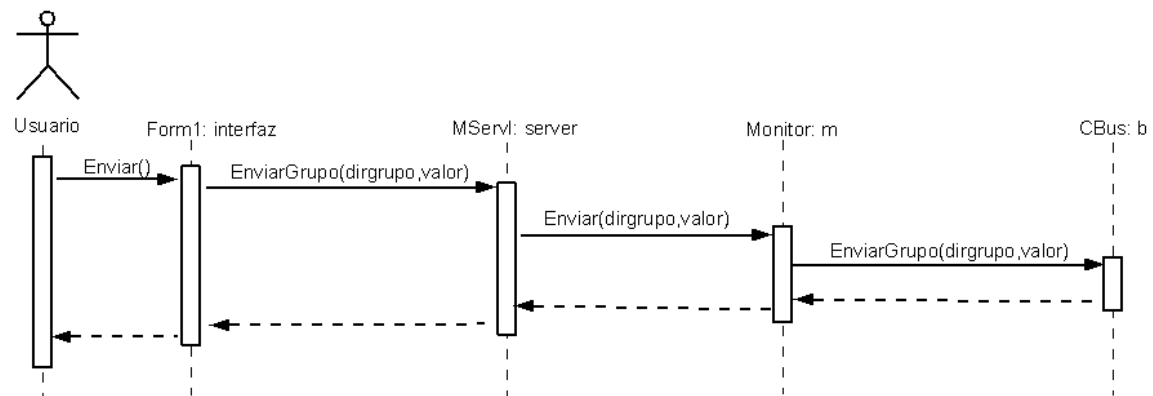


Ilustración 53. Diagrama de secuencia Acceder Instalación

### 4.6.2 Visualizar cámaras

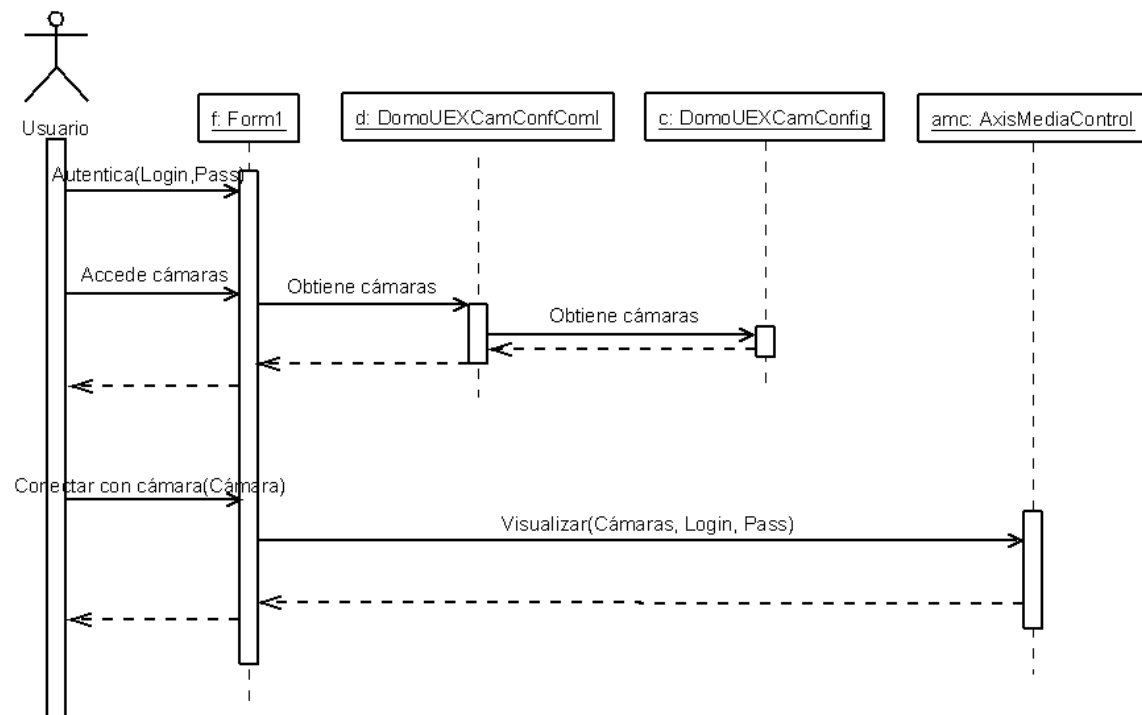


Ilustración 54. Diagrama de secuencia Acceder Cámaras

### 4.6.3 Recibir evento instalación

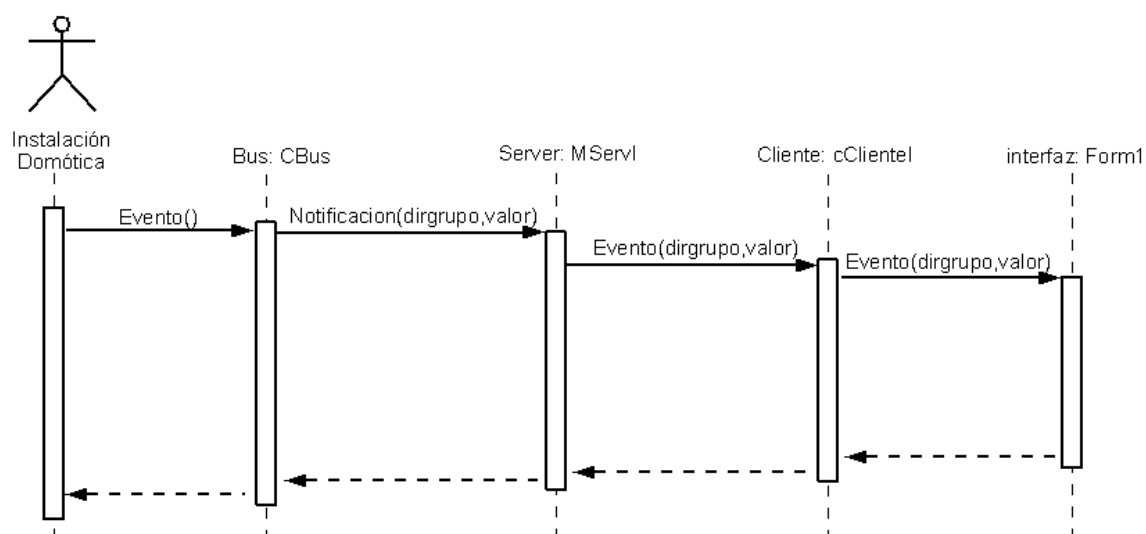


Ilustración 55 Diagrama de secuencia Recibir evento instalación

4.6.4 Recibir alarma cámara

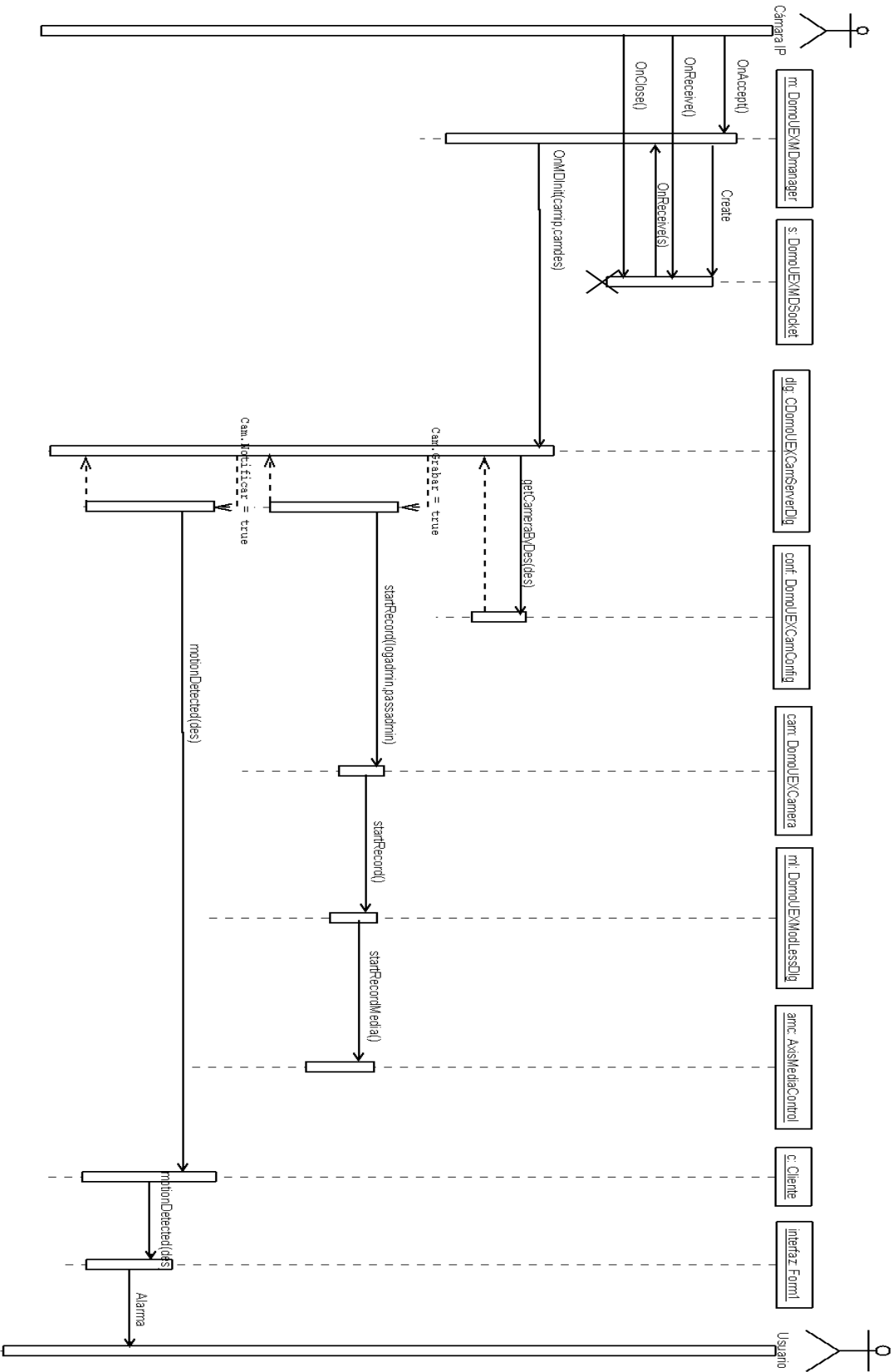


Ilustración 56. Caso de Uso Recibir alarma cámara

## 5 Manual de programador

### 5.1 Interfaces ICE

En esta sección se muestra la definición SLICE (similar al IDL de CORBA como se vio anteriormente) de los interfaces que deben hacer de enlace entre los diferentes componentes del sistema y una descripción de los métodos que implementan.

