Escuela Técnica Superior de Ingeniería de Telecomunicación

Grado en Ingeniería de Sistemas Audiovisuales y Multimedia

TRABAJO FIN DE GRADO

Desarrollo de una aplicación web para sistema domótico

Autor: Edgar Alejandro Barrero Mateo

Tutor: Jose María Cañas Plaza

Co-Tutor: José Centeno González

Curso académico 2013/2014

...si desastre soy, si caos, triste voy como acorde menor...
Teinfusion

Índice general

Índice de figuras

Agradecimientos

A dos personas que no están aquí. A ellas les debo todo.

Y por supuesto a mi padre, por su apoyo incondicional.

Resumen

La tecnología domótica está en auge. Un hogar domotizado ofrece numerosas ventajas y aumenta la calidad de vida de sus integrantes en términos de confort, eficiencia energética y seguridad. Por otro lado las aplicaciones web están tomando el relevo de las aplicaciones de escritorio convencionales gracias al gran avance en velocidad, capacidad de cómputo y nuevas funcionalidades de los navegadores.

El sistema Surveillance 5.1 desarrollado en este Trabajo de Fin de Grado une estas dos tecnologías ofreciendo una aplicación web atractiva e intuitiva orientada a controlar remotamente sensores y actuadores domóticos de un hogar. La aplicación web que se describe en esta memoria ofrece un flujo de vídeo desde una cámara web, un flujo de imagen de profundidad procedente de un sensor Kinect y su representación en 3D. También ofrece el acceso a un sensor de humedad y un actuador como ejemplos de dispositivos domóticos de bajo coste.

Surveillance 5.1 se ha desarrollado usando Ruby on Rails, un entorno de código abierto para el desarrollo de aplicaciones web. El servidor web se conecta a componentes JdeRobot que ofrecen interfaces ICE de objetos distribuidos. De esta forma obtiene datos de los distintos sensores y actuadores de la aplicación. En el lado cliente, el navegador refresca estos datos realizando peticiones AJAX.

El integrar las conexiones ICE con componentes externos dentro del propio código del servidor web deja la puerta abierta a nuevas aplicaciones web e interfaces de usuario destinadas a la robótica, domótica o visión artificial.

Capítulo 1

Introducción

En este capítulo se introducen los conceptos de domótica y aplicaciónes web. Además, se ofrece una visión general de las aplicaciones domóticas realizadas por el grupo de robótica de la URJC, que son el contexto inmediato de este TFG.

1.1. Domótica

Se llama domótica al conjunto de sistemas capaces de automatizar una vivienda, aportando servicios de gestión energética, seguridad y bienestar, y que pueden estar integrados por medio de redes interiores y exteriores de comunicación, cableadas o inalámbricas. Se podría definir como la integración de la tecnología en el diseño inteligente de un recinto cerrado.

Las ventajas y servicios de la dómotica se pueden resumir en:

- **Ahorro energético:** El gasto de energía puede disminuir mediante la regulación y programación de sistemas de calefacción o aire acondicionado. También se puede reducir el consumo mediante el uso de sensores, por ejemplo, bajando las persianas en las horas de más luz. Esta acción se traduciría en un ahorro en el consumo de aire acondicionado.

- **Confort:** Entendido como cualquier acción realizada que facilite y aumente la calidad de vida en el hogar. Un ejemplo puede ser la automatización de iluminación, hilo musical, integración del portero automático en móvil o televisor,etc.

- **Seguridad:** La domótica puede hacer una casa más segura. Existen en el mercado numerosos sistemas de alarma de presencia, de agua, gas, incendio, etc. Además, se

pueden programar sistemas para la simulación de presencia que enciendan luces y radio a horas aleatorias del día.

Las telecomunicaciones pueden facilitar la comunicación del usuario con el sistema de sensores y actuadores y éstos últimos con el exterior. Los ejemplos de estas prestaciones son innumerables: Visualización de alarmas y sensores en tiempo real desde ubicaciones remotas, teleasistencia, telemantenimiento, ubicuidad en el control tanto externo como interno de sistemas, mandos inalámbricos, etc. Es en este ámbito donde se centra este TFG.

La domótica puede ser una herramienta muy poderosa a la hora de conseguir el diseño universal [1] que consigue facilitar el día a día a todos los usuarios del sistema, incluidas personas con discapacidades.

La figura 1.1 representa una primera aproximación a la arquitectura del sistema domótico que se presenta en este TFG.

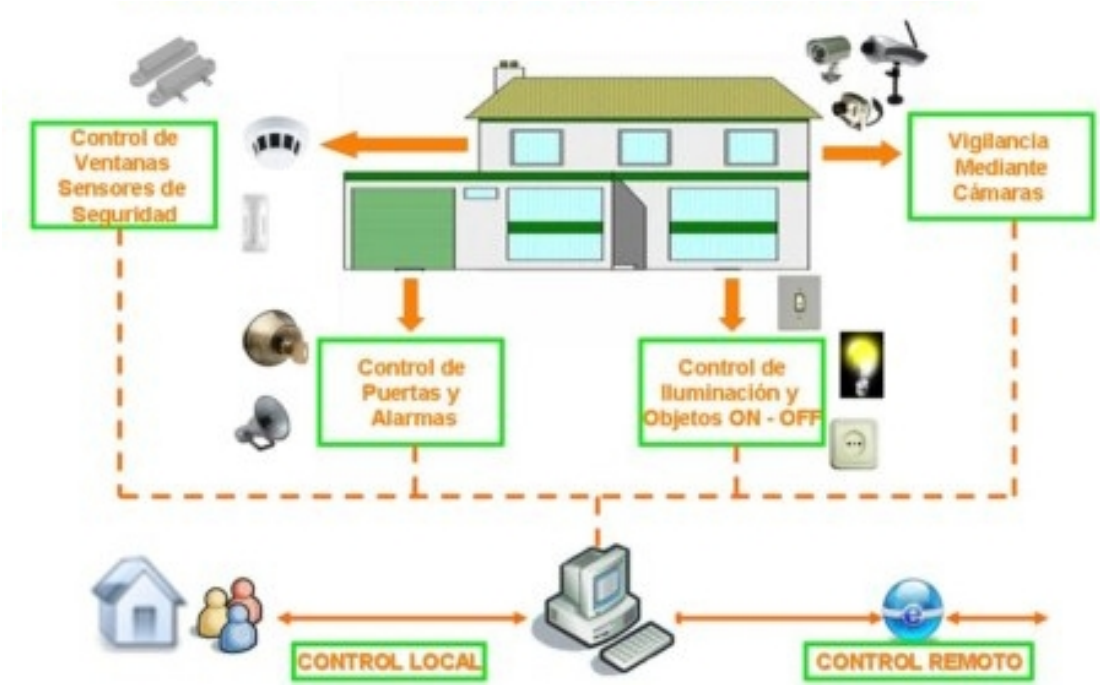

Figura 1.1: *Representación de un sistema domótico controlado mediante Internet*

El origen de la domótica se remota a la década de los setenta, cuando tras muchas investigaciones aparecieron los primeros dispositivos de automatización de edificios basados en la tecnología X10[29]. Posteriormente, surgieron otros estándares que permiten más flexibilidad en las configuraciones de los sistemas, como KNX[30], actual estándar europeo.

1.1.1. Estándares domóticos y conceptos

Los primeros sistemas comerciales fueron instalados, sobre todo, en Estados Unidos. Estos se limitaban a la regulación de la temperatura ambiente de los edificios de oficinas. Debido a la necesidad de interoperar distintos sistemas domóticos, por ejemplo ampliar una red ya existente con nuevos componentes, se comenzaron a definir estándares.

Si bien el estándar inicial X10 fue suficiente para soportar estas pequeñas aplicaciones, posteriormente los automatismos destinados a edificios de oficinas (junto con otros específicos) se han ido aplicando también a las viviendas de particulares u otro tipo de edificios, donde el número de necesidades a cubrir es mucho más amplio, dando lugar a nuevos estándares de comunicación.

- **X10** Protocolo estándar de comunicaciones para el control remoto de dispositivos eléctricos, desarrollado en 1975. Utiliza la red eléctrica para comunicar los distintos dispositivos del sistema a través de comandos predefinidos en el estándar.[29]

  Fue la primera tecnología domótica en aparecer, dando solución específica a las comunicaciones domóticas, diferentes de las comunicaciones arbitrarias entre dispositivos. Este estándar sigue en uso hoy en día, aunque ha sido redefinido para otorgarle más funcionalidades y existen protocolos alternativos más recientes, pero con menor número de dispositivos comerciales (debido principalmente al extenso tiempo que X10 lleva en el mercado).

- **KNX** Estándar europeo para la automatización de las viviendas y oficinas [30]. Para la transmisión de datos puede utilizar cableado de baja tensión (el más utilizado actualmente), de alta tensión (red eléctrica), transmisión inalámbrica o por Internet.

  Es un estándar domótico abierto que permite implementar un mayor número de comandos y sensores más heterogéneos que *X10*. Si bien *KNX* no es compatible con el estándar *X10*, existen numerosos pasarelas que permiten el uso de sistemas multiprotocolo.

- **SCP** Simple Control Protocol. Estándar de *Microsoft* y *General Electric* y actualmente libre de royalties.

  Necesita conexiones punto a punto y cableado de baja tensión, por lo que es prácticamente inviable para sistemas con un gran número de sensores. En general, este estándar no está teniendo gran acogida, además, es incompatible con la legislación Europea (en concreto, con la norma *EN-50065*[31]).

1.1.2. Mercado domótico

Recientemente ha crecido el número de empresas que ofrecen sistemas domóticos interoperables entre sí, permitiendo crear sistemas de forma flexible a un bajo precio. Normalmente, se necesitan conocimientos técnicos para poner a punto este tipo de redes. Con estos sistemas se pueden abarcar muchas soluciones: seguridad ante robos y averías domésticas, comodidad en el hogar y ahorro de energía. Generalmente, en estas soluciones comerciales, se pueden adquirir el número de nodos que se requieran.

Hay empresas que venden sensores aislados que no necesitan interconexión (por ejemplo sensores de vibración que emiten sonido) o que tienen una conexión limitada para instalar varios sensores del mismo tipo a una centralita. Esta centralita deslocalizada (principalmente interconectada por X10) suele ser la que emite la alarma o visualiza una cámara IP. Estos sistemas son generalmente baratos y fáciles de instalar y mantener, pero apenas ofrecen flexibilidad. En la figura 1.2 se muestra uno de estos dispositivos. En este caso, se trata de un sensor de movimiento y un avisador telefónico. Su precio es de 70€ aproximadamente.

Figura 1.2: *Sistema de alarma de presencia con avisador teléfonico*

Otras empresas ofrecen servicios de diseño e implementación de sistemas domóticos e inmóticos. Esto conlleva todas las ventajas y desventajas de sistemas creados a medida: no pueden ser reutilizados y suelen tener un precio alto.

A continuación, se ofrecen algunos ejemplos representativos del marco comercial de la domótica en España:

- **Z-Wave** Se autodefinen como estándar internacional[32]. Dispone de varios módulos de distintos fabricantes interoperables, con una gran variedad de sensores y actuadores. Permiten crear sistemas inalámbricos y gestión a través de teléfonos inteligentes.

Al ser sistemas de distintos fabricantes, son sistemas propietarios y los precios varían mucho. Por ejemplo, un controlador (nodo central) cuesta entre 60€ y 850€ (el más básico no dispone de conexión Ethernet, entre otras desventajas), un sensor cuesta entre 40€ y 60€ y una cámara, entre 70€ y 130€.

- **Verisure** Sistema de alarmas de la empresa *Securitas Direct*[33]. No se trata de un sistema domótico completo, pues únicamente dispone de sensores de alarma. Los nodos en este sistema se ofrecen formando un único juego y no se puede crear un juego de sensores a medida. Este juego contiene un nodo central, dos sensores de movimiento con cámara, un sensor de puerta abierta, un lector NFC y una sirena. La principal ventaja de este sistema consiste en el uso de sensores volumétricos adaptado a mascotas. Esto permite el uso de dichos sensores ante la presencia de animales sin provocar falsas alarmas.

  Por una cuota mensual de 99€ se dispone del juego de sensores y un de servicio de llamada a la policía controlado por la central de *Securitas direct*).

- **Prosegur** [34] Empresa similar a *Securitas Direct* y que ofrece servicios de vigilancia y domótica. En su página web indican que disponen de un sistema domótico completo (seguridad, confort y ahorro energético), aunque sólo se muestran módulos de vídeo-vigilancia (cámaras IP), detectores de movimiento, contactos magnéticos y centralita con etiqueta NCF. El modelo de negocio es parecido al de *Verisure*. La cuota mensual es de 30€ en adelante. El equipamiento incluido es también similar al de *Verisure*, salvo que incluyen otro sensor de movimiento en lugar del de puerta abierta.

- **Delta Dore** Ejemplo de sistema domótico actualmente comercializado, genérico, multipropósito e inalámbrico [35]. Se compone de distintos módulos según la necesidad del cliente. El sistema puede tener capacidad para gestionar la climatización y la luz de la casa, alarmas y apertura de puertas automáticas. El sistema se controla a través de un mando a distancia, desde un cuadro de mando o desde una aplicación para el móvil. Su precio es elevado. Por ejemplo, los nodos centrales cuestan a partir de 200€, sensores a partir de 65€, etc, aunque existen paquetes para reducir el precio final.

- **Libelium** Red de sensores de fabricación española que implementa *router* multiprotocolo, el cual puede interactuar con las tecnologías Wi-Fi, ZigBee, GPRS, Bluetooth y GPS en una misma máquina [36].

  Es un sistema principalmente pensado para desarrollo (utiliza Arduino y XBee como

base), aunque la propia empresa Libelium puede crear sistemas a medida a partir de sus placas y sensores.

Como un ejemplo internacional de mercado domótico, la firma Nest [10], actualmente propiedad de Google, comercializa sensores domóticos como termostatos, detectores de humo y CO2, etc. Los sensores están dotados con inteligencia que por ejemplo reduce el consumo energético del hogar o recuerda los niveles de humo aceptables para cada habitación. Estos sensores se pueden controlar mediante el ordenador o el teléfono móvil. La principal desventaja es su precio. Un termostato inteligente cuesta 249€. En la figura 1.3 se puede observar uno de estos termostatos.

Figura 1.3: *Termostato inteligente de la firma Nest*

1.2. Aplicación Web

Se denomina aplicación web a una aplicación informática que se conecta a un servidor web (parte servidor) y a la que el usuario accede mediante un navegador (parte cliente), usando para la comunicación entre servidor y navegador el protocolo HTTP. Las aplicaciones Web poco a poco han revolucionado la forma de utilizar Internet, aumentando el contenido de las páginas desde texto estático a un contenido multimedia cada vez mas rico e interactivo y por lo tanto escalable.

El modelo que utilizan las aplicaciones web es típicamente el modelo cliente servidor. Este modelo se representa en la figura 1.4.

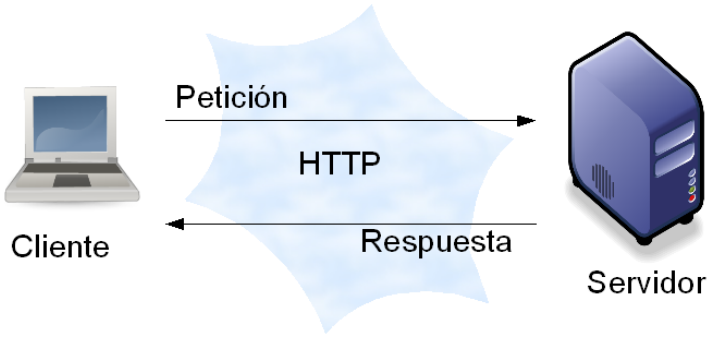

Figura 1.4: *Arquitectura cliente-servidor usando el protocolo HTTP*

El concepto de la aplicación web no es nuevo. Uno de los primeros lenguaje de programación para el desarrollo de aplicaciones web es Perl, inventado por Larry Wall en 1987 antes de que Internet se convirtiera en accesible para el público en general. Desde entonces hasta hoy en día, los navegadores han ido ganando en rapidez, capacidad de cómputo y funcionalidades. Tecnologías como HTML5, AJAX o WebGL hacen que en la actualidad las aplicaciones web tengan poco que envidiar a las aplicaciones de escritorio.

En la figura 1.5 se observa un gráfico sobre la evolución de la eficiencia de JavaScrpit. Este lenguaje es un estándar de facto en la programación de la parte cliente de las aplicaciones web. Se puede observar que su rendimiento crece exponencialmente con los años. Esta evolución explica el porqué de la apuesta por las aplicaciones web en el mundo de la informática.

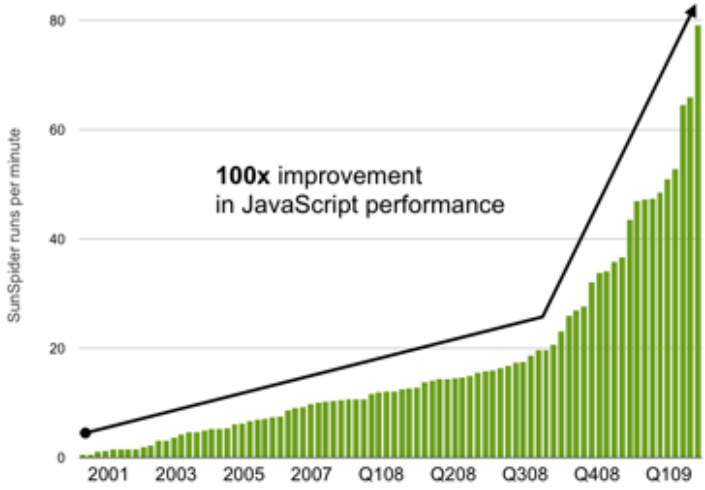

Figura 1.5: *Incremento del rendimiento de Javascript*

El uso de aplicaciones web además ofrece las siguientes ventajas:

- No se necesita tener instalado ningún software específico para que la aplicación funcione. Desde cualquier máquina que tenga un navegador web instalado podremos acceder a la aplicación. De esta forma, estamos ahorrando espacio de almacenamiento en el disco duro local.

- Es independiente de plataforma. Las aplicaciones web se pueden usar en cualquier sistema operativo (Linux, iOS, MacOS, Windows, etc). Ésto hace que sólo necesitemos desarrollar una vez la aplicación y ésta funcionará en cualquier arquitectura. Con esta tecnología se ahorra tiempo de programación. Además, abre las puertas a usar la aplicación en teléfonos móviles o *tablets* sin tener que dedicar ningún esfuerzo extra.

- Cualquier actualización que se haga en la aplicación es inmediata en todos los usuarios ya que estos no tienen descargado ningún software en local. Además, se asegura que todos los usuarios están usando la última versión de nuestra aplicación con lo que no existen problemas de compatibilidades entre versiones.

- La tecnología web guarda casi toda la información en bases de datos en el servidor. Si existe cualquier problema en el cliente (ataque de virus informático, borrado accidental del disco duro) no se perderá información.

- Las aplicaciones web actuales ofrecen unas prestaciones similares a las aplicaciones de escritorio en cuanto a capacidad y rapidez.

Las aplicaciones web también tienen desventajas:

- Se depende de terceros para el uso de la aplicación ya que se necesita de tener conexión a Internet.

- Aunque las aplicaciones web han avanzado muchísimo en los últimos años, de momento, ofrecen menos funcionalidades que las aplicaciones de escritorio.

En las comunicaciones entre servidor y cliente se usa el protocolo de nivel de aplicación HTTP (*Hipertext Transfer Protocol*). Este protocolo fue desarrollado por el World Wide Web Consortium y la Internet Engineering Task Force. En 1999 se publicó la versión 1.1 usada en la actualidad. HTTP define la sintaxis y la semántica que utilizan los elementos de software de la arquitectura web (clientes, servidores, *proxies*) para comunicarse.

Es un protocolo sin estado, es decir, que no guarda ninguna información sobre conexiones anteriores. El desarrollo de aplicaciones web necesita frecuentemente mantener

estado. Para ello, se usan las *cookies* que no son más que atributos clave-valor que viajan en cada mensaje HTTP.

Para la creación de aplicaciones web existen entornos de desarrollo que facilitan la labor del programador. Los más importantes son Django y Ruby on Rails. Es este último entorno el elegido para el desarrollo de este TFG.

La aplicación web, una vez programada, debe alojarse en un servidor web. Ejemplos de servidores usados en la actualidad son Apache, Microsoft IIS, Ngnix o Node.js.

Existen multitud de aplicaciones web exitosas en el mercado. Las webs de Amazon o Ebay son buenos ejemplos de aplicación web. Si buscamos aplicaciones web orientadas a videovigilancia, una de las mas usadas y similares a la de este TFG, es la página de la Dirección General de Tráfico mostrada en la figura 1.6.

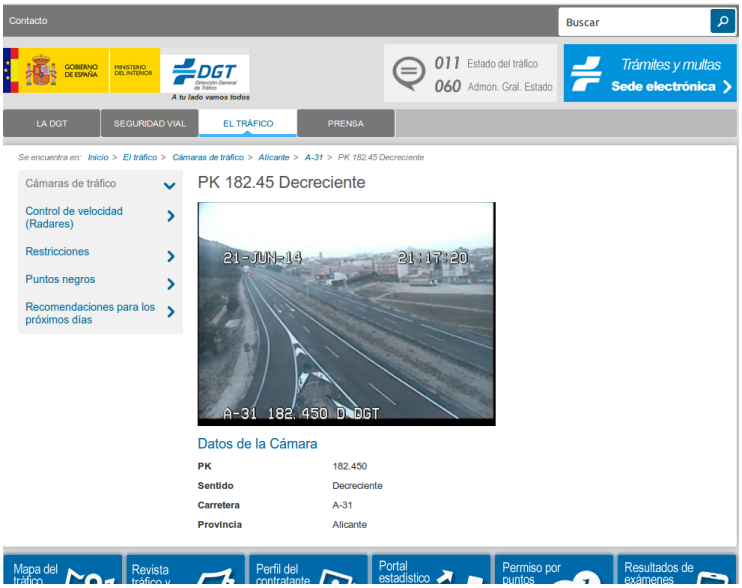

Figura 1.6: *Aplicación web de videovigilancia de tráfico de la DGT*

Las aplicaciones web a menudo se denominan web de contenido dinámico. El motivo es que la página que se muestra al usuario no es siempre la misma (estática), sino que depende de cómo el usuario interactúe con la aplicación web. El contenido de la página lo crea el servidor web al vuelo antes de servirlo al usuario. Con esta tecnología es posible realizar búsquedas de productos, añadir un producto al carrito de la compra, etc.

El futuro de las aplicaciones web apunta hacia WebRTC (Web Real-Time Communication) que es una API que está siendo elaborada por la World Wide Web Consortium (W3C) para permitir conexiones multimedia y el uso compartido de archivos P2P directamente entre navegadores y sin necesidad de *plugins*.

1.3. Domótica en la *URJC*

En este apartado se repasa varios aportes de la URJC al mundo de la domótica. Estos aportes, sobre todo el proyecto Surveillance, son el contexto cercano de este TFG.

Proyecto Surviellance

Surveillance 1.0[37] es el producto del Trabajo Fin de Máster de *Roberto Calvo Palomino*. Consiste en un sistema de vídeo vigilancia distribuido integrado en dispositivos móviles basados en la plataforma Android.

El sistema realiza grabaciones a través de las cámaras instaladas y lleva asociada una generación de alarmas mediante detección de movimiento. Todo ello es gestionado y visualizado desde un dispositivo móvil inteligente basado en Android. La conectividad entre las cámaras y el dispositivo se realiza de forma alámbrica. En la figura 1.7 se puede observar una captura de pantalla del proyecto Surveillance 1.0.

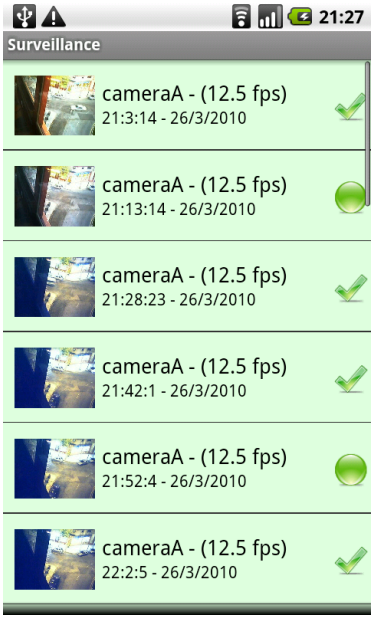

Figura 1.7: *Captura de pantalla de Surveillance 1.0*

Surveillance 2.0 se constituye como una exploración aparte, realizada por *Roberto Calvo*, para visualizar el valor de diversos sensores a través de un teléfono inteligente con Android. Para ello, se implementó una aplicación compuesta por diversos sensores conectados a una placa Arduino y cuyos datos finales eran visualizados en un dispositivo móvil. Para lograr las comunicaciones, tanto de los sensores con la placa, como de la placa con el dispositivo móvil, se usaron cables.

Se integraron sensores de infrarrojos (movimiento), temperatura y puerta abierta (sensor magnético). Para ello, los sensores se unieron a una placa Arduino mediante un cable USB, la placa procesa los datos y los envía a través de su interfaz Ethernet para poder visualizarlo en un tercer dispositivo. La comunicación entre el Arduino y el dispositivo final (la muestra se realizó con un dispositivo Android) se basa en los protocolos HTTP y JSON.

Surveillance 4.0 [8] [9] fue desarrollado por *Daniel Castellanos* como su Proyecto de Final de Carrera. Esta aplicación contaba con varios sensores de distinto tipo (humedad, temperatura, gas, etc) que se conectaban inalámbricamente con un nodo central situado en una Raspberry Pi. La conexión inalámbrica se hacía mediante transmisores Zigbee [11] con un protocolo propio llamado WHAP. El nodo central recibía los datos de los sensores y los mostraba mediante un servidor web que corría en la misma máquina. La apliación web se desarrolló en Python usando el entorno de desarollo Django. En Surveillance 4.0, los valores de los sensores se guardaban en una base de datos que la aplicación web consultaba cuando era necesario. Además, esta versión incluía un *streaming* de vídeo pero utilizando el software de código abierto M-JPEG Streamer.