#### 5.1.1 Servidor de instalación

Interfaz del servant que ofrece el servidor de la instalación, se corresponde con la clase abstracta *server* vista en los diagramas de clases y es implementado por la clase *MIServ*, que hereda de la anterior. Esta implementación es única y se incluye en el servidor de la instalación.

```
module Servidor
{
    interface CServ
    {
        string Conectar(string IP, string usuario, string contrasenia);

        void Desconectar(string IP);

        void EnviarGrupo(string IP, int valor);

        string ObtenerConfiguracion();

        void Notificacion(long dir, int valor);

        void Agregar(int valor);

        string ObtenerEstado();
    };
};
```

**string Conectar(string IP, string usuario, string contrasenia);**

- **Descripción:** Registra un cliente en el servidor
- **IP:** Dirección IP del cliente
- **Usuario:** Nombre de usuario del cliente

- **Contraseña:** Contraseña del usuario
- **Retorno:** Cadena que contiene el fichero con la descripción para el cliente de sus parámetros de configuración

**void Desconectar(string IP);**

- **Descripción:** Desconecta al cliente del servidor
- **IP:** Dirección IP del cliente

**void EnviarGrupo(string IP, int valor);**

- **Descripción:** Realiza una acción en la instalación domótica
- **IP:** Dirección de grupo de la funcionalidad sobre la cual se llevará a cabo la acción
- **Valor:** Valor de la operación a llevar a cabo

**string ObtenerConfiguracion();**

- **Descripción:** Obtiene la configuración de la instalación domótica
- **Retorno:** Cadena que contiene la configuración de la instalación domótica

**void Notificacion(long dir, int valor);**

- **Descripción:** Recibe una notificación de la instalación domótica
- **Dir:** Dirección sobre la cual ha ocurrido un evento
- **Valor:** Valor del evento ocurrido

**void Agregar(int valor);**

- **Descripción:** Introduce una funcionalidad en el estado de la instalación
- **Valor:** Valor que posee esa funcionalidad

**string ObtenerEstado();**

- **Descripción:** Obtiene el estado de la instalación
- **Retorno:** Cadena que contiene el estado en el que se encuentra la instalación

### 5.1.2 Servidor de videovigilancia

Interfaz del servant que ofrece el servidor de videovigilancia, se corresponde con la clase abstracta *DomoUEXCamConfCom* vista en los diagramas de clases y es implementado por la clase *DomoUEXCamConfComI*, que hereda de la anterior. Esta implementación es única y se incluye en el servidor de videovigilancia.



```
module DomoUEXCamCom
```

```
{
```

```
    struct Camara
```

```
    {
```

```
        string des;
```

```
        string ip;
```

```
        int port;
```

```
        bool ptz;
```

```
        bool activa;
```

```
        bool audio;
```

```
        bool movimiento;
```

```
        bool notificarmd;
```

```
        bool grabarmd;
```

```
    };
```

```
    exception ErrorRango
```

```
    {
```

```
        int index;
```

```
        int maxindex;
```

```
    };
```

```
    interface DomoUEXCamConfCom
```

```
    {
```

```
        nonmutating int getNumCamaras();
```

```
        nonmutating Camara getCamaraAt(int index) throws ErrorRango;
```

```
        nonmutating bool getPTZAt(int index) throws ErrorRango;
```

```
        nonmutating bool getActivaAt(int index) throws ErrorRango;
```

```
        nonmutating string getDescripcionAt(int index) throws ErrorRango;
```

```
        nonmutating string getIPAt(int index) throws ErrorRango;
```

```
        nonmutating int getPortAt(int index) throws ErrorRango;
```

```

        nonmutating bool getAudioAt(int index) throws ErrorRango;

        nonmutating bool getMovimientoAt(int index) throws ErrorRango;

        nonmutating bool getNotificarMDAt(int index) throws ErrorRango;

        nonmutating bool getGrabarMDAt(int index) throws ErrorRango;

        idempotent void setPTZAt(int index, bool newptz) throws ErrorRango;

        idempotent void setActivaAt(int index, bool newactiva, string ip) throws ErrorRango;

        idempotent bool setDescriptionAt(int index, string newdescripcion, string ip) throws
ErrorRango;

        idempotent void setIPAt(int index, string newip) throws ErrorRango;

        idempotent void setPortAt(int index, int newport) throws ErrorRango;

        idempotent void setAudioAt(int index, bool newaudio) throws ErrorRango;

        idempotent void setMovimientoAt(int index, bool newmovimiento) throws
ErrorRango;

        idempotent void setNotificarMDAt(int index, bool newnotificarmd) throws ErrorRango;

        idempotent void setGrabarMDAt(int index, bool newgrabarmd) throws ErrorRango;

        idempotent bool addCamara(Camara c, string ip);

        idempotent void deleteCamaraAt(int index, string ip) throws ErrorRango;

        idempotent void connect(string ip);

        idempotent void disconnect(string ip);

    };
};

```

### struct Camara

- **Descripción:** Contiene la información asociada a una cámara
- **Des:** Contiene la descripción asociada a la cámara: habitacion1...
- **IP:** IP donde se encuentra alojado el servicio de vídeo de la cámara
- **Port:** Puerto TCP de acceso al servicio de vídeo
- **PTZ:** Indica si la cámara es de tipo PTZ (con control de movimiento)
- **Active:** Informa del estado de la cámara: true activa y false inactiva
- **Audio:** Informa de si la cámara dispone de capacidad de audio o no
- **Movimiento:** Informa de si la cámara dispone de detección de movimiento o no
- **Notificarmd:** Indica si se debe notificar a los clientes ante una detección de movimiento
- **Grabarmd:** Indica si se debe grabar vídeo ante una detección de movimiento

**exception ErrorRango**

- **Descripción:** Indica que se ha introducido un índice erróneo al acceder a la lista de cámaras
- **index:** Índice introducido
- **maxindex:** Máximo índice permitido

**int getNumCamaras();**

- **Descripción:** Devuelve el numero de cámaras disponibles
- **Retorno:** Número de cámaras disponibles

**Camara getCamaraAt(int index) throws ErrorRango;**

- **Descripción:** Proporciona acceso a una determinada cámara
- **Index:** Índice de la cámara a la que se desea acceder
- **Retorno:** Cámara posicionada en index

**bool getPTZAt(int index) throws ErrorRango;**

- **Descripción:** Proporciona acceso al atributo ptz de una determinada cámara
- **Index:** Índice de la cámara a la que se desea acceder
- **Retorno:** Indicador de si la cámara es PTZ o no

**bool getActivaAt(int index) throws ErrorRango;**

- **Descripción:** Proporciona acceso al atributo activa de una determinada cámara
- **Index:** Índice de la cámara a la que se desea acceder
- **Retorno:** Indicador de si la cámara esta activa o no

**string getDescripcionAt(int index) throws ErrorRango;**

- **Descripción:** Proporciona acceso al atributo descripción de una determinada cámara
- **Index:** Índice de la cámara a la que se desea acceder
- **Retorno:** Descripción de la cámara

**string getIPAt(int index) throws ErrorRango;**

- **Descripción:** Proporciona acceso al atributo ip de una determinada cámara
- **Index:** Índice de la cámara a la que se desea acceder
- **Retorno:** IP de la cámara

**int getPortAt(int index) throws ErrorRango;**

- **Descripción:** Proporciona acceso al atributo port de una determinada cámara
- **Index:** Índice de la cámara a la que se desea acceder
- **Retorno:** Puerto TCP de acceso a la cámara

**bool getAudioAt(int index) throws ErrorRango;**

- **Descripción:** Proporciona acceso al atributo audio de una determinada cámara
- **Index:** Índice de la cámara a la que se desea acceder
- **Retorno:** Atributo audio de la cámara

**bool getMovimientoAt(int index) throws ErrorRango;**

- **Descripción:** Proporciona acceso al atributo movimiento de una determinada cámara
- **Index:** Índice de la cámara a la que se desea acceder
- **Retorno:** Atributo movimiento de la cámara

**bool getNotificarMDAt(int index) throws ErrorRango;**

- **Descripción:** Proporciona acceso al atributo notificarmd de una determinada cámara
- **Index:** Índice de la cámara a la que se desea acceder
- **Retorno:** Atributo notificarmd de la cámara

**bool getGrabarMDAt(int index) throws ErrorRango;**

- **Descripción:** Proporciona acceso al atributo grabarmd de una determinada cámara
- **Index:** Índice de la cámara a la que se desea acceder
- **Retorno:** Atributo grabarmd de la cámara

**void setPTZAt(int index, bool newptz) throws ErrorRango;**

- **Descripción:** Establece un nuevo valor para el atributo ptz de la cámara indicada
- **Index:** Índice de la cámara a la que se desea acceder
- **Newptz:** Nuevo valor del atributo

**void setActivaAt(int index, bool newactiva, string ip) throws ErrorRango;**

- **Descripción:** Establece un nuevo valor para el atributo activa de la cámara indicada
- **Index:** Índice de la cámara a la que se desea acceder
- **Newactiva:** Nuevo valor del atributo
- **IP:** IP del cliente que realiza la operación

**bool setDescriptionAt(int index, string newdescripcion, string ip) throws ErrorRango;**

- **Descripción:** Establece un nuevo valor para el atributo descripción de la cámara indicada
- **Index:** Índice de la cámara a la que se desea acceder
- **Newdescripcion:** Nuevo valor del atributo
- **IP:** IP del cliente que realiza la operación
- **Retorno:** true si la descripción se ha modificado correctamente o false en caso contrario

**void setIPAt(int index, string newip) throws ErrorRango;**

- **Descripción:** Establece un nuevo valor para el atributo ip de la cámara indicada
- **Index:** Índice de la cámara a la que se desea acceder
- **Newip:** Nuevo valor del atributo

**void setPortAt(int index, int newport) throws ErrorRango;**

- **Descripción:** Establece un nuevo valor para el atributo port de la cámara indicada
- **Index:** Índice de la cámara a la que se desea acceder
- **Newport:** Nuevo valor del atributo

**void setAudioAt(int index, bool newaudio) throws ErrorRango;**

- **Descripción:** Establece un nuevo valor para el atributo audio de la cámara indicada
- **Index:** Índice de la cámara a la que se desea acceder
- **Newaudio:** Nuevo valor del atributo

**void setMovimientoAt(int index, bool newmovimiento) throws ErrorRango;**

- **Descripción:** Establece un nuevo valor para el atributo movimiento de la cámara indicada
- **Index:** Índice de la cámara a la que se desea acceder
- **Newmovimiento:** Nuevo valor del atributo

**void setNotificarMDAt(int index, bool newnotificarmd) throws ErrorRango;**

- **Descripción:** Establece un nuevo valor para el atributo notificarmd de la cámara indicada
- **Index:** Índice de la cámara a la que se desea acceder
- **Newnotificarmd:** Nuevo valor del atributo

**void setGrabarMDAt(int index, bool newgrabarmd) throws ErrorRango;**

- **Descripción:** Establece un nuevo valor para el atributo grabarmd de la cámara indicada
- **Index:** Índice de la cámara a la que se desea acceder.
- **Newgrabarmd:** Nuevo valor del atributo.

**bool addCamara(Cámara c, string ip);**

- **Descripción:** Añade una nueva cámara en la ultima posición de la lista de cámaras actual.
- **c:** Cámara a insertar.
- **IP:** IP del cliente que realiza la operación.

**void deleteCamaraAt(int index, string ip) throws ErrorRango;**

- **Descripción:** Elimina la cámara situada en la posición indicada.
- **Index:** Índice de la cámara que se desea eliminar.
- **IP:** IP del cliente que realiza la operación

**void connect(string ip);**

- **Descripción:** Registra un cliente en la lista de usuarios conectados
- **IP:** IP del cliente

**void disconnect(string ip);**

- **Descripción:** Elimina un cliente de la lista de usuarios conectados
- **IP:** IP del cliente

### 5.1.3 Clientes

Interfaz del servant que ofrecen los clientes, se corresponde con la clase abstracta *cliente* vista en los diagramas de clases y es implementado por la clase *clientel*, que hereda de la anterior. Esta implementación, al contrario que en los dos interfaces anteriores y que la especificación del interfaz, no es única: existe una para los clientes que se conectan desde un PC y otra para los que lo hacen desde dispositivos móviles (la primera se implementará con C# y la segunda con MFC).

```
module Cliente
```

```
{  
  
    interface CClie  
    {  
  
        void Evento(string dir, int valor);  
  
        idempotent void camAdded(string des);  
  
        idempotent void camDeleted(int index);  
  
        idempotent void camDesChanged(int index, string newdes);  
  
        idempotent void camStateChanged(int index, bool newstate);  
  
        nonmutating void motionDetected(string name);  
  
        nonmutating void motionEnded(string name);  
  
    };  
};
```

**void Evento(string dir, int valor);**

- **Descripción:** Evento ocurrido en la instalación domótica
- **Dir:** Dirección en la cual ha ocurrido el evento
- **Valor:** Dirección en la cual ha ocurrido el evento

**void camAdded(string des);**

- **Descripción:** Nueva cámara añadida al sistema
- **Des:** Descripción de la nueva cámara

**void camDeleted(int index);**

- **Descripción:** Cámara eliminada del sistema

- **Index:** Índice de la cámara eliminada

**void camDesChanged(int index, string newdes);**

- **Descripción:** Descripción de cámara modificada
- **Index:** Índice de la cámara modificada
- **Newdes:** Nueva descripción de la cámara

**void camStateChanged(int index, bool newstate);**

- **Descripción:** Estado de cámara modificado
- **Index:** Índice de la cámara modificada
- **Newstate:** Nuevo estado de la cámara

**void motionDetected(string name);**

- **Descripción:** Movimiento detectado en una determinada cámara
- **Name:** Nombre de la cámara donde se ha detectado el movimiento

**void motionEnded(string name);**

- **Descripción:** Fin de movimiento detectado en una determinada cámara
- **Name:** Nombre de la cámara donde se ha terminado de registrar movimiento

## 5.2 API

La documentación de la API se ha generado con Doxygen y se incluye en la distribución del proyecto en formato HTML, RTF y LaTeX. Es accesible desde los directorios incluidos en la distribución de cada proyecto de la aplicación con el formato *NombreProyectoDoc*.

Doxygen es un generador de documentación para diversos lenguajes de programación (C++, C#, Java...), ampliamente probado y utilizado, capaz de generar documentación en formato LaTeX, HTML, RTF, CHM, PDF, PostScript y Man.

La información se extrae de etiquetas introducidas en el código fuente que actúan a modo de metadatos sobre los diversos elementos de éste.

Además, Doxygen utiliza el lenguaje de descripción de grafos *Dot* y el conjunto de aplicaciones de tratamiento de grafos *GraphViz* para generar diagramas (relacionales, UML, etc....)



### 5.2.1 Ejemplo: DomoUEXCamera

A modo de ejemplo, vamos a ver la documentación generada por Doxygen para la clase DomoUEXCamera. Hay que tener en cuenta que, en el caso de HTML, todos los nombres de métodos, atributos, clases... se muestran como enlaces navegables.

#### 5.2.1.1 Diagrama de colaboración

La primera sección muestra un diagrama en el que se pueden ver las relaciones entre clases colaboradoras.

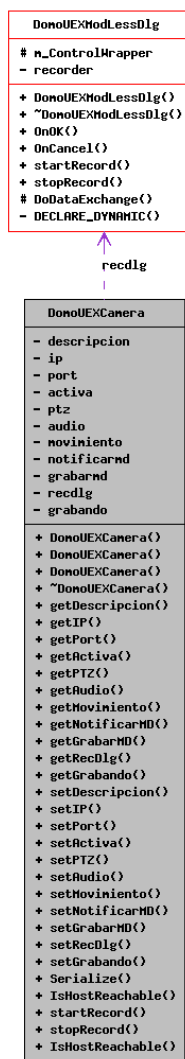
<a href="#">Página principal</a>	<a href="#">Namespaces</a>	<a href="#">Clases</a>	<a href="#">Archivos</a>
<a href="#">Lista de componentes</a>	<a href="#">Jerarquía de la clase</a>	<a href="#">Miembros de las clases</a>	

#### Referencia de la Clase DomoUEXCamera

Objeto que representa una camara. [Más...](#)

```
#include <DomoUEXCamera.h>
```

Diagrama de colaboración para DomoUEXCamera:



### 5.2.1.2 Descripción general de métodos

En esta sección se muestran los métodos que componen la clase junto con la descripción proporcionada para ellos (etiqueta \brief de los metadatos).

[Lista de todos los miembros.](#)

#### Métodos públicos

	<b>DomoUEXCamera</b> (void) <i>Constructor por defecto.</i>
	<b>DomoUEXCamera</b> (CString &_descripcion, CString &_ip, int _port, bool _activa, bool _ptz, bool _audio, bool _movimiento, bool _notificarmd, bool _grabarmd) <i>Constructor parametrizado.</i>
	<b>DomoUEXCamera</b> ( <b>DomoUEXCamera</b> *camera) <i>Constructor por copia.</i>
	<b>~DomoUEXCamera</b> (void) <i>Destructor.</i>
CString	<b>getDescription</b> () <i>Proporciona acceso al atributo descripcion de la camara.</i>
CString	<b>getIP</b> () <i>Proporciona acceso al atributo ip de la camara.</i>
int	<b>getPort</b> () <i>Proporciona acceso al atributo port de la camara.</i>
bool	<b>getActiva</b> () <i>Proporciona acceso al atributo activa de la camara.</i>
bool	<b>getPTZ</b> () <i>Proporciona acceso al atributo ptz de la camara.</i>
bool	<b>getAudio</b> () <i>Proporciona acceso al atributo audio de la camara.</i>
bool	<b>getMovimiento</b> () <i>Proporciona acceso al atributo movimiento de la camara.</i>
bool	<b>getNotificarMD</b> () <i>Proporciona acceso al atributo notificarmd de la camara.</i>
bool	<b>getGrabarMD</b> () <i>Proporciona acceso al atributo grabarmd de la camara.</i>
<b>DomoUEXModLessDlg</b> *	<b>getRecDlg</b> () <i>Proporciona acceso al atributo recdlg de la camara.</i>
bool	<b>getGrabando</b> () <i>Proporciona acceso al atributo grabando de la camara.</i>
void	<b>setDescription</b> (CString &_descripcion) <i>Establece un nuevo valor para el atributo descripcion de la camara indicada.</i>
void	<b>setIP</b> (CString &_ip) <i>Establece un nuevo valor para el atributo ip de la camara indicada.</i>
void	<b>setPort</b> (int _port) <i>Establece un nuevo valor para el atributo port de la camara indicada.</i>
void	<b>setActiva</b> (bool _activa) <i>Establece un nuevo valor para el atributo activa de la camara indicada.</i>
void	<b>setPTZ</b> (bool _ptz) <i>Establece un nuevo valor para el atributo ptz de la camara indicada.</i>
void	<b>setAudio</b> (bool _audio) <i>Establece un nuevo valor para el atributo audio de la camara indicada.</i>
void	<b>setMovimiento</b> (bool _movimiento) <i>Establece un nuevo valor para el atributo movimiento de la camara indicada.</i>
void	<b>setNotificarMD</b> (bool _notificarmd) <i>Establece un nuevo valor para el atributo notificarmd de la camara indicada.</i>
void	<b>setGrabarMD</b> (bool _grabarmd) <i>Establece un nuevo valor para el atributo grabarmd de la camara indicada.</i>
void	<b>setRecDlg</b> ( <b>DomoUEXModLessDlg</b> *_recdlg) <i>Establece un nuevo valor para el atributo recdlg de la camara indicada.</i>
void	<b>setGrabando</b> (bool _grabando) <i>Establece un nuevo valor para el atributo grabando de la camara indicada.</i>
void	<b>Serialize</b> (CArchive *archivo) <i>Sobreescritura de la función Serialize para escribir objetos en ficheros.</i>
void	<b>IsHostReachable</b> () <i>Método encargado de averiguar si la cámara se encuentra activa.</i>
void	<b>startRecord</b> (CString login, CString password) <i>Comienza la grabacion del video emitido por la camara.</i>
void	<b>stopRecord</b> () <i>Finaliza la grabacion de video.</i>

#### Métodos públicos estáticos

static UINT	<b>IsHostReachable</b> (LPVOID pParam) <i>Método de transición al que accede el hilo de ejecución.</i>
-------------	---

### 5.2.1.3 Descripción detallada de métodos

Aquí se detallan los métodos, incluyendo un grafo donde se puede ver qué métodos de la misma u otras clases llaman al método comentado y viceversa. Por cuestiones de espacio sólo mostramos el detalle del método *setPTZ*.

```
void DomoUEXCamera::setPTZ ( bool _ptz ) [inline]
```

Establece un nuevo valor para el atributo ptz de la camara indicada.

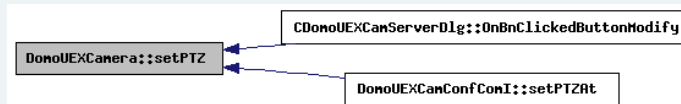
**Parámetros:**

*\_ptz* Nuevo valor del atributo.

Definición en la línea 218 del archivo [DomoUEXCamera.h](#).

Referenciado por [CDomoUEXCamServerDlg::OnBnClickedButtonModify\(\)](#), y [DomoUEXCamConfComI::setPTZAt\(\)](#).

Here is the caller graph for this function:



## 6 Manual de usuario

### 6.1 Instalación

QuercusDomoSystem utiliza diferentes herramientas, suministradas junto a la aplicación, para funcionar:

- Última versión de las librerías ICE o ICE-E (para dispositivos móviles)
- Librerías de acceso al bus eteC Falcon
- Axis Media Control

Una vez instaladas estas herramientas basta con copiar el directorio de distribución de cada parte del sistema (servidor de videovigilancia, servidor de instalación y cliente) en la ruta deseada.

### 6.2 Configuración

La aplicación de configuración del sistema permite cargar un plano desarrollado mediante cualquier aplicación de edición y configurar los dispositivos de los que disponemos en la instalación en dicho plano.