Sistema de riego

Trabajo fin de máster de *Rubén Orellana Galloso*[38], realizado en el *Departamento de tecnología electrónica* de la *URJC*. El proyecto plantea la modernización de un sistema de riego de accionamiento manual a otro automatizado que haga un uso óptimo del agua. Se basa en la creación de un sistema de riego automático utilizando las tecnologías Arduino, ZigBee, Android, Bluetooth y otros elementos de bajo coste para monitorizar el estado de la tierra y mejorar el consumo del agua en un emplazamiento gaditano a partir de un sistema de regadío ya existente.

Proyecto ElderCare

ElderCare [39] es un sistema autónomo y no intrusivo, diseñado para detectar caídas mediante el uso de cámaras y *Kinect*s (o *Xtion*s). Su principal mercado es el cuidado de personas mayores, por lo que su uso principal se encuentra en residencias de ancianos o domicilios particulares donde vivan éstos. El sistema permite reducir drásticamente el tiempo de respuesta a la asistencia en caso de caída, estimando la posición del usuario caído y avisando a sus cuidadores o a la asistencia médica.

El Trabajo de Fin de Grado presentado en esta memoria desarrolla una aplicación web orientada a la dómotica donde se puede ver el flujo de vídeo de una cámara web y el flujo de vídeo de un sensor Kinect. Además, se obtienen datos de un sensor de humedad y se puede activar o desactivar un actuador que enciende y apaga una luz.

En el siguiente capítulo [2] se presentan los objetivos de este TFG. En el capítulo de Infraestructura necesaria [3] se hace un resumen de las tecnologías usadas en esta aplicación. En el capítulo de Desarrollo [4] se profundiza en el diseño y programación de la aplicación. Las pruebas y los materiales necesarios para ellas se detallan en el capítulo de Pruebas [5]. Por último, el capítulo de Conclusiones [6] hace un resumen de los objetivos logrados y presenta futuras líneas de trabajo.

Capítulo 2

Objetivos

El objetivo principal de este Trabajo de Fin de Grado es crear una aplicación web para el control de distintos sensores y actuadores dómoticos. La arquitectura del sistema debe permitir que los sensores y actuadores se conecten mediante una red de datos con el servidor web. Este objetivo general se ha articulado en tres subobjetivos que abordan cada uno una parte de la funcionalidad de la aplicación web.

El primer subobjetivo de esta versión es que el flujo de vídeo se realice directamente desde el servidor ICE sin usar ningún software adicional. Para conseguir esta funcionalidad, las imágenes se deben recibir mediante una conexión con un *middleware* de comunicaciones.

El segundo objetivo es incluir un flujo de imágenes de profundidad. Estas imágenes se recibirán de un sensor profundidad tipo Kinect. Dentro del subobjetivo anterior, se debe crear una imagen 3D con la nube de puntos obtenida.

El tercer subobjetivo es obtener información de sensores de bajo ancho de banda. También se deben incluir actuadores en la aplicación. Se debe integrar dentro de la aplicación las peticiones de información a los sensores. Tanto los sensores como los actuadores se deben manejar directamente con los GPIO de una placa RaspberryPI.

Además de los objetivos, la solución diseñada debe cumplir algunas características adicionales:

El primer requisito es que la aplicación web debe ser escalable. Para ello debe ser compatible con cualquier sensor o actuador que implemente una interfaz ICE. Esta orientación viene marcada por la trayectoria del proyecto libre JdeRobot. Este proyecto tiene desarrollados multitud de componentes en el ámbito de la robótica y la domótica. Cada componente tiene los drivers necesarios para cada tipo de dispositivo y ofrece una interfaz de tipo ICE para dicho dispositivo.

Como segundo requisito de la aplicación, los flujos de vídeo tanto de imagen convencional como imagen de profundidad deben ser lo suficientemente fluidos como para crear la sensación de movimiento sin que haya cortes ni interrupciones de ningún tipo.

El tercer requisito es que la página web tenga un diseño sencillo y atractivo. Además, este diseño debe ser accesible para cualquier persona que no tenga conocimientos técnicos.

2.1. Métodología y plan de trabajo

Para el desarrollo de este TFG se ha seguido el método de desarrollo en espiral. Este sistema se basa en iteraciones sucesivas en la que cada bucle o iteración representa un conjunto de actividades. Estas actividades incluyen tal y como muestra la figura 2.1: análisis de los requerimientos del sistema, diseño del mismo, implementación y pruebas.

Figura 2.1: *Esquema del desarrollo en espiral seguido en este TFG*

Este modelo se basa en plantear en primer lugar tareas sencillas que impliquen poca complejidad y por medio de iteraciones llegar a la solución final que recoja todos los objetivos fijados inicialmente.

Desde el comienzo de este TFG se planteó una serie de reuniones semanales entre tutores y alumno donde se iban analizando cada una de estas cuatro etapas del desarrollo en espiral.

Al comienzo del proyecto se diseñó un plan de trabajo y se dividió en apartados. La figura 2.2 refleja el desarrollo cronológico de estos apartados:

- Aprendizaje de Ruby y Rails.
- Aprendizaje de la plataforma JdeRobot.
- Desarrollo de aplicación de prueba.
- Desarrollo del módulo de *streaming* de vídeo.
- Desarrollo del módulo de *streaming* de imágenes de profundidad.
- Aprendizaje de WebGL y representación gráfica de puntos.
- Desarrollo del componente `GPIOServer`.
- Desarrollo del módulo para sensores y actuadores.
- Diseño de apariencia de la aplicación final.
- Integración de todos los módulos en la aplicación final

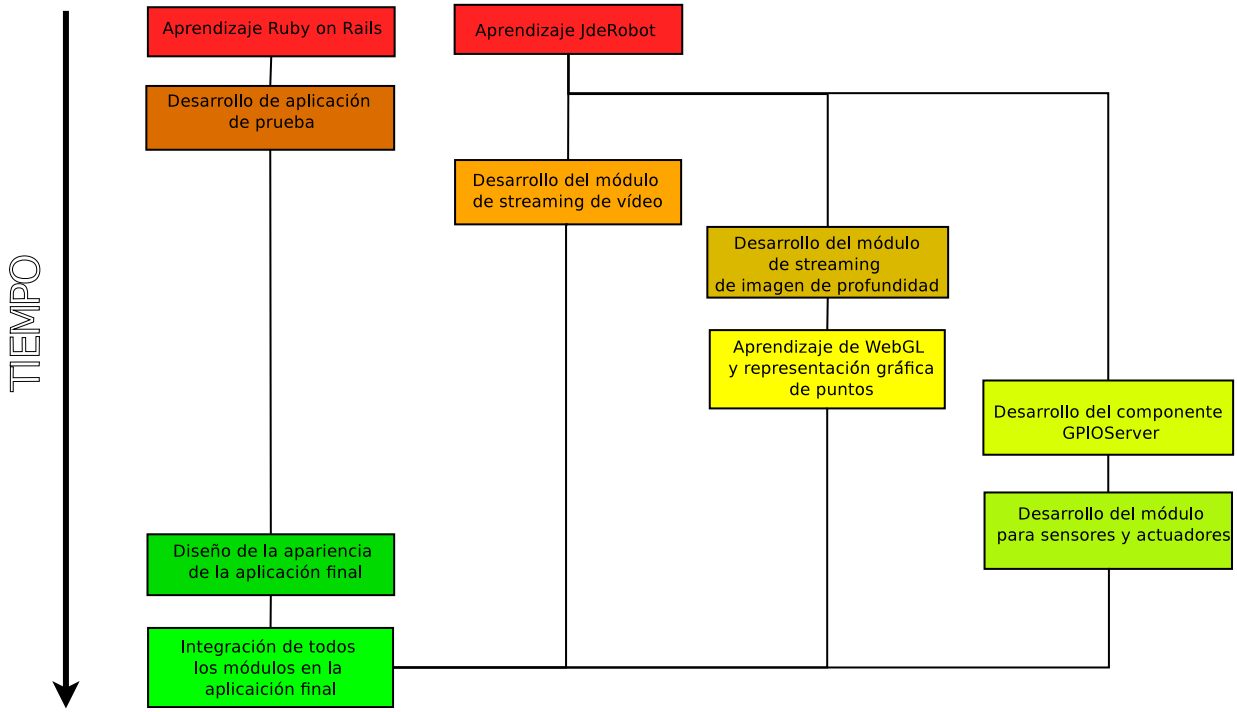

Figura 2.2: *Diagrama del desarrollo cronológico de Surveillance 5.1*

Se comenzó con una aplicación web de prueba a la que progresivamente se iban añadiendo módulos.

El desarrollo en espiral se aplicó individualmente para cada tarea. En primer lugar, se definía el objetivo de cada tarea y se planteaba un primer diseño. Cuando se conseguía que ese diseño funcionara, se iba optimizando para conseguir mayores prestaciones o nuevas funcionalidades.

Una vez que los módulos estuvieron lo suficientemente maduros, se procedió a la creación final de la aplicación web donde se integraron todos ellos, dando esta vez importancia a aspectos como la apariencia y la simplicidad en el manejo de la aplicación.

Este TFG se ha ido documentando en una *mediawiki* [12]. Este cuaderno de bitácora ha servido de apoyo a las reuniones semanales con los tutores. Como complemento a esta *mediawiki* se ha utilizado un repositorio público svn [13] donde se ha guardado y está accesible todo el código fuente de este TFG.

Capítulo 3

Infraestructura necesaria

En este capítulo se especifica qué *software* se ha utilizado en este TFG. Para cada software se indica la versión y para qué se ha usado en concreto en la aplicación web.

3.1. Ruby

Ruby [15] es un lenguaje de programación de código abierto, interpretado y orientado a objetos. Tiene una sintaxis inspirada en Python y Perl. Fue creado en 1995 por el programador japonés Yukihiro "Matz"Matsumoto. Se caracteriza por una sintaxis elegante y un código fácil de escribir y de leer. Existen multitud de bibliotecas para Ruby llamadas Gemas.

La parte del servidor web de la aplicación se ha programado en Ruby 2.0.

3.2. Ruby on Rails

Ruby on Rails (RoR) [16] es un entorno de desarrollo de código abierto para la creación de aplicaciones web. Usa Ruby como lenguaje de programación. Rails se distribuye a través de RubyGems, que es el formato oficial de paquete y canal de distribución de bibliotecas y aplicaciones Ruby.

Ruby on Rails usa el paradigma modelo-vista-controlador para la creación de aplicaciones. El paradigma modelo-vista-controlador permite separar el código de la aplicación web en tres partes:

- **El Modelo:** Está formado por clases. Cada clase es un modelo, y cada modelo representa una tabla en la base de datos. Por tanto, el modelo es el encargado de trabajar con la lógica de la base de datos.
- **La Vista:** Es la representación final de una petición. Está formada por código HTML. También puede contener algo de código Ruby, pero será mínimo ya que la vista se encarga de la presentación y no de la lógica de la petición.
- **El Controlador:** Se encargará de la lógica de la petición. Cada controlador es una clase con su propios métodos. El controlador hace de puente entre la vista y los modelos. Cuando se recibe una petición, el controlador consulta la base de datos del modelo y con estos datos construye la vista final que se envía al cliente.

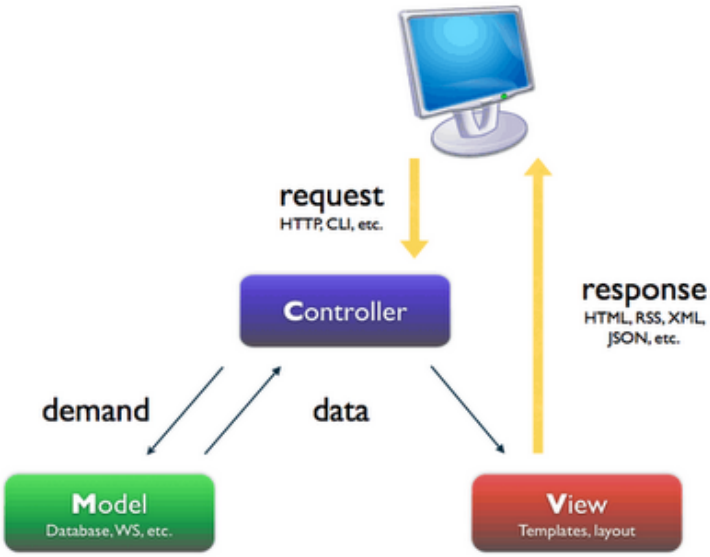

Figura 3.1: *Ruby on Rails usa el paradigma modelo-vista-controlador*

Ruby on Rails se basa además en dos principios para hacer que el código sea mínimo y lo más legible posible:

DRY (*Dont Repeat Yourself)*: Un principio por el cual cada definición debe hacerse sólo una vez. Dado que Ruby on Rails es un entorno de desarrollo de pila completa, los componentes están integrados de manera que no hace falta establecer puentes entre ellos. Esto hace que las definiciones hechas en un componente sean visibles en el resto.