A continuación se muestra el entorno inicial visible cuando se abre la aplicación:

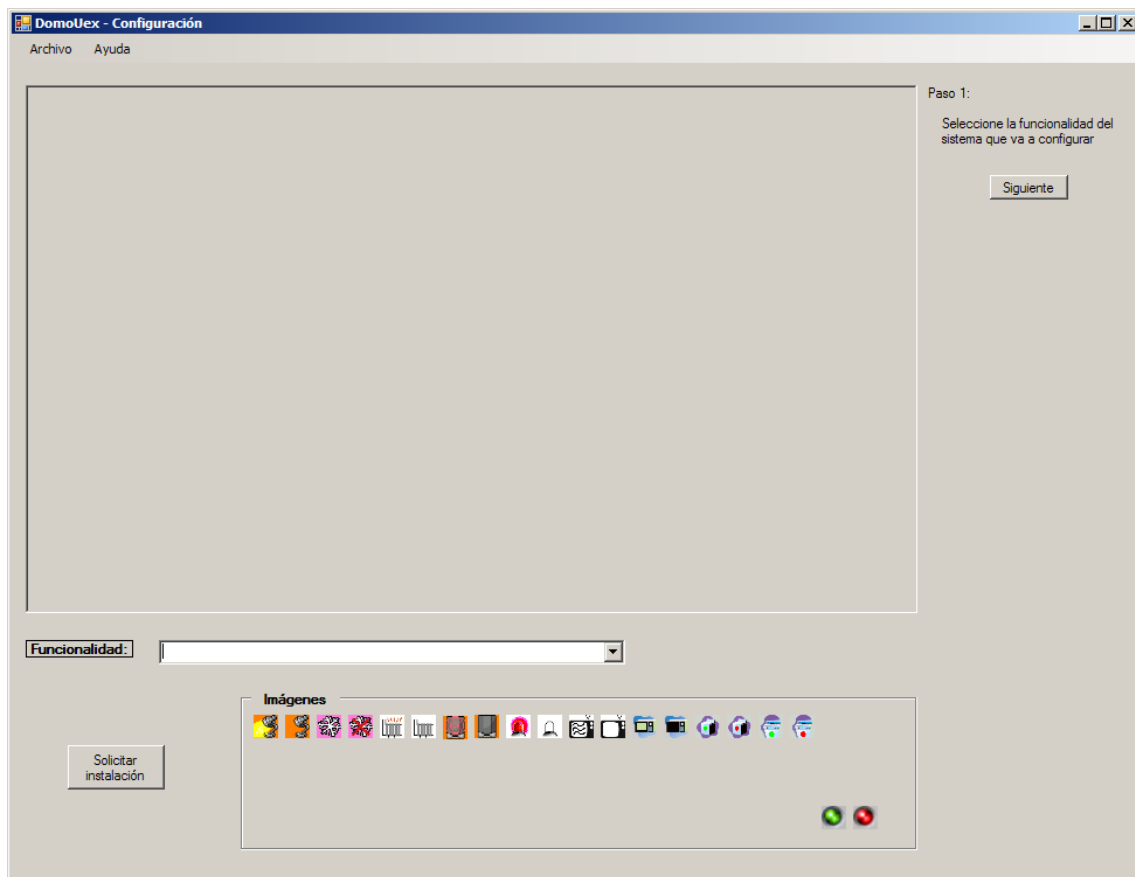


Ilustración 57. DUConfig

Se puede observar que existen tres partes claramente diferenciadas pero dependientes unas de otras. La primera de ellas es el lugar donde se carga la imagen del plano de la vivienda. La segunda es donde se presentan las funcionalidades configurables del sistema junto con una lista de imágenes para asociar. La tercera es la guía de configuración, mediante la cual, se dice al usuario los pasos a seguir para configurar su vivienda.

A continuación se muestran los pasos a seguir para realizar la configuración:

1. Archivo → Crear nuevo. Seleccionamos el plano de la vivienda y posteriormente el archivo donde se guardará la configuración que establezcamos.
2. Solicitar conexión: presionando el botón obtenemos las funcionalidades, desde el servidor, de la vivienda que vamos a configurar.
3. A partir de aquí, únicamente hay que ir realizando los pasos que se nos indican en la parte derecha del interfaz, e ir navegando por ellos.
4. Cuando se termina de configurar toda la instalación, Archivo→Terminar, y aparecen unos campos que hay que rellenar para establecer el nombre de usuario y contraseña para esa configuración.

### 6.3 Servidor instalación domótica

A continuación se muestra el interfaz gráfico del servidor de la instalación domótica, que será comentado posteriormente.

The screenshot shows the DomoUex application window. It is divided into two main sections: 'Dispositivos' (Devices) on the left and 'Direcciones de grupo' (Group addresses) on the right. The 'Dispositivos' section has a tab labeled 'Instalados' (Installed) and a dropdown menu. Below it is a 'Borrar' (Delete) button. There is also a 'Nuevo' (New) tab with input fields for 'Nombre:' (Name) and 'IP:', and a 'Guardar' (Save) button. The 'Direcciones de grupo' section has a tab labeled 'Configuradas' (Configured) and a dropdown menu. Below it is a 'Borrar' (Delete) button. There is also a 'Nueva' (New) tab with two dropdown menus for 'Dispositivo 1' and 'Dispositivo 2', an 'IP:' input field, a checkbox for 'Activa inicialmente:' (Initially active), and an 'Enlazar' (Link) button. A 'Salir' (Exit) button is located at the bottom center of the window.

Ilustración 58. Servidor instalación domótica

En el servidor, existen dos partes claramente diferenciadas, una está relacionada con los dispositivos del sistema y otra con las funcionalidades del mismo.

En la parte relacionada con los dispositivos, se permite la introducción de los dispositivos físicos de los que consta la instalación, para ello, se introduce el nombre del mismo y su dirección física y se presiona el botón Guardar. Al guardar el dispositivo en la aplicación, se introduce en la lista desplegable, por si se desea consultar o eliminar.

La parte relacionada con las funcionalidades de la instalación domótica permite la asociación de dispositivos para formar una funcionalidad. Los dispositivos se seleccionan en las listas desplegables, después de estar seleccionados, hay que introducir la dirección de grupo que se le asignará y el estado inicial de la misma, y después se presiona Enlazar y se almacena en la aplicación. La funcionalidad queda almacena en otra lista desplegable para poder consultarla o eliminarla si se desea.

## 6.4 Servidor videovigilancia

### 6.4.1 Configuración

La configuración inicial del servidor de videovigilancia se encuentra especificada en el fichero *camcom.cfg* (en el directorio donde se encuentra el ejecutable). Es recomendable no modificar el contenido de este fichero. No obstante, comentaremos cada uno de los campos por si en algún determinado momento fuese necesaria su edición:

- DomoUEXCamConfCom.Endpoints=tcp -p 10007  
Este campo define el puerto TCP donde el servidor atenderá las peticiones de los clientes (por defecto 10007).
- Ice.Warn.Connections=1  
Indica si el servidor debe informar acerca de fallos en las conexiones o no (1 ó 0).
- Ice.Trace.Network=3  
Indica el nivel de detalle que debe tener el servidor al informar acerca de fallos en la red (0 mínimo – 3 máximo).
- Ice.Trace.Protocol=1  
Indica si el servidor debe informar acerca de fallos en el protocolo de comunicación o no (1 ó 0).

### 6.4.2 Datos persistentes

El servidor de videovigilancia almacena varios tipos de datos en el sistema, a saber:

- *servidor.log*: este fichero contiene datos acerca de los eventos registrados por el servidor (conexión de clientes, movimientos detectados...)
- *ncamaras.cfg* y *camaras.cfg*: contienen la información relativa a la configuración actual del sistema de videovigilancia.
- MDdd\_MM\_YY\_hh\_mm\_ss.asf: los ficheros con este formato contienen vídeo grabado al detectar movimiento en una cámara configurada a tal efecto.

### 6.4.3 Interfaz

Este es el aspecto que presenta el interfaz del servidor de vídeo:

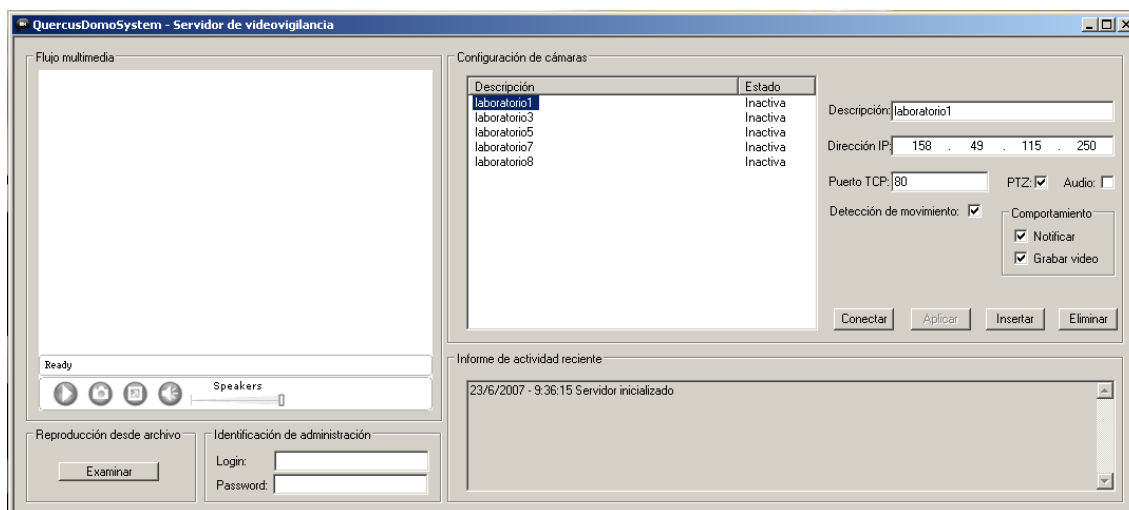


Ilustración 59. Servidor videovigilancia

En la parte izquierda de la pantalla disponemos de un control *Flujo multimedia* donde podemos interactuar con las cámaras del sistema (más adelante veremos cómo conectar con una cámara). Debajo de este se sitúan el botón *Examinar*, que permite cargar vídeo desde cualquier fichero de vídeo (por ejemplo desde los ficheros de movimiento detectado), y el campo de *Identificación de administración*, donde deben indicarse el nombre de usuario y contraseña de administrador del sistema para poder configurar las cámaras.

En el centro tenemos una lista donde se muestran las cámaras configuradas y en qué estado se encuentra cada una (activa o inactiva). Obviamente, no es posible conectar con una cámara inactiva.

A la derecha podemos ver los controles que nos permiten editar la configuración de las cámaras:

- Descripción: servirá para diferenciar las cámaras entre sí, debe ser un descriptor único.
- Dirección IP: IP de la cámara.
- Puerto TCP: puerto TCP donde la cámara atiende las conexiones (normalmente es el puerto 80).
- PTZ: indica si la cámara tiene capacidad para mover la lente, hacer zoom, recorridos, etc.
- Audio: indica si la cámara puede transmitir audio.
- Detección de movimiento: si se activa el sistema interpreta que la cámara puede detectar movimientos.
- Notificar: si la cámara dispone de detección de movimiento y se activa esta opción los clientes serán notificados cuando se detecte algún movimiento.
- Grabar: si la cámara dispone de detección de movimiento y se activa esta opción el sistema almacenará el vídeo correspondiente a la detección del movimiento.

Debajo de estos controles tenemos los botones que nos permiten conectar con la cámara seleccionada en la lista, aplicar los cambios realizados en la configuración de



dicha cámara (hasta que no se pulse este botón los cambios no quedarán almacenados en el sistema), insertar una nueva cámara o eliminar la cámara seleccionada.

## 6.5 Cliente

El cliente del sistema está dividido en dos partes, una está relacionada con la instalación domótica y otra está relacionada con la videovigilancia del sistema.

### 6.5.1 Instalación domótica

El siguiente interfaz es el de la parte relacionada con la instalación domótica.

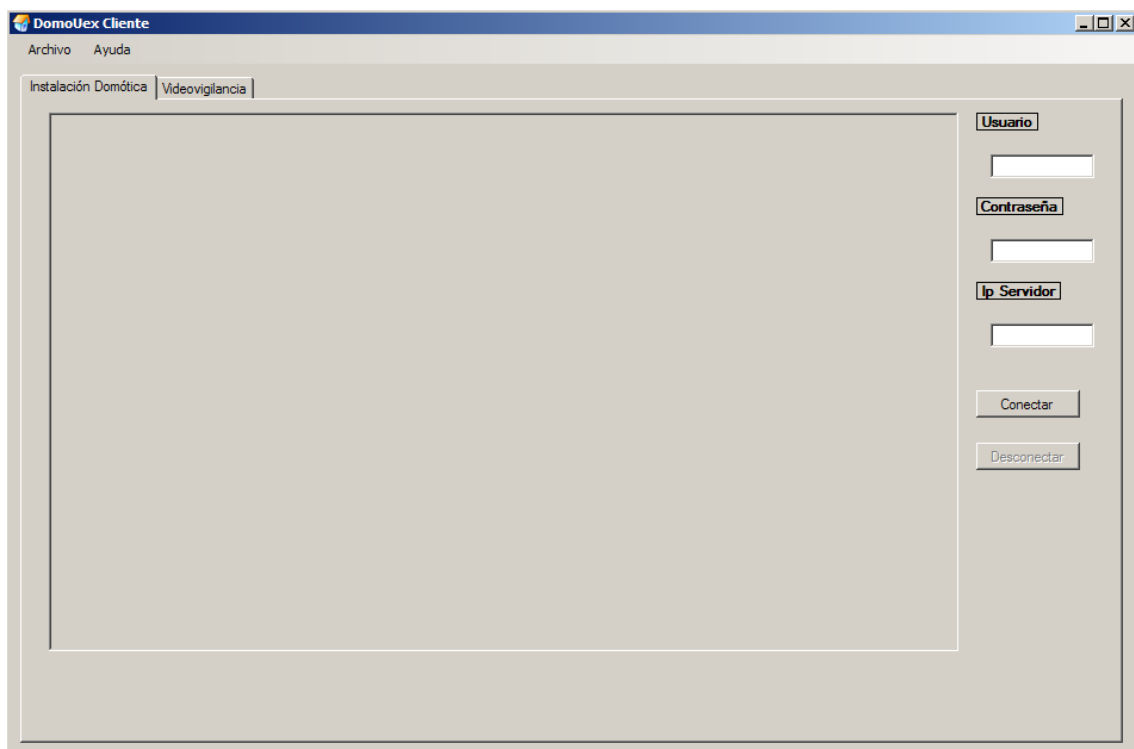


Ilustración 60. Cliente - Instalación domótica

En el interfaz se presentan tres cuadros que hay que rellenar para poder interactuar con el servidor, o lo que es lo mismo, con la instalación domótica. El primero de ellos, Usuario, hace referencia al nombre de usuario que quiere cargar su configuración, el segundo, Contraseña, se refiere a la contraseña del usuario y el tercero, y último, es la IP donde reside el servidor. La dirección IP del servidor es opcional si se ejecutan en la misma máquina cliente y servidor.

Después de rellenar los campos anteriormente citados, se procede a realizar la conexión con el servidor presionando el botón Conectar. Tras esto, aparecerá, en la parte donde se carga la imagen, nuestro plano de la vivienda junto con los dispositivos que configuramos con la herramienta descrita anteriormente.

Para interactuar con la vivienda, lo único que hay que hacer es hacer click en el dispositivo que queremos que cambie de estado.

Cuando cerramos la aplicación, hay que desconectarse del servidor, y para ello hay que presionar el botón Desconectar.

### 6.5.2 Videovigilancia

Este es el interfaz que presenta la ventana de vídeo vigilancia:

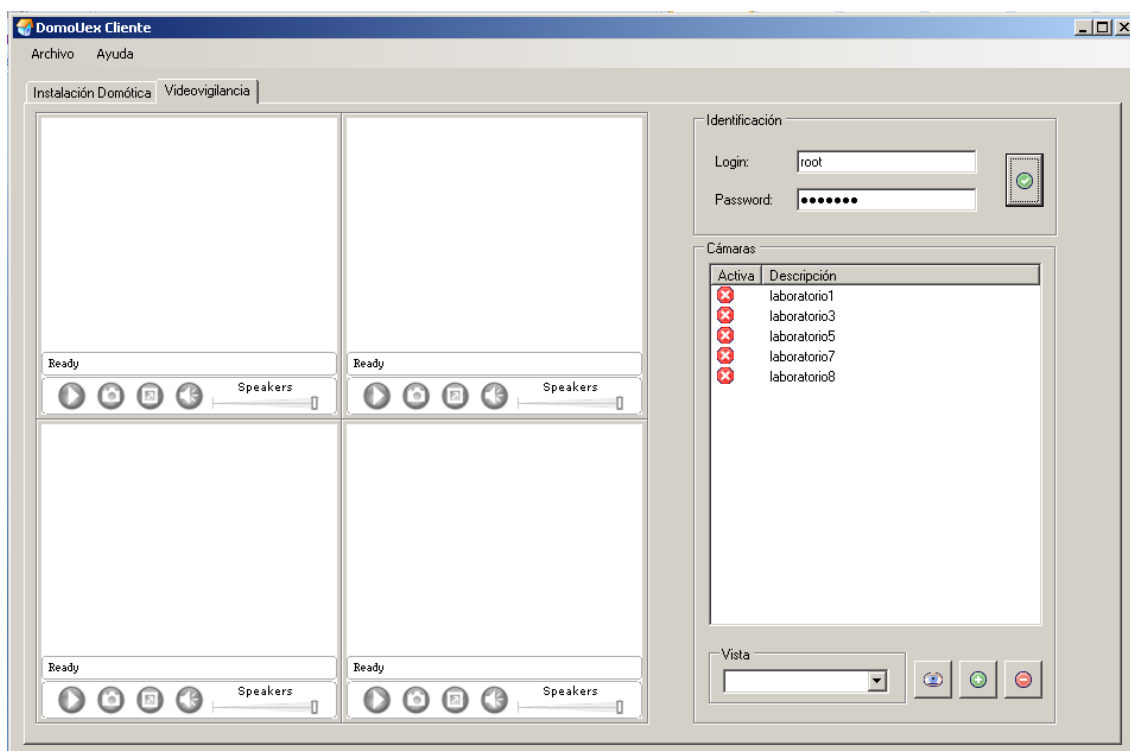


Ilustración 61. Cliente - Videovigilancia

En la parte izquierda tenemos los controles donde se podrá interactuar con las cámaras del sistema.

En la esquina superior derecha tenemos los campos donde deberán introducirse los datos de usuario de cámaras: nombre de usuario y contraseña. Una vez validadas tendremos acceso al panel de *Cámaras* (debajo de los controles anteriores).

En este panel podemos ver una lista con las cámaras disponibles en el sistema y el estado en que se encuentran (activas o inactivas). Debajo hay una lista desplegable donde podemos seleccionar el control donde queremos trabajar con la cámara seleccionada y, a su derecha, un botón para conectar con la cámara (icono de un ojo), otro para añadir una nueva cámara al sistema y otro para eliminar la cámara seleccionada.

Tanto si hacemos doble click sobre una cámara de la lista como si pulsamos el botón de añadir una nueva cámara el cliente mostrará un panel de configuración de cámara (en el primer caso con los datos de la cámara seleccionada y en el segundo vacío para que indiquemos los de la nueva cámara).

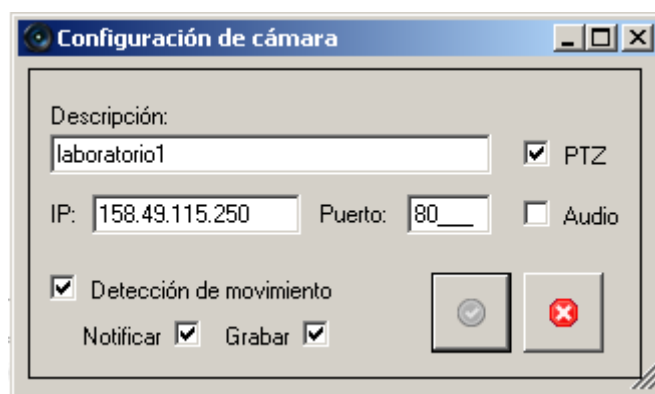


Ilustración 62. Cliente - Configuración de cámara

Estos campos son los mismos que se comentaron anteriormente al describir el servidor de videovigilancia (ver sección 6.5.3 *Interfaz*).

## 6.6 Dispositivos móviles

El primer paso al iniciar el cliente para dispositivos móviles es indicar la dirección IP de los servidores (puesto que sólo se encuentra implementada la parte de videovigilancia sólo será necesario indicar donde se encuentra el servidor de ésta).

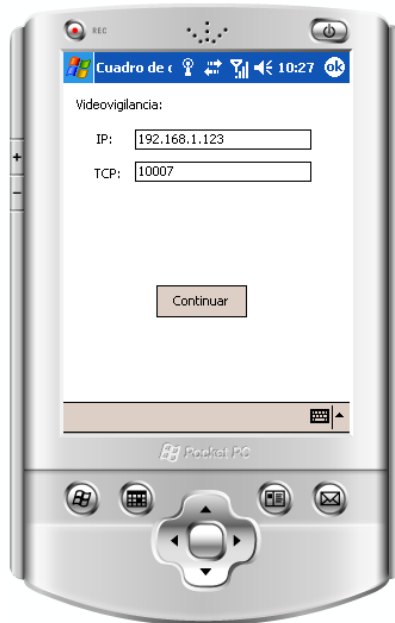


Ilustración 63. Configuración servidores para dispositivos móviles

Una vez validados estos datos se accede a la interfaz principal que está dividido en dos partes: *Instalación* (no implementada aún) y *Videovigilancia*.

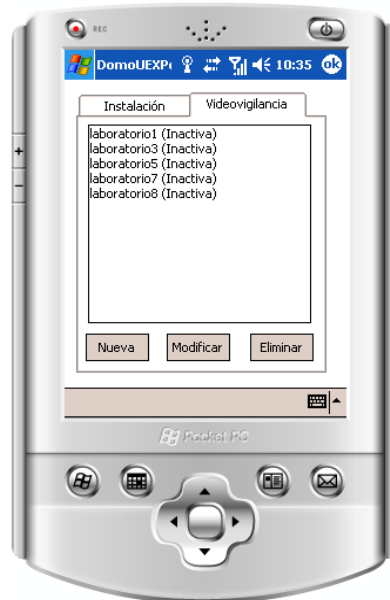


Ilustración 64. Interfaz dispositivos móviles

La parte de videovigilancia es similar a la explicada para el cliente de PC (ver sección 6.6.2 *Videovigilancia*) con algunas diferencias: al hacer doble click sobre una cámara de la lista estaremos indicando que queremos conectar con ella (para lo que se abrirá una nueva ventana) y para editar su configuración deberá utilizarse el botón *Modificar* (abriéndose entonces el panel de configuración de cámaras).

## 7 De DomoUEx a QuercusDomoSystem

Una vez terminado el desarrollo, se exponen las mejoras que ha supuesto QuercusDomoSystem respecto al sistema anterior centrándonos en la arquitectura y funcionalidades de ambos sistemas.