Convención sobre configuración: El programador sólo necesita definir aquella configuración que no es convencional. Esto se debe a que Rails tiene convenciones para casi todos los aspectos que necesite el programador de la aplicación.

Toda la aplicación web de Surveillance 5.1 se ha programado dentro del entorno Rails. Se usó la versión 4.0 del mismo.

3.3. Ice

ICE [17] Se puede entender como un *middleware* de conectividad que hace posible que aplicaciones distribuidas puedan ejecutarse sobre distintas plataformas heterogéneas. Estas plataformas pueden tener distintos sistemas operativos, usar distintos protocolos de red, e incluso, estar escritos en distintos lenguajes de programación.

ICE (*Internet Communication Engine*) es un *middleware* orientado a objetos distribuidos, es decir, ICE proporciona herramientas, APIs, y soporte de bibliotecas para construir aplicaciones cliente-servidor orientadas a objetos distribuidos.

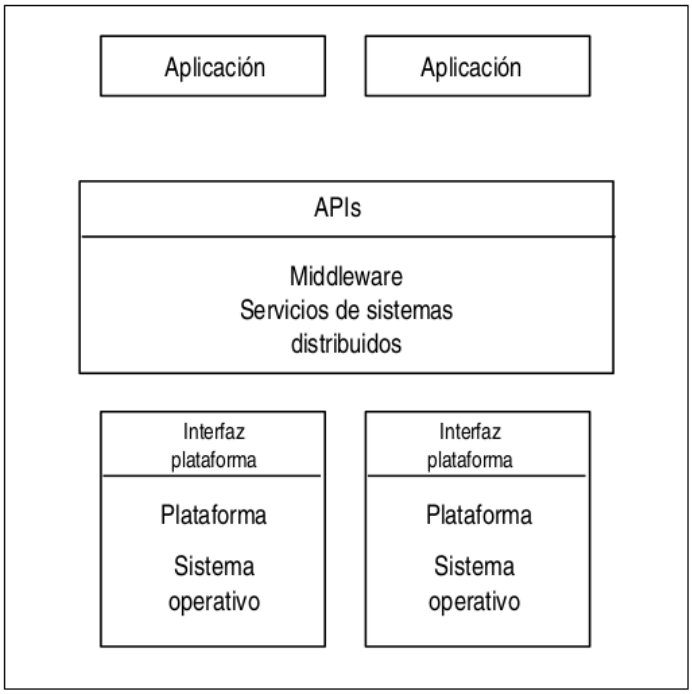

Figura 3.2: *Esquema general de un middleware*

Los principales objetivos de diseño de ICE son los siguientes:

- Proporcionar un *middleware* listo para usarse en sistemas heterogéneos.
- Proveer un conjunto completo de características que soporten el desarrollo de aplicaciones distribuidas reales en un amplio rango de dominios.

- Evitar una complejidad innecesaria, haciendo que ICE sea fácil de aprender y de usar.
- Proporcionar una implementación eficiente en ancho de banda, en uso de memoria, y en carga de CPU.
- Proporcionar una implementación basada en la seguridad, de forma que se pueda usar sobre redes no seguras.

  Cada objeto en ICE implementa una interfaz con un número de métodos. Estos métodos, junto con los tipos de datos que se intercambian entre cliente y servidor se definen usando el lenguaje de *Slice*. Este lenguaje es un lenguaje neutro, es decir, independiente al lenguaje en que se programe el cliente o servidor. Estos *Slice* en los que se escriben las interfaces se deben traducir después al lenguaje específico que use servidor y cliente. Servidor y cliente implementarán dicha interfaz, en el lenguaje en que cada uno esté escrito.

  Una vez que se establece la conexión ICE entre cliente y servidor, el cliente puede invocar métodos del objeto ICE que estén definidos en la interfaz. Esos métodos están implementados en el código del servidor. Por defecto, la llamada a estos métodos es síncrona, es decir, el flujo de ejecución se detiene en la llamada al procedimiento remoto y no continúa hasta haberlo ejecutado y obtenido el valor que devuelve.

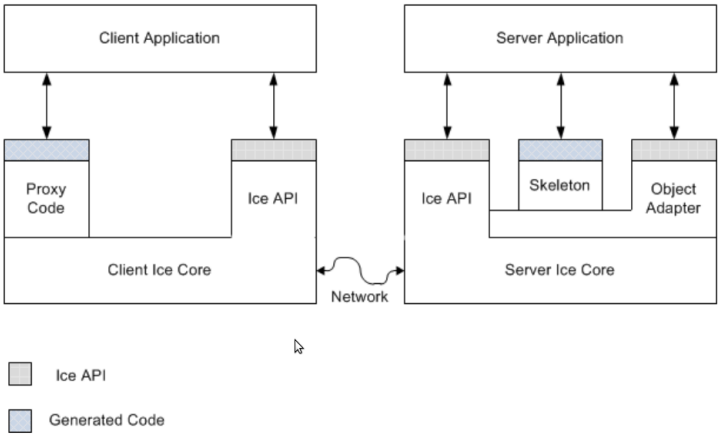

Figura 3.3: *Estructura de un cliente y un sevidor ICE*

Para la instalación de la plataforma JdeRobot se requiere la versión de ICE 3.4.1. Esta versión está disponible como paquete Debian. Sin embargo, para poder usar Ruby 2.0 en el desarrollo de la aplicación, fue necesario compilar el código fuente de ICE 3.4.2 for Ruby.

Todas las conexiones realizadas entre el servidor web y los componentes de JdeRobot en Surveillance 5.1 se realizan mediante ICE.

3.4. JdeRobot

JdeRobot [2] es una plataforma de desarrollo de aplicaciones orientadas a la robótica, domótica y visión artificial. La versión actual de JdeRobot es la 5.2. Esta plataforma está diseñada para facilitar la programación de software que dote de cierta inteligencia a hardwares tan distintos como cámaras, actuadores y en general robots de todo tipo. Casi todas estas aplicaciones están escritas en lenguaje C++ y basadas en un entorno de componentes distribuidos. Cada aplicación está construida con la concurrencia de varios componentes asíncronos. Cada componente puede ejecutarse en una computadora distinta y para su interconexión usan el *middelware* de comunicaciones ICE.

JdeRobot simplifica el acceso a los dispositivos hardware desde el programa principal. Así, obtener una medida de un sensor u ordenar un movimiento a un motor es tan simple como llamar a una función local.

Los componentes desarrollados por JdeRobot pueden conectarse a sensores y actuadores reales o pueden conectarse a un simulador. Estas conexiones pueden ser locales (dentro de la misma máquina) o usando Internet.

3.4.1. Cameraserver

Cameraserver [19] es un componente incluido en Jderobot que ofrece *streaming* de vídeo de una o varias cámaras. Usa GStreamer [21] internamente para manejar y procesar las fuentes de vídeo. Como otros componentes de Jderobot está escrito en C++ y ofrece una interfaz ICE con la que comunicarse. La figura 3.4 ofrece un esquema del funcionamiento de `cameraserver` junto a otro componente de Jderobot que muestra el *streaming* de vídeo.

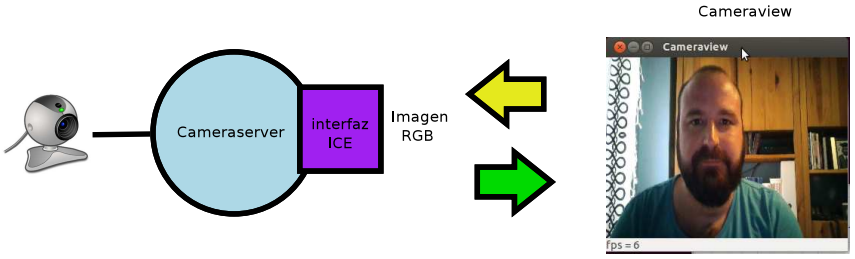

Figura 3.4: *Ejemplo del uso del componente cameraserver y el visor cameraview*

Cualquier programa que quiera conectarse con el componente `cameraserver` debe importar la interfaz `camera.ice`. Esta interfaz a su vez importa otra llamada `image.ice`. Esta última contiene un método llamado `getImage()`. Al invocar este método se devuelve una estructura formada por una imagen en RGB8 y las características de la misma.

Para la ejecución de este componente, es necesario proporcionar la ruta de su archivo de configuración. En este fichero de configuración se define la IP y puerto donde escucha el servidor, el número y nombre de las cámaras que tiene y la configuración de las mismas.

3.4.2. Openni1Server

`Openni1Server`[20] es otro componente incluido en Jderobot. Está escrito en C++ y para su funcionamiento usa las librerías Opencv[22] y Openni[23] entre otras. Este componente tiene todos los drivers necesarios para acceder al sensor Kinect y ofrece tres tipos de datos: una imagen RGB, una imagen de profundiad y una nube de puntos. Al ofrecer tres tipos distintos de datos tiene tres interfaces ICE. En este TFG sólo se va a hacer uso de la interfaz de imágenes de profundidad. Para que un programa haga uso de esta interfaz se debe importar las mismas interfaces ICE que para usar el componente `cameraserver`. Al tratarse de *streaming* de imágenes en ambos componentes, aunque de distinto tipo, se ha unificado la interfaz usada. De nuevo la interfaz contiene un método `getImage()` que devuelve la imagen de profundidad. De esta forma el componente para visualizar ambos flujos de vídeo puede ser el mismo. En la figura 5.3 se refleja éste funcionamiento.

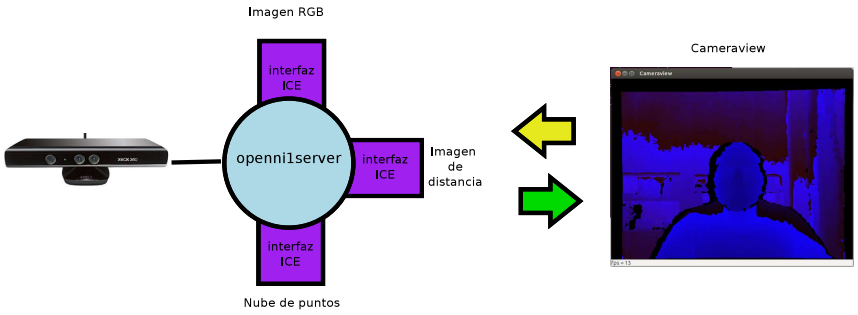

Figura 3.5: *Ejemplo del uso del componente openni1server y el visor cameraview*

Para la ejecución de este componente, es necesario proporcionar la ruta de su archivo de configuración. De nuevo, este fichero es similar al explicado para el componente `cameraserver`, pero en este caso, se añade información sobre la configuración del sensor de profundiad.

3.5. WebGL

WebGL [3] es un API para JavaScript que permite mostrar gráficos 3D en los Navegadores. Es un estándar muy reciente y aún está siendo desarrollado. WebGL hace

uso de la aceleración gráfica por hardware (GPU) y no necesita de *plugins*. Hace uso del elemento *canvas* de HTML5 para el dibujo de las escenas. Debido a que utiliza este elemento, WebGL se integra perfectamente con la interfaz del *Document Object Model*.

WebGL usa el *shading* del estándar OpenGL. De hecho, la semántica de WebGL le resultará familiar a cualquiera que conozca OpenGL. Esta similitud facilita el uso de esta tecnología web como estándar.

WebGL no maneja matrices. Esta funcionalidad se proporciona con bibliotecas de matrices escritas en JavaScrpit tales como glMatrix, TDL, o MJS. En este TFG se usa glMatrix para realizar las operaciones matriciales.

WebGL es una tecnología diseñada para trabajar directamente con la GPU (unidad de procesamiento gráfico) y puede ser difícil de usar en comparación con otros estándares web más accesibles. Por eso, han surgido muchas bibliotecas de JavaScript para resolver este problema. Entre ellas `Three.js` es la más popular en términos de número de usuarios. Es ligera y tiene un bajo nivel de complejidad en comparación con la especificación WebGL original. En este TFG no se han usado ninguno de estos recubrimientos y se ha programado directamente en WebGL la visualización 3D de los puntos del sensor Kinect.

WebGL permite la importación de escenas desde herramientas de creación de contenidos como Blender o Autodesk Maya.

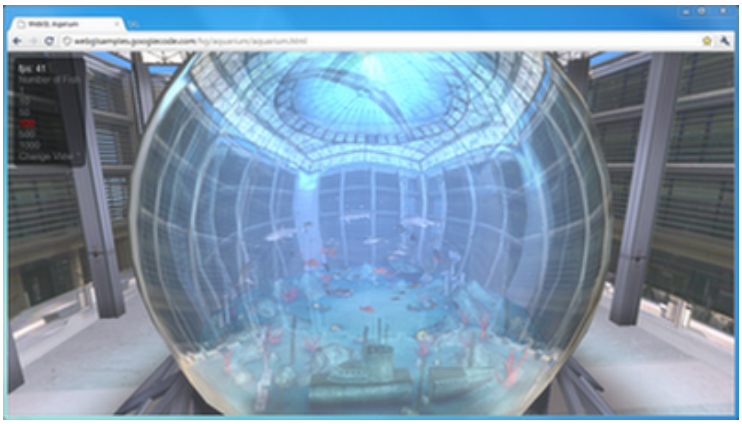

Figura 3.6: *Ejemplo de la potencia de WebGL en la creación de gráficos*

3.6. JavaScript

JavaScript (a veces abreviado como JS) es un lenguaje ligero e interpretado, orientado a objetos. Este lenguaje se conoce sobre todo por ser un estándar de facto en las páginas

web. También se utiliza en otros entornos sin navegador, por ejemplo en documentos PDF o aplicaciones de escritorio (mayoritariamente *widgets*).

Se utiliza principalmente en su forma del lado del cliente, permitiendo mejoras en la interfaz de usuario y páginas web dinámicas. También existe una forma de JavaScript del lado del servidor (*Server-Side JavaScript* o SSJS).

El JavaScript estándar es ECMAScript. A partir de 2012, todos los navegadores modernos soportan completamente ECMAScript 5.1. Los navegadores más antiguos soportan por lo menos ECMAScript 3. Una sexta revisión del estándar está en proceso.

3.7. AJAX

AJAX es el acrónimo de *Asynchronous JavaScript and XML*. En realidad AJAX no es un lenguaje de programación sino un nuevo uso de los estándares de desarrollo web ya existentes para crear aplicaciones web interactivas.

Ajax es una tecnología asíncrona. Los datos se solicitan al servidor web y se cargan en segundo plano sin interferir con la visualización ni el comportamiento de la página web. Con esta tecnología es posible realizar cambios sobre las páginas sin necesidad de recargarlas completamente. De esta forma se mejora la interactividad, velocidad y usabilidad en las aplicaciones. En la figura 3.7 se puede observar un esquema que explica el comportamiento asíncrono de AJAX frente al comportamiento clásico de la web.

El lenguaje en el que normalmente se efectúan las funciones de llamada de Ajax es JavaScript mientras que el acceso a los datos se realiza mediante XMLHttpRequest, objeto disponible en los navegadores actuales. El objeto *XMLHttpRequest* es usado para el intercambio de datos en segundo plano entre cliente y servidor web. La respuesta a una petición AJAX puede venir como texto plano, XML, JavaScript o JSON.

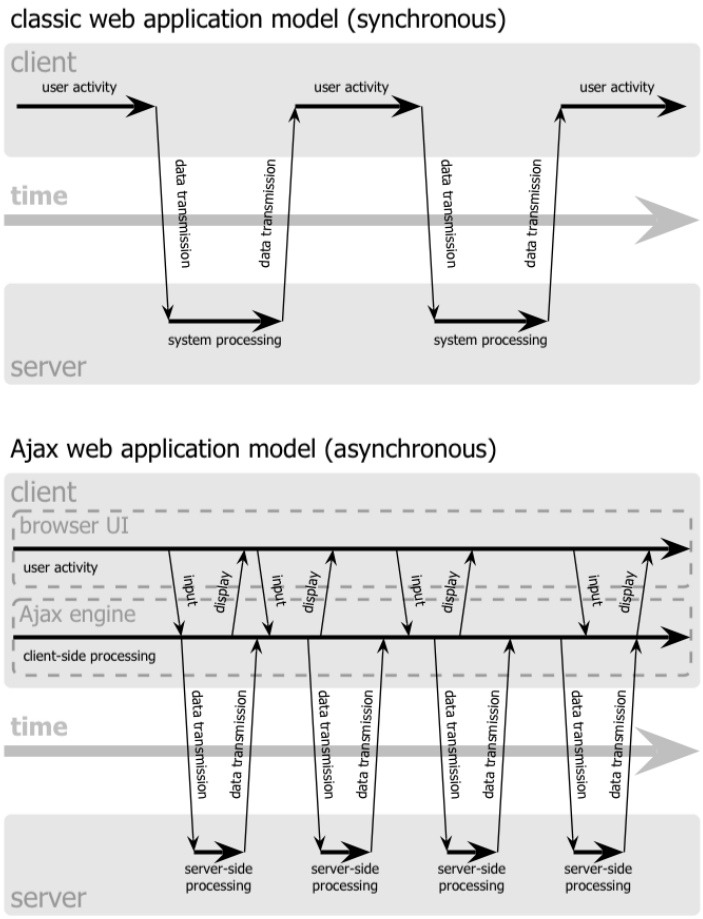

Figura 3.7: *La tecnología AJAX se basa en peticiones asíncronas*

Capítulo 4

Desarrollo

En este capítulo se explica en detalle Surveillance 5.1. En el primer apartado se explica el diseño y la arquitectura de la aplicación. La arquitectura de la aplicación consta de tres partes bien diferenciadas: Sensores y actuadores domóticos, servidor web y cliente web. Cada una de estas partes están explicadas en su apartado correspondiente.

4.1. Diseño

En este apartado se detalla el diseño y la arquitectura elegida para el sistema Surviellance 5.1. El diseño que se presenta se ha elegido porque cumple todos los objetivos y requisitos expuestos en el capítulo 2. La arquitectura del sistema tiene tres partes diferenciadas tal y como se observa en la figura 4.1.

- **Sensores y actuadores domóticos.** En este TFG se han integrado tres tipos de sensores a la aplicación web: una cámara web, un sensor Kinect y un sensor de humedad. También se ha añadido un actuador para el encendido de iluminación.

  Para el control de cada uno de estos sensores o actuadores existe un componente de JdeRobot. Los componentes para la cámara web y el sensor Kinect ya existían dentro de JdeRobot. Para el sensor de humedad y el actuador se ha creado un componente específico. Cada uno de estos componentes contiene los *drivers* necesarios para acceder al *hardware* correspondiente. Además, todos los componentes ofrecen una interfaz ICE para comunicarse y transmitir esos datos al servidor Web o a otros componentes. Ésto hace que cada componente pueda correr en una máquina distinta.

- **Servidor web.** Para la creación de la aplicación web, se ha optado por la programación de un servidor web en Ruby on Rails. En este servidor reside toda la lógica de la aplicación.

  El servidor web se encarga de crear las conexiones ICE con los distintos componentes. En este sentido crea dos hilos de ejecución, uno para la cámara y otro para el sensor de humedad. Estos hilos se encargan de recoger periódicamente los datos de los sensores y los ponen a disposición de los controladores de la aplicación web.

  Al recibir una petición HTTP, el servidor crea una instancia del controlador específico para esa petición. Según qué controlador se haya instanciado se llevará a cabo una de estas tres acciones:

  - En el caso de la cámara y el sensor de humedad, el controlador recoge la información que le ofrece el correspondiente hilo de ejecución que hace continuamente peticiones asíncronas al componente ICE respectivo
  - En el caso del sensor Kinect, se ejecuta un método remoto de la interfaz ICE que comparten servidor web y `openni1Server`. Este método devuelve síncronamente la información del sensor Kinect.
  - En el caso de la interfaz del componente para actuadores, se ejecuta síncronamente un método remoto. Este método no devuelve nada y activa o desactiva el actuador.

  La información recogida de la cámara y el sensor Kinect se recibe en el servidor web en forma de un *array* de bytes. En el caso del sensor de humedad se recibe como un *booleano*.

  Cada controlador tiene asociada una vista. Las vistas son plantillas HTML que se rellenan con los datos proporcionados por el controlador para crear al vuelo la página solicitada por el cliente web.

- **Cliente web.** El cliente web se encarga de realizar las peticiones HTTP de cada página solicitada por el usuario y de refrescar la información de los sensores mediante AJAX. También es tarea del cliente representar gráficamente los datos 3D de profundidad obtenidas del sensor Kinect.

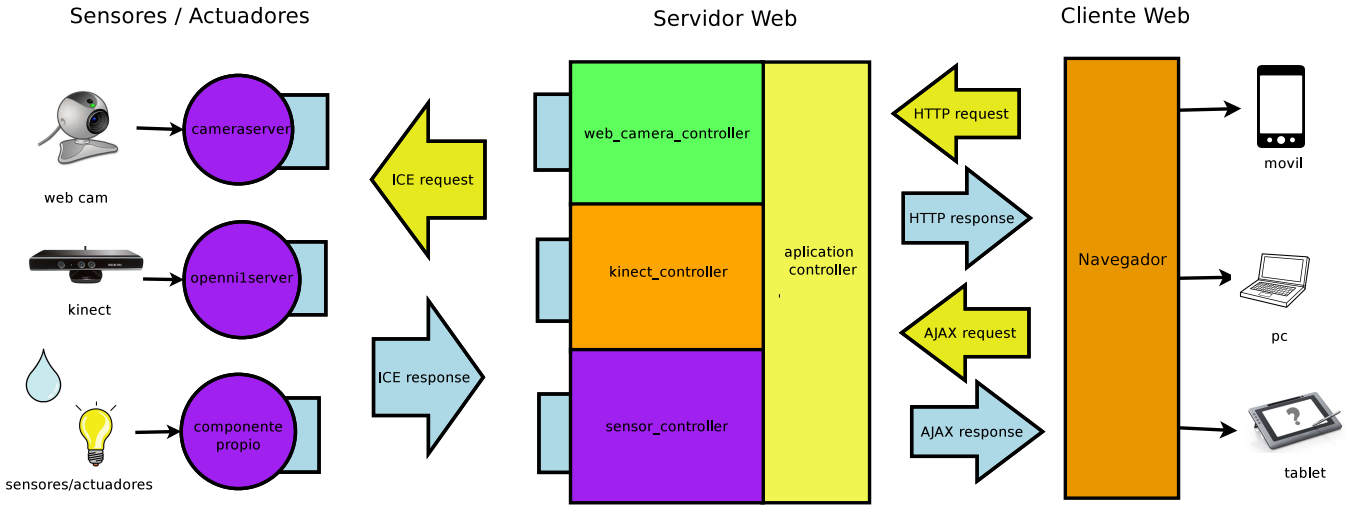

Figura 4.1: *Arquitectura del sistema domótico Surveillance 5.1*

4.1.1. Uso del middleware de comunicaciones ICE

El uso del *middelware* ICE es fundamental en la arquitectura y desarrollo de este TFG. Surveillance 5.1 implementa ICE como tecnología de comunicación con los sensores dentro del entorno de desarrollo Ruby on Rails.

ICE permite que se puedan implementar arquitecturas con varias máquinas distribuidas trabajando simultáneamente. Estas máquinas pueden estar cada una en una ubicación física con la única condición de que estén interconectadas en red. Esto confiere gran flexibilidad al diseño del sistema. Como se puede ver en la figura 5.1 del capítulo 5, para la realización de este TFG se han usado cuatro máquinas distintas, una para el servidor web, otra para el navegador y dos más para alojar distintos sensores.

En domótica es especialmente interesante tener varias máquinas trabajando simultáneamente. En muchos casos, los sensores y actuadores necesitan de procesadores con poca potencia computacional que pueden ser alimentados con baterías o placas solares. El bajo precio de estos computadores tipo Raspberry Pi o MK802 permite colocar varios cubriendo todas las zonas del recinto. Mientras, el servidor web sobre el que recae la mayor carga computacional, puede ser una máquina potente conectada a la corriente eléctrica y con salida a Internet. La figura 4.2 ilustra este concepto. Este diseño ofrece flexibilidad en cuanto a qué procesador va a alojar cada funcionalidad, si se alojan varias funcionalidades en un mismo procesador y el tipo de procesador a usar.

Además, este diseño hace que el sistema sea fácilmente escalable, pues facilita la integración de cualquier componente que implemente una interfaz ICE. De esta forma, podemos incluir tantos componentes como queramos, ya sean del mismo tipo, por ejemplo

varias cámaras de vigilancia o de otro tipo.

Otra ventaja que ofrece este diseño es que es multilenguaje. Con este tipo de *middleware* de comunicaciones cada componente puede estar desarrollado en un lenguaje distinto, y a su vez, distinto al del servidor web. En este TFG se han incluido los componentes `cameraserver` y `openni1Server` escritos en C++ y un componente para sensores-actuadores desarrollado en Python. El servidor web está escrito en Ruby. Los navegadores en el lado del cliente ejecutan Javascript.

Además, el diseño elegido es independiente de la plataforma, es decir, no importa el sistema operativo que se use en cada parte del sistema. Una muestra de ello es que en este TFG se ha usado el sistema operativo Raspbian para la Raspberry, Ubuntu para el servidor Web y `cameraserver` y Mac OS y Android para los navegadores.

Por último, el uso de ICE dentro de JdeRobot facilita enormemente la comunicación entre componentes. Hay que tener en cuenta que dentro de este proyecto se trabaja en áreas tan distintas como visión e inteligencia artificial, localización espacial o domótica. Cada una de estas áreas utiliza hardware específico y cada componente se puede programar en un lenguaje de programación distinto. Por último, es posible que personas que trabajan en un proyecto no coincidan temporalmente. En este entorno, resulta de gran ayuda el poder interactuar con componentes escritos por otros simplemente invocando métodos de una interfaz ICE de objetos distribuidos.