### 7.1 Arquitectura

- DomoUEx fue desarrollado utilizando el lenguaje de programación Java, de Sun Microsystems, y el middleware de comunicaciones CORBA, de la OMG, mientras que en el nuevo sistema nos decantamos por utilizar Visual Studio .NET, de Microsoft, y el middleware ICE, de Zeroc.
- En QuercusDomoSystem hemos prescindido de las librerías EIBlet Engine, de JNetSystem, que utilizaba el anterior proyecto. Estas eran propietarias y ya no se daba soporte por parte de la compañía. Las hemos implementado, en gran parte, ampliando nuevos aspectos. Gracias a esto, hemos eliminado una capa y la hemos embebido en la capa de aplicación de nuestro sistema.
- Debido a la eliminación de las librerías anteriores, en el nuevo proyecto se ha introducido la licencia que poseemos de las librerías Falcon de acceso al bus. Esto se debe a que al eliminar las librerías EIBlet Engine utilizadas en DomoUEx, se interactúa de manera directa con las Falcon.
- Mientras que en DomoUEx existía único servidor, en QuercusDomoSystem conviven dos servidores, uno para la instalación domótica y otro para videovigilancia.

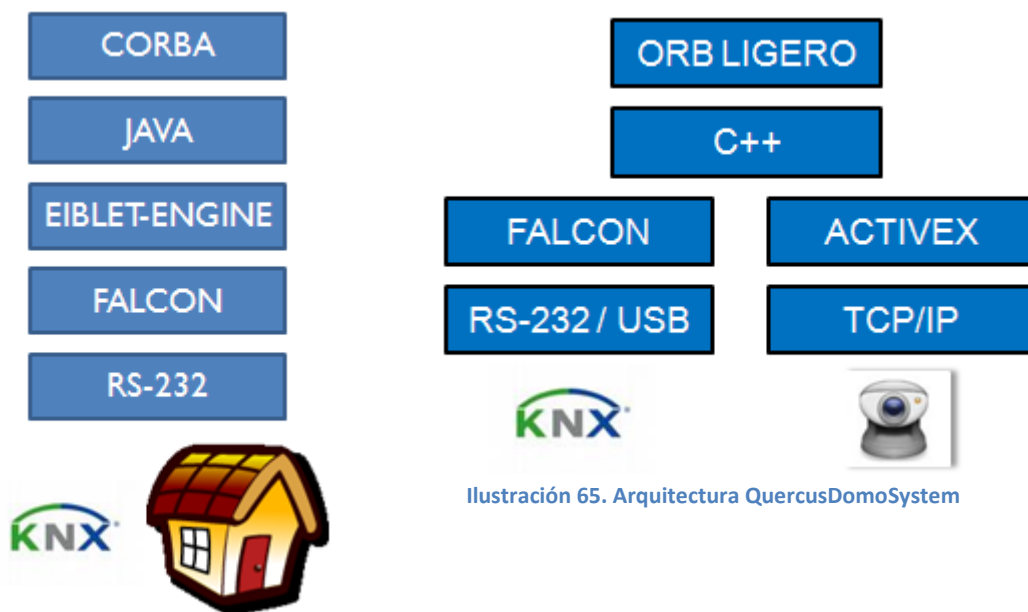


Ilustración 65. Arquitectura QuercusDomoSystem

Ilustración 66. Arquitectura DomoUEx

## 7.2 Funcionalidades

Los principales avances respecto a la funcionalidad son los siguientes:

- QuercusDomoSystem es un sistema independiente de la instalación domótica que se tenga y del modo de acceso a la misma, mientras que DomoUEX es un sistema estático, es decir, dependiente de la instalación domótica y del modo de acceso.
- El módulo de vigilancia de QuercusDomoSystem es un módulo completamente nuevo que antes no existía en DomoUEX, y ofrece posibilidades de todo tipo en lo referente a videovigilancia.

Existen otras diferencias funcionales importantes, no tanto como las anteriores, pero que hay que destacar:

- El proyecto antiguo, DomoUEX, sólo permite acceder al bus desde el puerto serie de nuestro ordenador, y QuercusDomoSystem ofrece además la posibilidad de conexión por USB.
- QuercusDomoSystem permite la configuración de planos realizados en AutoCad y adaptarlos a gusto del usuario.
- En QuercusDomoSystem se ofrece la posibilidad de realizar comunicaciones por voz bidireccional cuando el usuario lo desee.
- DomoUEX permite la interacción con la instalación domótica a nivel de byte, mientras que en QuercusDomoSystem todavía está en proceso de desarrollo.

## 8 Herramientas utilizadas

Nombre	Versión	Descripción
<b>Visual Studio .NET</b>	2005	Plataforma de desarrollo
<b>ICE</b>	3.1.1 & 3.1.2	Middleware
<b>ICE-E</b>	1.1.0	Middleware (dispositivos móviles)
<b>Microsoft Office</b>	2007	Procesador de texto
<b>CmapTools</b>	4.09	Editor de mapas conceptuales
<b>Doxygen</b>	1.5.1	Generador de documentación
<b>ETS</b>	3	Administrador de instalaciones domóticas
<b>Dia</b>	0.96.1	Editor de diagramas
<b>eteC Falcon</b>	1.3	Librerías de acceso al bus

## 9 Trabajo futuro

Son muchas las posibilidades que ofrecen la domótica y la videovigilancia a la hora de imaginar ampliaciones y mejoras para este proyecto:

- **Seguridad:** ampliar las funciones de seguridad del sistema es una tarea crítica para el futuro de QuercusDomoSystem. Convendría estudiar sistemas que garanticen la confidencialidad, autenticidad e integridad de los datos. Algunas opciones podrían ser el uso de dispositivos físicos que garanticen la no suplantación del usuario (como lectores de huellas dactilares), sistemas de transmisión codificada (SSL), algoritmos de clave pública...
- **Interfaz web:** sería interesante disponer de una vía de acceso web al sistema. La complejidad de esta posibilidad depende de los requisitos que queramos marcarnos: desde un sistema simple de consulta-respuesta, fácilmente implementable mediante servicios web, a un sistema con características similares al desarrollado para .NET, donde tendríamos que estudiar las opciones de las *Rich Internet Applications* (RIA) y las nuevas tecnologías web como Flex, Ajax, Laszlo, etc.
- **Escenarios:** entendemos como escenario la posibilidad de combinar diferentes funcionalidades para generar una determinada situación en el sistema. Por ejemplo, podríamos definir un escenario en el que al detectar un movimiento en el jardín se cerrasen todas las cerraduras de puertas y ventanas u otro en el que al llegar las ocho de la tarde se encendiese la calefacción. Estos problemas son de naturaleza totalmente lógica por lo que, a día de hoy, y con el núcleo de funcionalidades que proporciona QuercusDomoSystem no debería ser muy costoso abordar su realización. Sería interesante plantear el problema desde el punto de vista del procesamiento de lenguajes creando un lenguaje escrito (posteriormente ampliable con un lenguaje visual) de especificación de escenarios donde el usuario pudiese indicar sus requerimientos. En éste caso, el problema principal sería abordar la construcción del intérprete para ese lenguaje.

Además, la integración del control de la instalación domótica en dispositivos móviles, un punto importante del desarrollo, ha quedado pospuesto por falta de tiempo. Hay que mencionar que, una vez introducidas las librerías ICE, el resto del proceso sería únicamente cuestión de diseñar una interfaz adaptada a este tipo de entornos.



## 10 Personal

Desde el inicio de este proyecto hasta el día de hoy ha pasado más de un año. Durante este tiempo nos hemos enfrentado a muchos de los retos que supone la realización de un proyecto de dimensiones reales: investigación, organización, resolución de problemas en el desarrollo...

Son varios los puntos que queremos comentar en este capítulo por lo que los hemos decidido dividirlo en secciones que presentamos a continuación.

### 10.1 Trabajo en equipo

Hemos intentado mantener siempre una buena coordinación colaborando en los puntos comunes y manteniendo la independencia en aquellas partes del proyecto que correspondían a cada uno. Estamos convencidos de que esta experiencia es enriquecedora para los estudiantes ya que es esencial para cualquier ingeniero informático aprender a convivir en el desarrollo, superar diferencias, organizar el trabajo de forma eficiente y, en resumen, saber trabajar en equipo. Por esto, nuestra propuesta es que los proyectos fin de carrera que involucren a más de un estudiante dejen de ser un caso excepcional para convertirse en la norma habitual.

### 10.2 Equipo

En lo referente a los medios físicos utilizados para el desarrollo del proyecto, hemos tenido a nuestra disposición varios ordenadores preparados para el desarrollo del proyecto así como una cámara IP AXIS 212 PTZ.

Respecto a los dispositivos domóticos, hemos contado con una instalación de pruebas pequeña. Quizás hubiese sido más interesante tener acceso a una instalación de mayor tamaño para poder comprobar el funcionamiento del sistema en un entorno más real.

### 10.3 Investigación

Fue mucho el tiempo dedicado a la fase de investigación de las diversas tecnologías que se incluyen o podrían haberse incluido en el proyecto. Sería imposible detallar la cantidad de documentos consultados en los meses dedicados a esta etapa y es difícil plasmar todo ese esfuerzo en este documento. No obstante, y visto desde la distancia, es fundamental aportar todo el trabajo posible a la fase de investigación puesto que, más tarde, será crítico a la hora de tomar decisiones sobre la forma de afrontar el desarrollo.

Recomendamos a los alumnos que comiencen a plantearse cuál es el ámbito sobre el que quieren desarrollar sus proyectos antes de llegar al último año de carrera y comiencen a profundizar en él para poder tener una visión global de cómo enfrentarse a él.

## 10.1 Conocimientos adquiridos

Debemos de estar agradecidos por el proyecto que se nos dio la oportunidad de realizar, ya que ha sido enriquecedor en cuanto a conocimientos técnicos adquiridos. Hemos explorado tecnologías que, hasta la fecha, apenas conocíamos. A continuación presentamos los conocimientos aprendidos de mayor relevancia:

- Creación, configuración, programación y mantenimientos de sistemas distribuidos mediante el middleware ICE y ICE-E.
- Control avanzado de cámaras de videovigilancia.
- Gestión y realización de proyectos mediante la plataforma Visual Studio .NET.
- Uso de emuladores de dispositivos móviles con fines de desarrollo.
- Programación utilizando las librerías Falcon de KNX.
- Manipulación de controles ActiveX.
- Creación física de instalaciones domóticas.
- Programación de instalaciones domóticas mediante ETS3.

## 10.2 Futuro

Nuestra aportación a QuercusDomoSystem no termina por ahora, son muchos los objetivos futuros que podemos plantearnos para mejorar el sistema y convertirlo en una aplicación mejor. Es por esto que seguiremos desarrollando para mejorar esta serie de aplicaciones y, si puede ser, que llegue a más algún día y no quede dentro del ámbito de la Universidad de Extremadura.

## 11 Agradecimientos

A nuestras familias por todo el apoyo que nos han dado desde el primer día hasta el último.

A M<sup>a</sup> del Pilar y M<sup>a</sup> Rosa por su paciencia y comprensión y por estar siempre ahí.

A José María Conejero Manzano por aportarnos su experiencia y por todos esos minutos que le hemos robado poco a poco.

A Juan Hernández Núñez por confiar en nosotros para realizar este proyecto.

## 12 Referencias

Detallar cada documento consultado durante el desarrollo de este proyecto, se extendería a través de innumerables páginas por lo que hemos decidido hacer resaltar aquellas fuente que han servido como una referencia constante y no puntual.