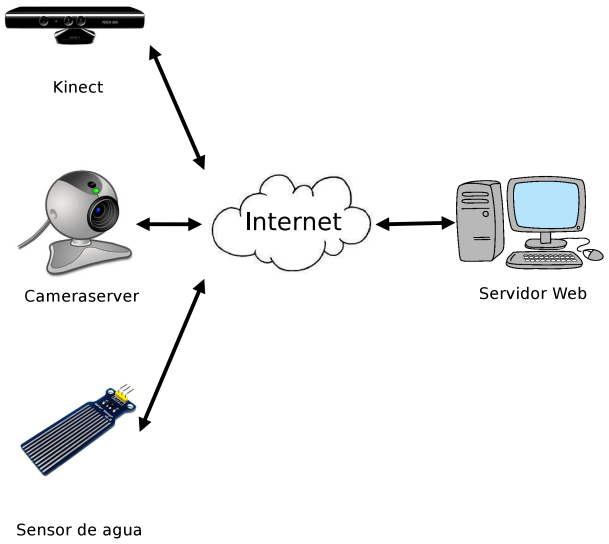

Figura 4.2: *La tecnología ICE hace posible la separación de componentes y servidor Web*

4.2. Sensores-actuadores domóticos

En esta sección se detalla el funcionamiento de los tres componentes de JdeRobot usados en este TFG. Los dos primeros, `cameraserver` y `openni1Server` ya existían previamente y el tercero, `GPIOServer` se ha creado específicamente para esta aplicación.

4.2.1. Cameraserver

Cameraserver [19] es el componente de Jderobot encargado de acceder a la cámara web para entregar vía ICE las imágenes necesarias para *streaming* de vídeo. El funcionamiento de este componente se detalla en el capítulo de Infraestructura necesaria 3.

4.2.2. Openni1Server

`Openni1Server`[20] es el componente de Jderobot necesario para acceder al sensor Kinect. En esta aplicación, este componente entrega por medio de una conexión ICE las imágenes de profundidad al servidor web. El funcionamiento de este componente se explica en el capítulo de Infraestructura necesaria 3.

4.2.3. GPIOServer

`GPIOServer` es un componente creado específicamente en este TFG para acceder a los sensores-actuadores conectados a una Raspberry Pi. Este componente está escrito en Python y como cualquiera desarrollado por el grupo de robótica de la URJC implementa una interfaz ICE.

Esta interfaz ofrece tres métodos. El primero de ellos, `getValue()` devuelve un entero que representa el estado del sensor de humedad. Un 1 significa que no hay alarma y un 0 que sí la hay. Los métodos `switchOn` y `switchOff` no devuelven nada y sirven para activar o desactivar el actuador.

```
interface Sensor
{
    int  getAlarm();
    void switchOn();
```

```
        void switchOff();
    };
```

En las primeras líneas de código de este componente se importa la biblioteca necesaria para el middleware ICE. La interfaz ICE no se puede importar tal cual está ya que está escrita en el lenguaje neutro en el que se escriben los `Slices` de ICE. Es necesario cargarla mediante en método `LoadSlice`. Este método convierte en tiempo de ejecución el `Slice` a lenguaje Python.

Por último, se carga la biblioteca `wiringpi2` [4]. Esta biblioteca es la que permite acceder a los pines GPIO de la Raspberry PI.

```
import sys, traceback, Ice, time
import wiringpi2 as wiringpi

Ice.loadSlice('Sensor.ice')
import Demo
```

En el cuerpo del programa principal se crea el servidor ICE. Este servidor se crea pasándole directamente la configuración como un *string*. Por este motivo, no es necesario fichero de configuración.

```
ic = Ice.initialize(sys.argv)
adapter = ic.createObjectAdapterWithEndpoints("SimpleSensorAdapter
   ", "default -p 10000")
object = SensorI()
adapter.add(object, ic.stringToIdentity("SimpleSensor"))
adapter.activate()
```

Además, se implementan los métodos que se definen en la interfaz ICE haciendo uso de la librería `wiringpi2` para leer o escribir el valor de un determinado pin.

Los pines que van a ser usados como pines de entrada o salida deben inicializarse. Una vez hecho ésto, los métodos `digitalRead()` y `digitalWrite()` leen y escriben respectivamente en los pines GPIO de la Raspberry Pi.

Para el método `getValue()` se lee el pin número 22 y se devuelve en forma de entero el resultado de la lectura. En este pin se conecta el sensor de humedad. Este sensor entrega una corriente de 3.3 Voltios si no hay alarma y 0V si hay alarma.

Los métodos `switchOn` y `switchOff` escriben el valor 1 ó 0 en el pin número 17. Esta escritura se traduce en un voltaje de 3.3 V si se escribe un uno y 0 V si se escribe un cero. Esta corriente es la que se usa como corriente de control en el relé sobre el que actúa. Este relé deja pasar una corriente de 220 V en función de si existe o no corriente de control.

```
class SensorI(Demo.Sensor):
    def getAlarm(self, current=None):
  value=wiringpi.digitalRead(22)
        print value
  return value

    def switchOn(self, current=None):
  print "switch_on"
        wiringpi.digitalWrite(17,1)

    def switchOff(self, current=None):
        print "switch_off"
        wiringpi.digitalWrite(17,0)
```

4.3. Servidor web

En el diseño de esta aplicación, el servidor web se encarga, por un lado, de comunicarse con los sensores y actuadores domóticos usando tecnología ICE, y por el otro responder las peticiones HTTP que le haga el cliente web.

Dentro del servidor web se han creado 4 controladores dentro del entorno Rails: `application_controller`, `web_camera_controller` , `kinect_controller` y

`sensor_controller`.

4.3.1. Controlador de aplicación

El controlador de aplicación es el encargado de importar todas las librerías y paquetes necesarios para el buen funcionamiento de la aplicación web. Estos paquetes se importan mediante los comandos `require` o `require_relative`. La diferencia entre ellos es que el primero busca los paquetes en la ruta por defecto de Rails mientras que el segundo busca los paquetes en el mismo directorio desde el que se lance el servidor.

En el caso de Surveillance 5.1 se requieren varios paquetes Ruby. El primero es la librería necesaria para usar el *middleware* ICE. La segunda librería necesaria es `oily_png` para el procesamiento de imágenes. Estas dos librerías están instaladas en la ruta por defecto de Rails.

Las interfaces ICE necesarias están en el directorio de trabajo desde donde se lanza la aplicación. Existen dos formas de importar las interfaces ICE:

- La primera forma es la usada para importar la interfaz de `cameraserver` y `openni1server`. Esta interfaz se ha traducido del lenguaje neutro que usa ICE para escribir *Slice* a Ruby. Esta traducción es automática y se realiza ejecutando en el terminal el comando `slice2rb camera.ice`. Como la librería está escrita en Ruby se puede importar directamente con `require_relative`
- La segunda forma es la usada si el *Slice* no ha sido traducido a Ruby previamente. Existe una forma de traducirlo en tiempo de ejecución para que esté disponible para el resto de la aplicación. Esto se hace mediante la orden `Ice::loadSlice("Sensor.ice")`.

```
require 'Ice'
require "oily_png"
require_relative 'camera.rb'
Ice::loadSlice("Sensor.ice")
```

Antes de crear la instancia del controlador específico, el controlador de aplicación o `aplication_controller` ejecuta cuatro métodos. Este comportamiento se consigue gracias a la siguiente línea de código que hace que antes de instanciar cualquier controlador se

ejecuten los métodos especificados.

```
before_action :authorize, :ice_camera, :ice_kinect, :ice_sensor
```

Método authorize

El primero de los métodos que se invocan es `authorize`. Este método se encarga de buscar dentro de la base de datos de usuarios el que tenga un identificador que coincida con el identificador de sesión. En caso de que no se encuentre un usuario con esa identificación de sesión, es decir, que el usuario todavía no se haya identificado, se le redirige a la página de *login* y se muestra un mensaje de alerta.

```
def authorize
    unless User.find_by(id: session[:user_id])
        redirect_to login_URL, notice: "Please log in"
    end
end
```

Este identificador de sesión es la herramienta que ofrece Rails para dotar de estado a las conexiones HTTP. La información de sesión puede guardarse en forma de *cookies*, como tabla en la base de datos en el servidor o como almacenamiento local en el cliente[5]. Surveillance 5.1 utiliza las *cookies* como método de almacenamiento de sesión ya que la información que viaja en las peticiones HTTP es pequeña y consta sólo de usuario y contraseña.

La página de *login*, tal y como muestra la figura 4.10 en la página 56, ofrece un formulario donde autenticarse.

Método ice_camera

El método `ice_camera` se encarga de crear la conexión ICE con el componente `cameraserver` y de pedirle periódicamente imágenes.

Para crear la conexión ICE con `cameraserver` se necesita un archivo de configuración. Este tipo de archivos contienen un determinado número de parejas nombre-valor, cada una en una línea. Este fichero de configuración es estándar en el entorno JdeRobot. Tiene extensión `.cfg` y entre sus parámetros más importantes indica la IP y puerto de la máquina con la que crearemos la conexión ICE. El contenido del fichero de configuración que se usa en método `ice_camera` es el siguiente:

```
Cameraview.Camera.Proxy=cameraA:default -h localhost -p 9999
```

Cuando los componentes ICE se lanzan desde el terminal, la ruta donde se encuentra el archivo de configuración se especifica con la propiedad `Ice.Config`. Para proporcionar esta información, desde dentro del servidor Rails, se crea un *array* que contiene el *string* con la ruta del archivo de configuración. Con la información del *string* se crea un *proxy* del tipo `cameraserver`. Es este *proxy* sobre el que se ejecutan las llamadas a los métodos remotos que se hayan definido en la interfaz ICE.

```
def ice_camera
    if !@@camera_ice_connection
        @@time = Time.now
      status = 0
        ic = nil
        arguments= ["--Ice.Config=cameraview.cfg"]
        ic = Ice::initialize(arguments)
        base = ic.propertyToProxy("Cameraview.Camera.Proxy")
        cprx_camera = Jderobot::CameraPrx::checkedCast(base)
        puts "ICE camera connection OK"
```

Es muy importante crear una sola conexión ICE por cada componente. La variable de clase `@@camera_ice_connection` asegura que sólo habrá una sola conexión ICE abierta. Esta variable es de tipo *booleano* y está inicializada a *false*. Cuando se crea la conexión ICE, este *booleano* se pone a *true* lo que hace que no se vuelva a realizar ninguna conexión ICE.

Una vez creada la conexión con el componente `cameraserver` el método `ice_camera` crea un hilo de ejecución que se encargará de pedir imágenes periódicamente. La figura 4.3 representa el funcionamiento con varios hilos del servidor web.

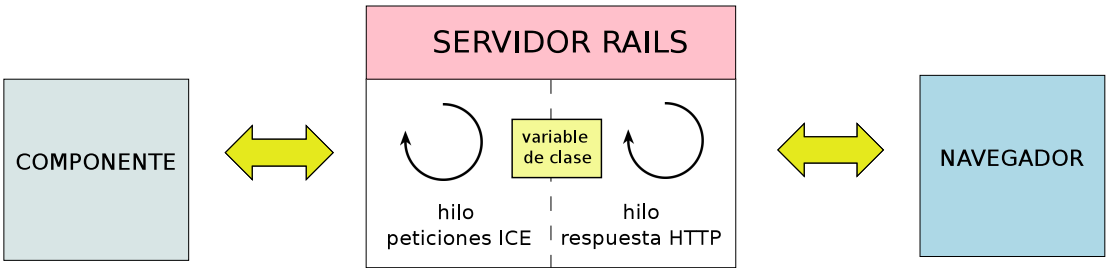

Figura 4.3: *El servidor crea dos hilos de ejecución, uno para refrescar datos mediante ICE y otro para responder peticiones http*

El hilo de ejecución de peticiones ICE es un bucle donde se invoca el método `getImageData()` del *proxy* de `cameraserver`. Este método devuelve una estructura tal y como muestra la figura 4.4. Dicha estructura está formada, por un lado, por información sobre el tipo de imagen que contiene y, por otro lado, por un `array` de bytes que se corresponden con los valores en RGB8 de la imagen.