**The Code Project** (<http://www.codeproject.com>) Comunidad de desarrollo para Visual Studio y .NET.

**CodeGuru** (<http://www.codeguru.com>) Comunidad de desarrollo para Visual Studio y .NET.

**Wikipedia** (<http://www.wikipedia.org>) Enciclopedia libre.

**MSDN** (<http://msdn2.microsoft.com>) Documentación oficial de Visual Studio .NET.

**Axis Communications** (<http://www.axis.com>) Web de la empresa Axis Communications.

**Domótica Net** ([www.domotica.net](http://www.domotica.net)) Web de temas relacionados con la domótica.

## ANEXO 1: ETS3

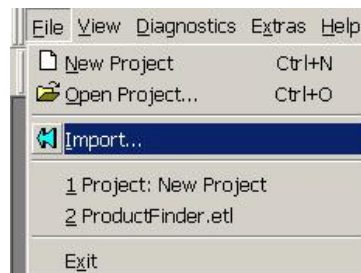
A continuación presentamos, de manera básica, el programa desarrollado por KNX para la configuración de sus instalaciones domóticas, el ETS3. Éste software se puede descargar en una versión demo de la página oficial y, si se quiere, comprar una licencia.

Presentaremos las opciones más básicas de la aplicación, como es cargar la base de datos de los dispositivos en ella, crear un proyecto, la inserción de dispositivos al proyecto y la creación de direcciones de grupo, es decir, las funcionalidades del sistema.

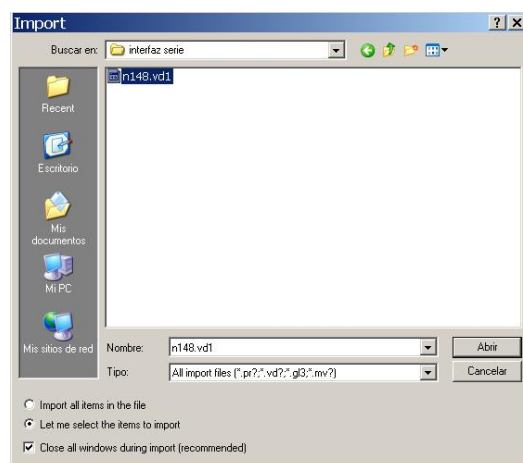
### Base de datos

El programa, necesita de una base de datos de productos, que se descargan de las páginas webs de los fabricantes de ellos. Cuando instalamos el ETS3, se crea una base de datos vacía, a la que, se agregan las bases de datos comentadas antes de la siguiente manera:

1. Seleccionar la opción *Import* del menú.

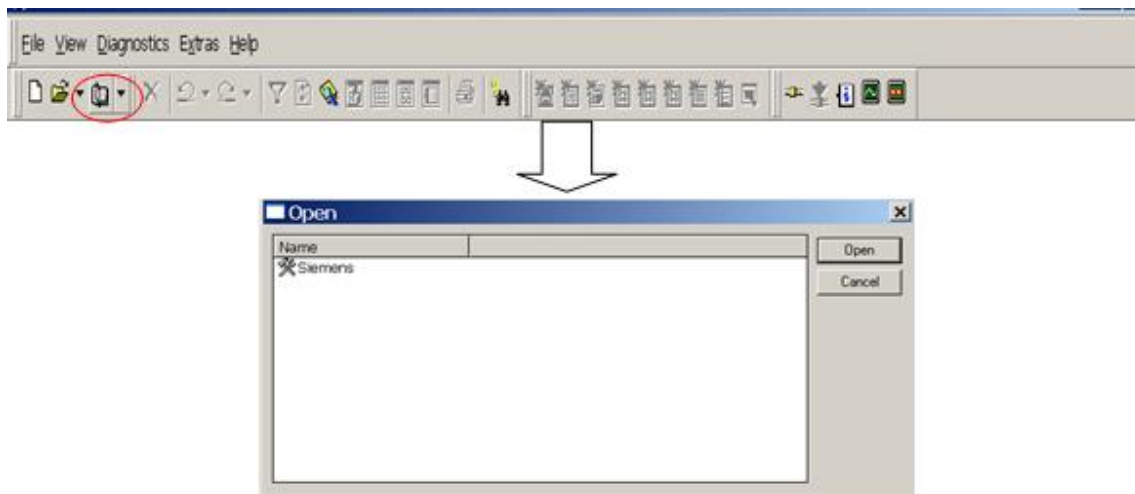


2. Seleccionar la base de datos (con extensión .vd1 o .vdx) en la unidad en la que esté guardada y pulsar OK.

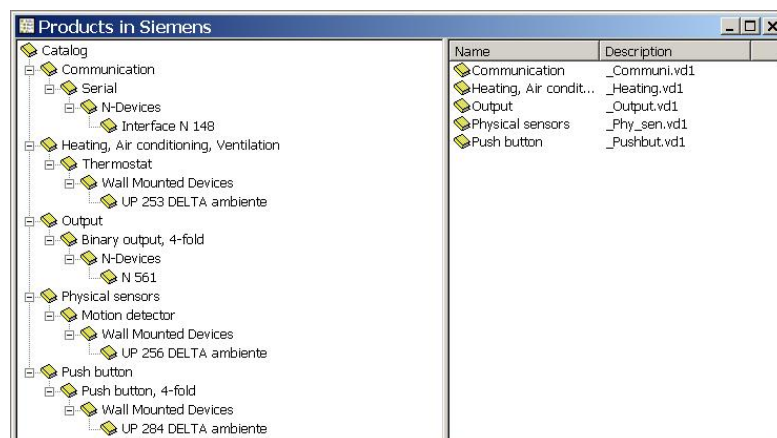


Después de seleccionar la base de datos a importar, hay que seleccionar uno a uno los dispositivos que queremos cargar, o darle a la opción de importar todo.


3. Una vez importada la base de datos, pulsar el botón cambiar en la barra de herramientas. Seleccionar en el diálogo, la base de datos importada y pulsar Open.

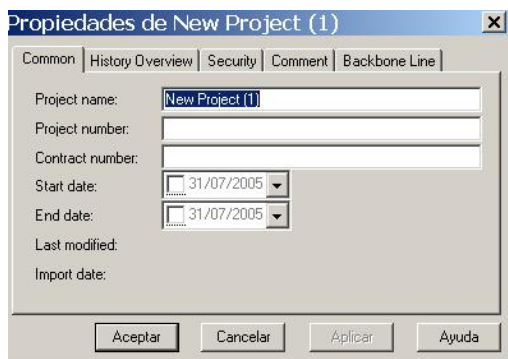


4. La base de datos queda instalada. Los productos se presentan en forma de árbol estructurado.

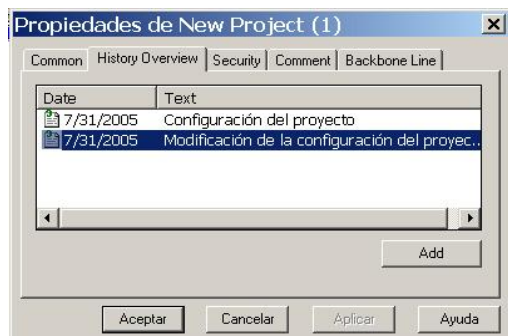


## Creación de un nuevo proyecto

Mediante la opción **File/New Project** o mediante el icono  de la barra de herramientas, se puede crear un nuevo proyecto. Cuando se crea un nuevo proyecto, aparece la ventana **New Project Properties**. Esa ventana consta de varias pestañas:



La pestaña Common: Permite introducir y cambiar el nombre del proyecto, así como otros datos del proyecto.

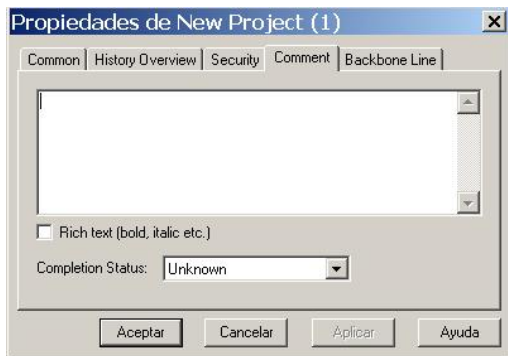


History Overview: Se puede añadir una nueva entrada pulsando el botón **Add**, también permite modificar las entradas existentes. Este historial de cambios se va añadiendo automáticamente los comentarios que se añaden al cerrar un proyecto.

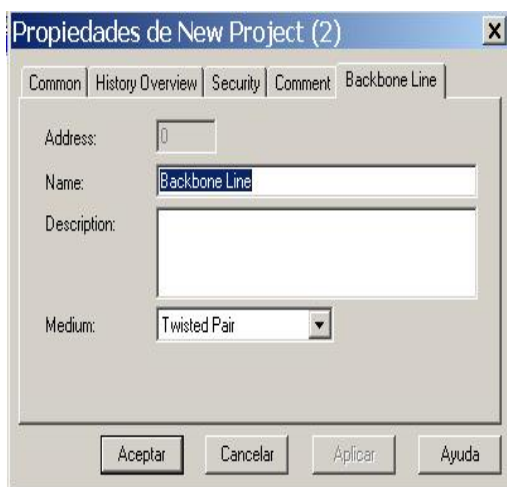


En la tercera pestaña, se puede introducir una clave para el proyecto o para la BCU el caso de dispositivos BCU2/BIM M112.

Introduciendo una clave para el proyecto se protege el proyecto contra accesos no autorizados, cada vez que se abre el proyecto se solicitará la clave. La clave puede ser cambiada en cualquier momento desde esta ventana.



En la pestaña **Comment** se pueden almacenar comentarios para el proyecto junto con una etiqueta describiendo el progreso o la finalización del proyecto.



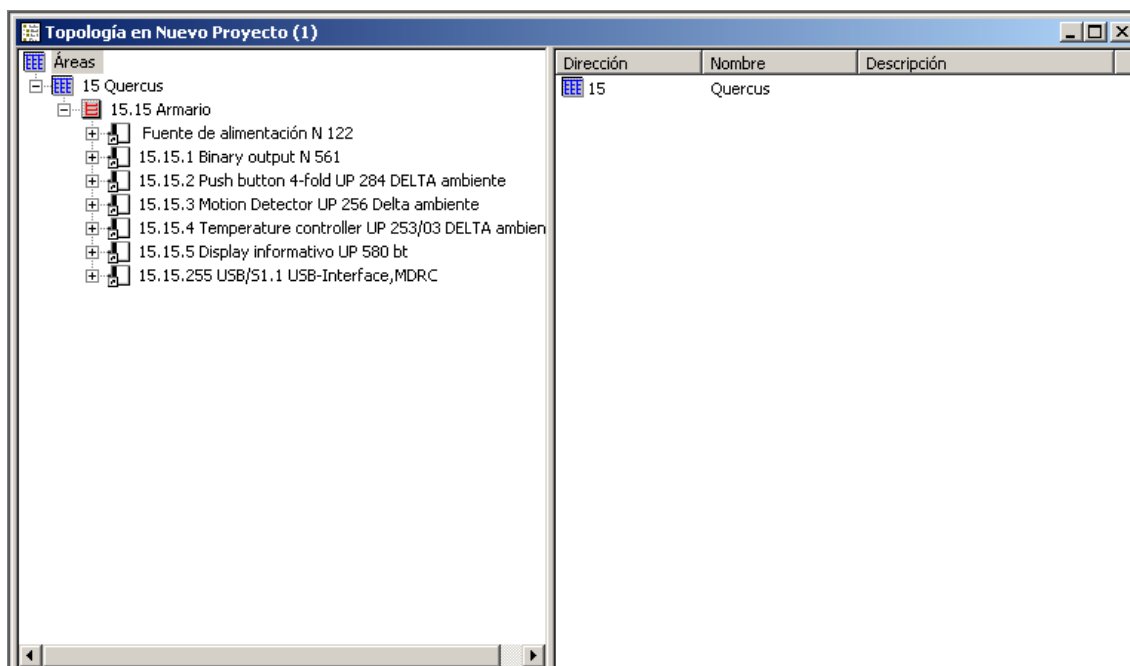
Esta pestaña contiene información sobre el backbone. El **backbone** es la línea del área con número 0. Interconecta las áreas de un sistema EIB. En esta pestaña se puede dar un nombre, comentarios adicionales y definir el medio de comunicación del backbone. Este diálogo solo se necesita cuando se tiene seleccionada la opción "Topology Display like in ETS2" localizada en **Extras/Options**, más concretamente en la página **Presentation** en la carpeta **Browser**. En las vistas del ETS3, este dato se puede introducir en el cuadro de propiedades del Backbone, como para cualquier otra línea normal.

### *Inserción de dispositivos al proyecto*

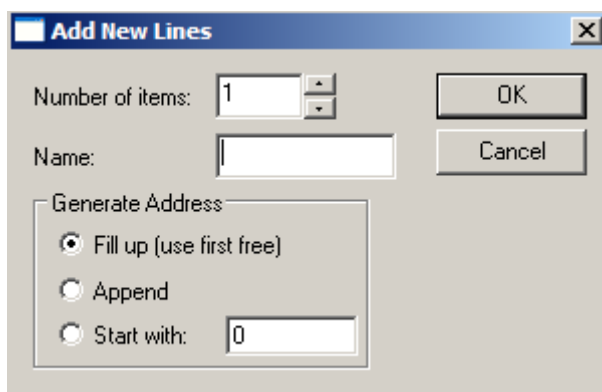
Para comenzar a insertar elementos al proyecto creado anteriormente, hay que hacer lo siguiente:

En la ventana de topología del proyecto, que se muestra a continuación, se van insertando primero una breve descripción del sistema, con áreas, líneas, etc. Posteriormente se insertan los dispositivos pertenecientes a la instalación.

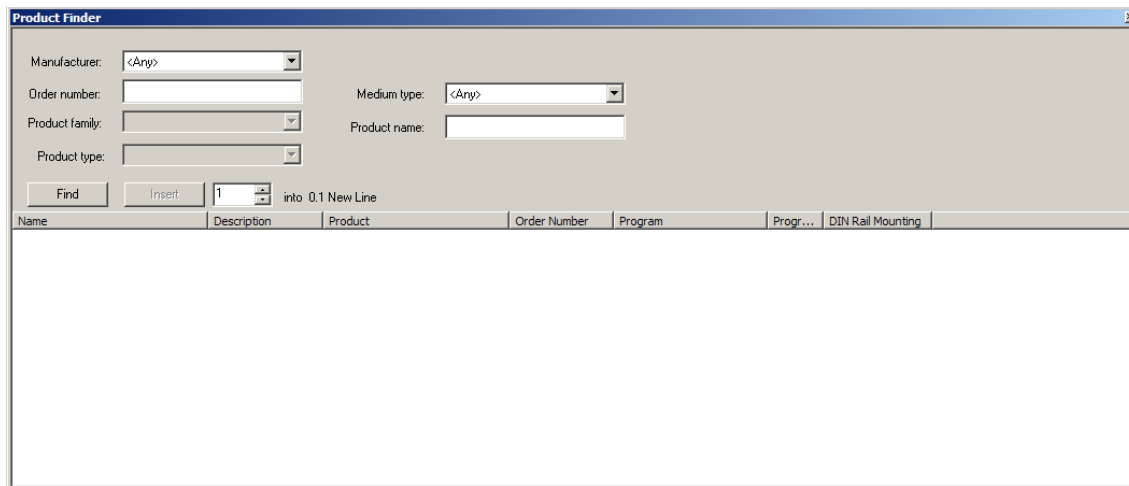




Para insertar, se hace click en **Area**, y se le da a **Add Line**, e introducimos un nombre para la línea que vamos a configurar.

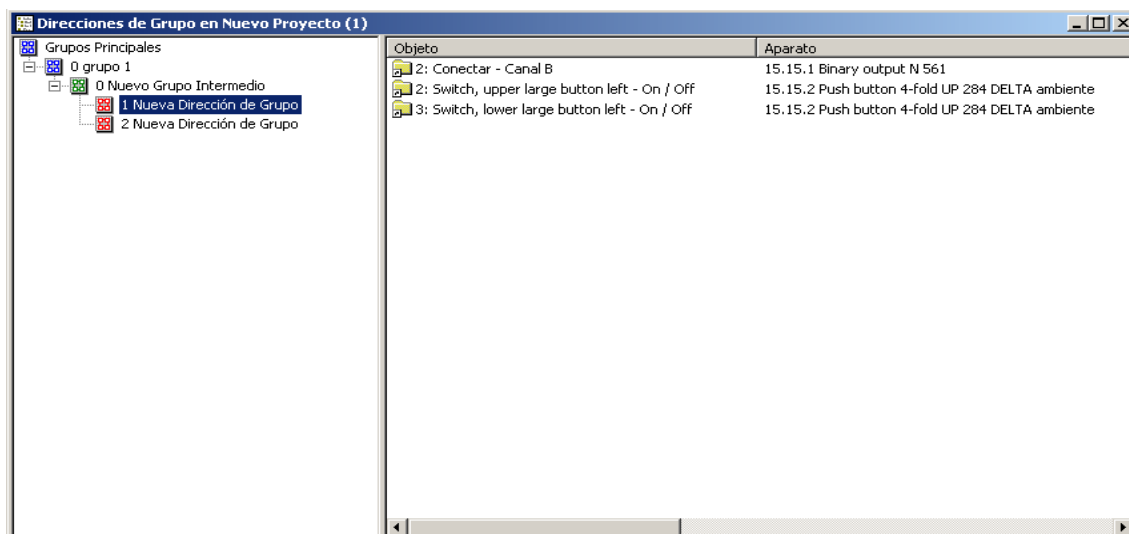


Al tener ya la línea en el proyecto, hacemos lo mismo que antes, pero esta vez presionamos añadir dispositivos, **Add device**, y se abre la ventana siguiente, en la que, se busca el dispositivo que queremos añadir a la instalación.



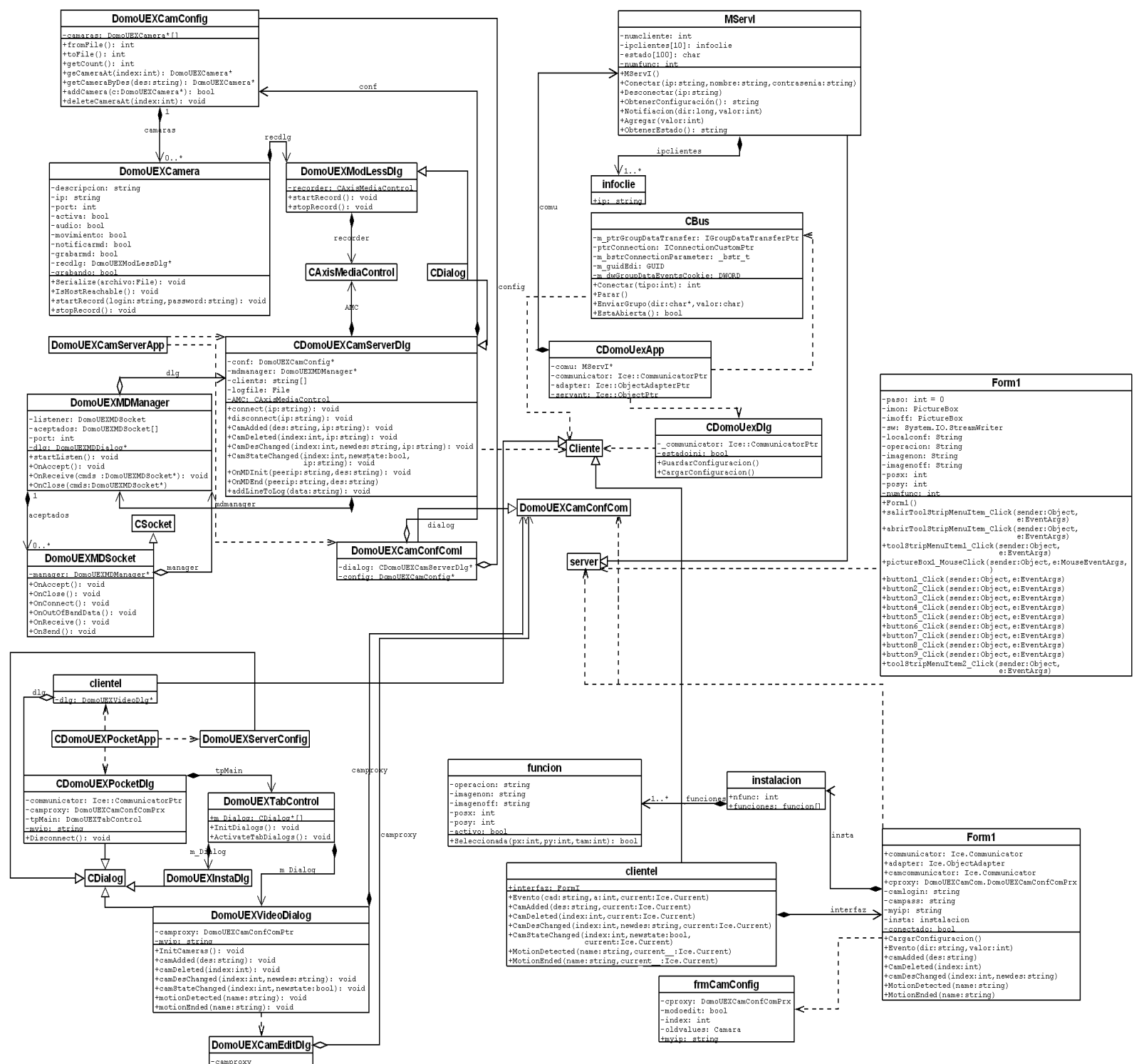
### Creación de direcciones de grupo

Los dispositivos que se insertan en la aplicación, según el método explicado anteriormente, pueden relacionarse entre si formando direcciones de grupo. A continuación se muestra una captura de la ventana donde se van creando las direcciones de grupo:



Lo primero que hay que hacer es, como antes, ir creando la estructura para crear la dirección de grupo, y para ello, a partir de Grupos Principales vamos creando la arquitectura de la dirección. Primero se crea el grupo y posteriormente un grupo intermedio, y, dentro del grupo intermedio creamos la dirección de grupo. Después de tener la dirección creada, únicamente hay que ir insertando los dispositivos instalados en la topología y ya está.

## ANEXO 2: DIAGRAMA DE CLASES AMPLIADO



### Ilustración 67. Diagrama de clases ampliado