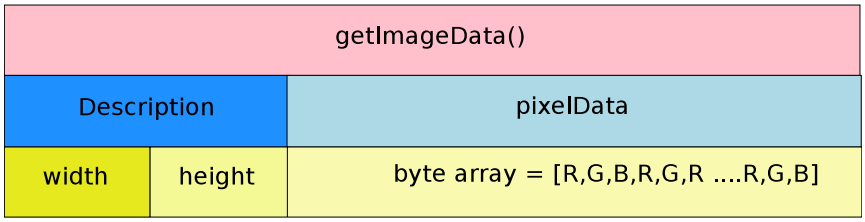

Figura 4.4: *Esquema de la estructura de imagen usada en JdeRobot*

Una vez que se tiene la imagen en RGB8, se transforma a PNG para disminuir el ancho de banda usado en las comunicaciones cliente web-servidor web. Para esta transformación se usa la gema Chunky PNG [6]. Esta gema es una librería para leer y escribir imágenes en PNG. Además incluye una funcionalidad muy útil para este proyecto que es la creación de URL virtuales. Por medio de este mecanismo de creación de URL virtuales se evita el paso por memoria, lo que ralentizaría todo el proceso de refresco de imágenes mediante AJAX. Esta URL se guarda en una variable de clase que es accesible por el hilo de ejecución que responde peticiones HTTP. La comunicación entre los dos hilos se puede ver en la figura 4.3

```
thr1 = Thread.new do
```

```
@@mutex = Mutex.new
    while @@keep_thread_alive do
        if time_flag
            t1 = Time.now
            time_flag=false
        end
        data = cprx_camera.getImageData()
        imagen=(ChunkyPNG::Canvas.from_rgb_stream(data.description
   .width,data.description.height,data.pixelData))\

                          .resize(160,120).to_data_url
        @@mutex.synchronize do
            @@image_ice=imagen
        end
        sleep(time_to_sleep)
        ice_requests=ice_requests +1
        if ice_requests > 12
            time_flag=true
            t2 = Time.now
            if (t2-t1)<1
                time_to_sleep=time_to_sleep + 0.01
            else
                time_to_sleep=time_to_sleep - 0.01
                if time_to_sleep<0
                    time_to_sleep = 0
                end
            end
            @@fps_ice = ice_requests/(t2-t1)
            ice_requests=0
        end
  end
```

Una funcionalidad muy importante de este hilo de ejecución es que ajusta automáticamente a un valor nominal de peticiones ICE por segundo. Existe una variable que indica el tiempo que se debe esperar entre peticiones ICE consecutivas. Para ajustar este valor de espera, se toma el tiempo en la primera petición ICE. Cuando se han realizado el valor nominal de peticiones por segundo, se vuelve a tomar un valor de tiempo. Si la

diferencia entre esos tiempos es menor a un segundo, se disminuye el tiempo de espera entre peticiones. Si la diferencia es mayor a un segundo se aumenta el tiempo de espera.

Si no existiera este mecanismo de autoajuste, el servidor web saturaría de peticiones ICE al componente `cameraserver`. Además, con este mecanismo se asegura que el sistema será estable sea cual sea el ancho de banda de la conexión entre servidor web y `cameraserver`.

Por último, como el valor de peticiones ICE por segundo es muy interesante para el funcionamiento de la aplicación, este valor se guarda para que esté disponible en el hilo que responde las peticiones HTTP.

El uso de variables de clase en este sistema se debe al funcionamiento estándar del servidor Rails que asegura la independencia entre peticiones. En este funcionamiento, ninguna variable sobrevive de una petición HTTP a la siguiente. En general este mecanismo resulta muy útil para el programador, ya que asegura que no existan condiciones de carrera en el acceso a variables. La solución más común para el problema de compartición de datos entre peticiones es el uso de bases de datos. En este TFG se ha decidido no usar bases de datos ni escritura en ficheros para optimizar la velocidad de respuesta del servidor.

Para solventar los posibles problemas de condiciones de carrera en el acceso de los distintos hilos de ejecución a variables compartidas, se han usado técnicas de exclusión mutua. En particular se han empleado *mutex*. El lenguaje Ruby implementa este tipo de herramienta. Para su uso se crea una variable de tipo `Mutex`. Cuando se quiere modificar una variable sin riesgo de condiciones de carrera se invoca el método `.synchronize` de la variable `Mutex` que se ha creado. De esta forma, se pueden crear tantos `Mutex` como variables compartidas se tengan.

Existe la posibilidad de usar un recubrimiento para los hilos de ejecución llamado *background workers* o el uso de gemas que son otro recubrimiento adicional para esta solución. Estos *background workers* se ejecutan en un hilo paralelo al de la aplicación principal y se comunican con él por medio de mensajes. De esta forma se evitan posibles condiciones de carrera. La explicación del funcionamiento y utilización de estas herramientas queda fuera del ámbito de este TFG.

Método ice_kinect

El método `ice_kinect` es el encargado de realizar la conexión con el componente `openni1Server`. La creación de dicha conexión es análoga a la explicada para el método `ice_camera`.

La diferencia entre estos dos métodos es que `ice_kinect` no crea un hilo de ejecución para la petición periódica de imágenes. Este método sólo crea la conexión y guarda en una variable el *proxy* creado. Es el controlador específico el que pide las imágenes a `openni1Server` bajo demanda. Estas peticiones se harán de forma síncrona con las peticiones HTTP tal y como muestra la figura 4.5. Por tanto, es responsabilidad del navegador el ritmo de petición de dichas imágenes.

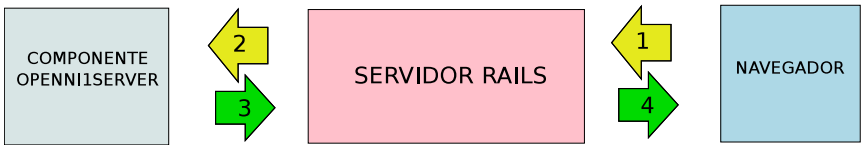

Figura 4.5: *Las peticiones de imágenes de profundidad son síncronas con las peticiones HTTP*

La diferencia de funcionamiento entre los dos métodos se debe a dos factores:

- En el caso del *streaming* de vídeo las imágenes son procesadas en el servidor web. El proceso de transformación a PNG es un proceso computacionalmente costoso. Si se hace de forma asíncrona a las peticiones HTTP se gana velocidad en respuesta.

- En el caso del *streaming* de imágenes de profundidad, las imágenes se envían sin comprimir ya que se necesitan los valores de cada píxel para el dibujo de la nube de puntos en 3D. Por lo tanto, el tener un hilo paralelo de peticiones de imágenes de profundidad no aporta apenas velocidad de respuesta al servidor web.

  Por otro lado, la tasa de refresco que se puede alcanzar con las imágenes de profundidad no es muy alta ya que el navegador tiene que construir la imagen de profundidad y realizar el dibujo de imagen 3D. Esto significa una carga computacional considerable y es finalmente lo que limitará la tasa de refresco de este *streaming* y no la velocidad de respuesta del servidor web.

Método ice_sensor

El método `sensor_controller`, al igual que los dos métodos descritos anteriormente, crea una y sólo una conexión ICE. Una vez hecho ésto, crea un hilo de ejecución.

Este hilo de ejecución se encarga de realizar cada segundo peticiones ICE al componente del sensor de humedad. Estas peticiones se realizan mediante el método *getValue()*. Este método devuelve un *booleano* indicando si hay o no alarma. Este *booleano* se guarda en una variable compartida para que el hilo de ejecución que responde las peticiones HTTP pueda acceder a esta variable.

```
def ice_sensor
    if !@@sensor_ice_connection
    status = 0
    ic = nil
    @@sensor_ice_connection=true
    begin
      ic = Ice::initialize()
      base = ic.stringToProxy("SimpleSensor:default -h
   192.168.1.18 -p 10000")
      @@cprx_sensor = Demo::SensorPrx::checkedCast(base)
            puts "ICE sensors connection OK"
            thr1 = Thread.new do
            while true do
                    @@value= @@cprx_sensor.getAlarm()
                    sleep(1)
                end
            end
        end
    end
end
```

4.3.2. Controladores secundarios

Para cada petición HTTP, el `application_controller` revisa la URL de la petición y la hace coincidir con un controlador secundario concreto. Se crea una instancia de este controlador secundario y se llama al método específico que genera la vista que se corresponda con la petición HTTP. Esta correspondencia entre peticiones HTTP y métodos se almacena como una lista en el fichero `routes.rb`.

```
Survillance::Application.routes.draw do
  resources :kinects
  resources :web_cameras
```

```
  resources :users
  resources :sensors

  get "admin/index"
  get 'admin' => 'admin#index'
  controller :sessions do
      get 'login' => :new
      post 'login' => :create
      delete 'logout' => :destroy
  end
  get "sensors/refresh", to: 'sensors#refresh'
  post "sensors/refresh", to: 'sensors#refresh'
  post "sensors/switchOn", to: 'sensors#switchOn'
  post "sensors/switchOff", to: 'sensors#switchOff'
  post "web_cameras/refresh2", to: 'web_cameras#refresh2'
  post "kinects/refresh3", to: 'kinects#refresh3'
  get "sessions/create"
  get "sessions/destroy"

  root 'admin#index'
end
```

En esta memoria se detalla el funcionamiento de los controladores secundarios más importantes: `web_camera_controler`, `kinect_controller` y `sensor_controller`.

Controlador secundario `web_camera_controller`

Este controlador consta de varios métodos. El más importante es el que se encarga de refrescar la imagen de la cámara cada vez que recibe una petición AJAX del cliente.

```
def refresh_camera
    @fps_ice=@@fps_ice.round
    @@mutex.synchronize do
      @image= @@image_ice
    end
```

```
  respond_to do |format|
    format.js
  end
end
```

Asíncronamente, este método se encarga de tomar la variable de clase que almacena la imagen (variable compartida entre los dos hilos de ejecución) y transformarla en una variable de instancia. Además, se hace la misma transformación para la variable que guarda el número de peticiones ICE por segundo. Este comportamiento asíncrono se refleja en la figura 4.6.

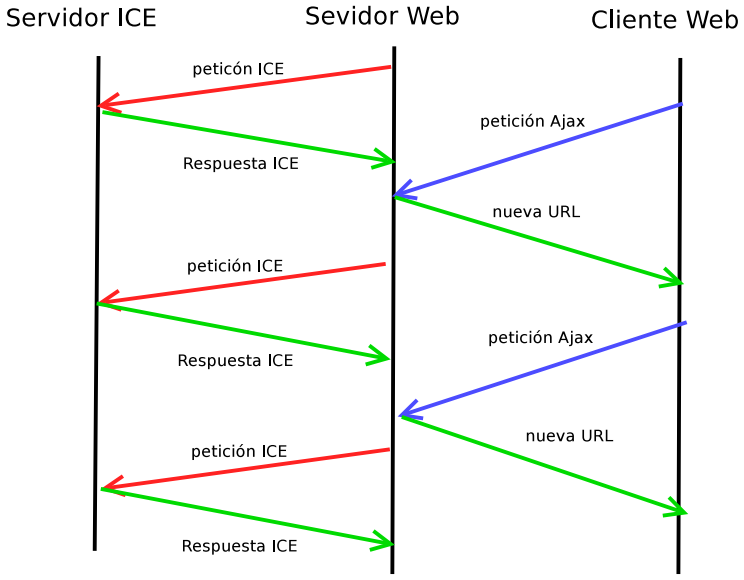

Figura 4.6: *Esquema de peticiones asíncronas en el flujo de vídeo*

Cuando se accede a la variable con la URL de la nueva imagen se hace mediante un *mutex*. Con este mecanismo se evitan posibles condiciones de carrera.

Para la variable de número de peticiones ICE por segundo no es necesario este mecanismo de protección. Esta variable es un dato secundario en el funcionamiento de la aplicación y su acceso mediante mecanismos de exclusión mutua podría ralentizar la respuesta.

Como respuesta a la petición AJAX se envía un código en Javascript que se ejecuta en el cliente. Este código sirve para remplazar la nueva URL por la antigua y refrescar el valor del número de peticiones ICE por segundo. Para ello, se usa el siguiente código de JQuery:

```
$('#camera_image').html("<%= escape_javascript image_tag(@image,
   size: "500x281",:onload=>"imageLoad()") %>");
$('#fps_ice').html("<%= @fps_ice %>");
```

Controlador secundario `kincet_controller`

Al igual que el controlador anterior, `kinect_controller` se encarga entre otras cosas de refrescar la imagen de profundidad. Para ello se invoca el método `refresh_kinect`. Este método usa la conexión ICE creada en el controlador principal con `openni1Server`. La imagen de profundidad se consigue síncronamente (refresco síncrono). Cada vez que el cliente web solicita una imagen, el servidor web la solicita al servidor ICE para responder la solicitud. La figura 4.7 refleja este comportamiento síncrono.

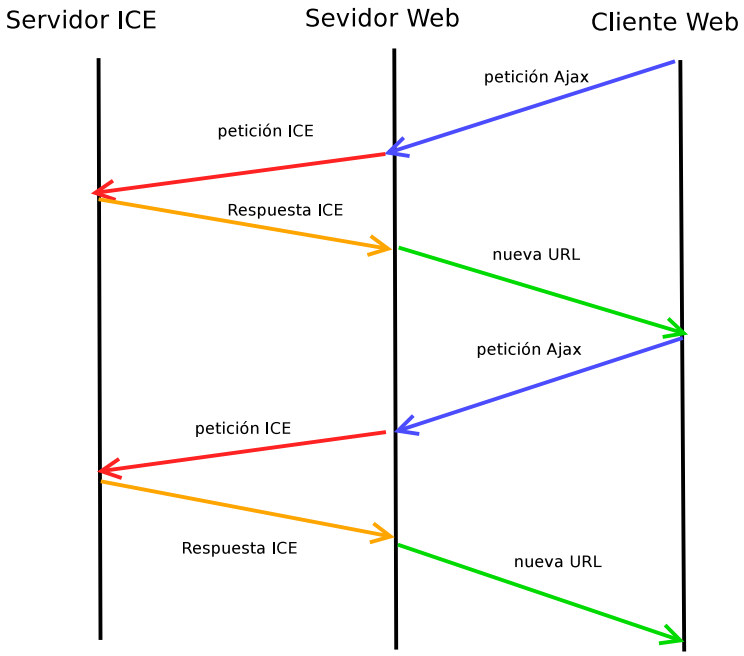

Figura 4.7: *Esquema de peticiones síncronas en el flujo de imágenes de profundidad*

```
def refresh_kinect
    aux= @@cprx_kinect.getImageData().pixelData.bytes
    @vertices=[]
    for i in 0..120
        for j in 0..160
```

```
            @vertices[160*i+j]= aux[(640*3*2*2*i+j*2*2*3)]
          end
      end
end
```

La imagen de profundidad se obtiene invocando el método `getImageData()`. Este método devuelve una estructura idéntica a la mostrada en la figura 4.4 pero, en este caso, el *array* de bytes transporta la información de profundidad. En el byte que correspondería al nivel de rojo de un píxel se almacena la distancia de ese píxel dividida por la máxima distancia de toda la imagen, es decir, la distancia normalizada.

La distancia que proporciona el sensor Kinect tiene una precisión tal que se debe representar como un número de 16 bits. Debido a esto, el componente `openni1server` usa el byte que correspondería al verde en una imagen convencional para guardar los 8 bits más significativos del valor de la distancia y el byte que correspondería al azul para los 8 bits menos significativos. Se puede ver un esquema de la estructura de imagen que obtenemos de `openniserver` en la figura 4.8

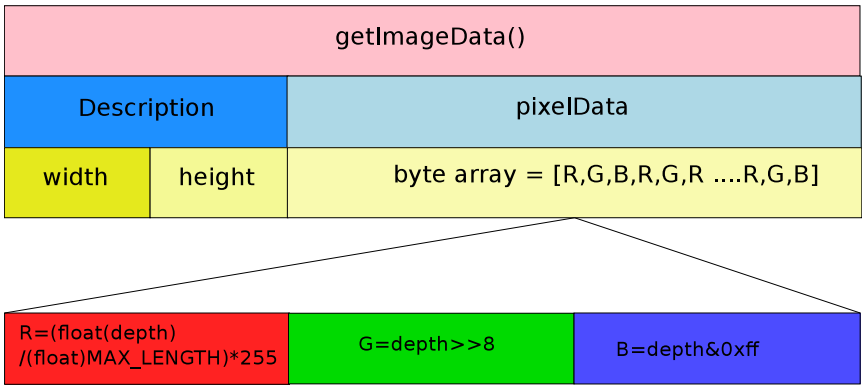

Figura 4.8: *Esquema de la estructura de la imagen de profundidad usada en JdeRobot*

A diferencia del `streaming` de vídeo, en este caso no se puede comprimir la imagen a PNG ya que en el lado cliente se necesita tener la imagen sin comprimir para poder extraer de él los valores exactos de distancia de cada píxel. El sensor Kinect ofrece una resolución de 640x480. Ésta es una resolución demasiado grande para transmitirla en *streaming*. En Surveillance 5.1 se ha optado por reducir la imagen hasta conseguir una resolución de 160x120. Además, se elimina el primer byte de cada píxel ya que la información normalizada que contiene es redundante. La imagen reducida se envía al cliente en formato JSON.

Controlador secundario `sensor_controller`

Este controlador, como en los casos anteriores, tiene un método que permite refrescar una variable cuando se recibe una petición AJAX, pero además, tiene dos métodos para la activación-desactivación de actuadores en el servidor ICE. Cada vez que se recibe una petición AJAX para refrescar el valor del sensor se consulta el valor de la variable de clase (variable compartida con el hilo de peticiones al servidor ICE) y se envía al cliente en formato JSON. Al igual que el controlador de imagen convencional, este controlador tiene un mecanismo de refresco de datos asíncrono.

```
def refresh
    if @@value==1
        alarms= true
    else
        alarms = false
    end
  respond_to do |format|
      format.json { render :json => alarms.to_json }
  end
end
```

Para el caso de los métodos de activación-desactivación de actuadores, al recibir la petición AJAX correspondiente se ejecuta un método remoto del servidor ICE. El método ejecutado provoca una acción en el Servidor ICE que activará o desactivará alguno de sus GPIO.

```
def switchOn
    @@switch.switchOn()
end
def switchOff
    @@switch.switchOff()
end
```

4.4. El cliente Web

El cliente web tiene un papel fundamental en Surviellance 5.1 ya que se encarga de enviar peticiones AJAX al servidor para refrescar los datos de los sensores. Además, en el caso del *streaming* de profundidad el navegador se encarga de dibujar en pantalla tanto la imagen 2D como la imagen 3D.

El cliente web está programado en JavaScript. Usa el nuevo estándar HTML5 para la creación de páginas web. Algunos elementos usados, como `canvas` o `audio`, son propios de este lenguaje. La apariencia de las distintas páginas se ha definido en una hoja de estilos CSS.

En este apartado se explican las partes principales de la aplicación en el lado del cliente. La figura 4.9 muestra un mapa del sitio web.

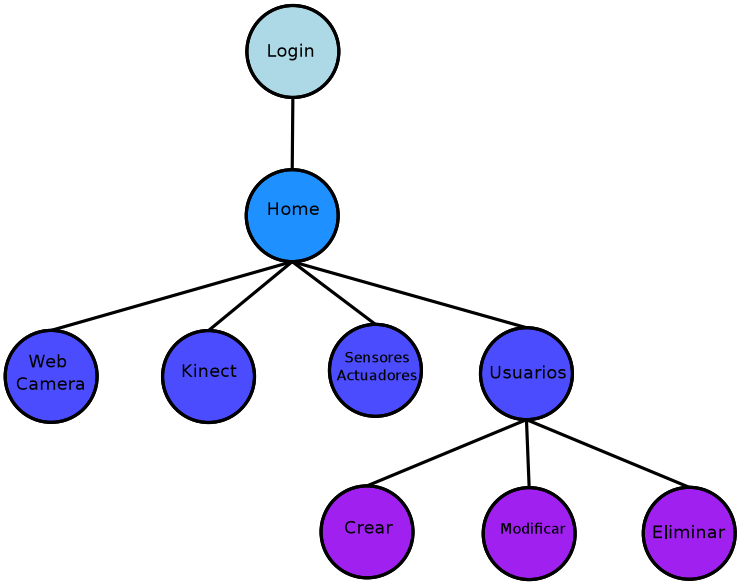

Figura 4.9: *Mapa del sitio web Surveillance 5.1*

4.4.1. Página de *login*

La primera página que se encuentra el usuario es la página de Login. Esta página tiene un formulario para que el usuario introduzca su *nick* y *password* tal y como se muestra en la figura 4.10. Si un usuario sin identificarse intenta acceder a cualquier página de la aplicación, es redireccionado a la página de *login*.

Una vez se hayan introducidos las credenciales oportunas el usuario accede a la página *home*. Ésta es la página principal de surviellance 5.1 y desde ella se puede acceder al

resto de páginas. Esta página tiene en su parte superior tres pestañas para redireccionar al usuario al *home*, a la página de *login* y a la página de gestión de usuarios. Además, existen 3 iconos en la parte inferior donde se puede acceder a los distintos sensores: cámara, Kinect y sensores-actuadores.

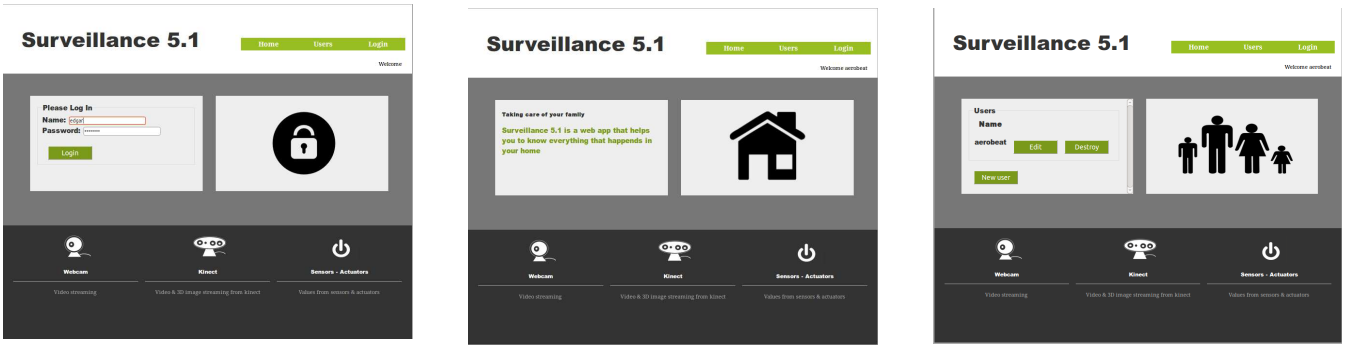

Figura 4.10: *De izquierda a derecha: página de login, página principal y página de gestión de usuarios de Surveillance 5.1*

4.4.2. Página de flujo de vídeo

En la figura 4.11 se muestra la página Webcam. En ella se ofrece el *streaming* de vídeo junto a una ventana que contiene información en tiempo real de la tasa de refresco de la imagen y del número de peticiones ICE por segundo.

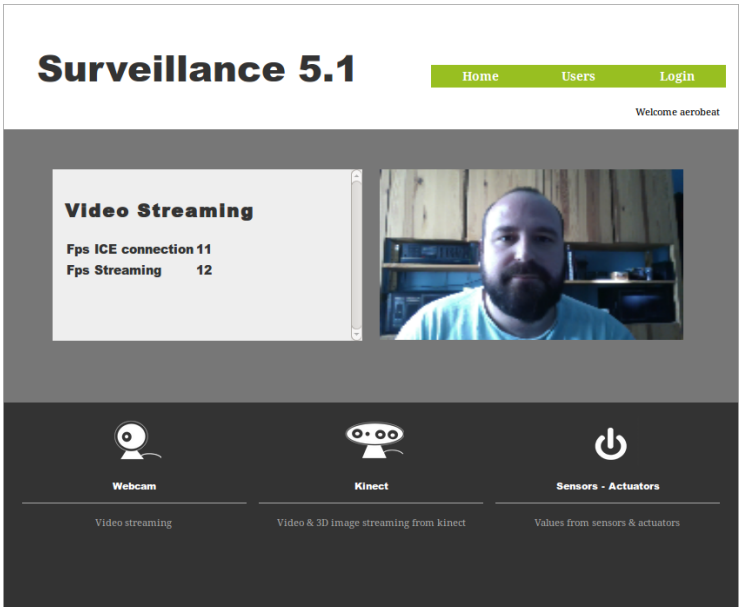

Figura 4.11: *Página de flujo de vídeo de Surveillance 5.1*

El streaming de vídeo se realiza como una sucesión de imágenes que el cliente va

solicitando al servidor a una frecuencia dada. El cliente al cargar la página, ejecuta una función escrita en JavaScript que programa un temporizador. Este temporizador se encarga de la petición periódica de imágenes mediante tecnología AJAX. La respuesta a cada petición es un código JQuery. Este código se ejecuta en el navegador nada más ser recibido y sirve para sustituir la antigua URL por una nueva. Una vez hecho esto, el cliente pide al servidor la imagen especificada en la nueva URL. El diagrama de peticiones entre servidor y cliente es el que se muestra en la figura 4.12. Este comportamiento resulta decisivo a la hora de la elección de navegador.

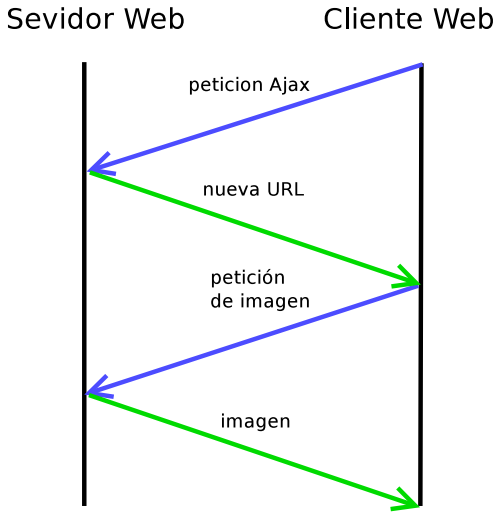

Figura 4.12: *Esquema de peticiones-respuestas entre cliente y servidor para el streaming de vídeo*

4.4.3. Página de flujo de imagen de profundidad

Tal y como se muestra en la figura 4.13, en la página Kinect aparecen dos ventanas. En la ventana de la derecha se puede ver el *streaming* de imagen de profundidad y en la ventana de la izquierda se observa la representación en 3D de la nube de puntos. Con las teclas W, A, S, D, O y L se puede mover la cámara dentro de la escena 3D para cambiar el punto de vista de dicha representación. También se puede ver la ventana de representación 3D a pantalla completa haciendo doble *click* encima de la ventana.

Para el *streaming* de las imágenes de profundidad se usa un procedimiento distinto al de las imágenes convencionales. En este caso, el navegador realiza también peticiones AJAX al servidor web, pero ahora el servidor en vez de responder con una URL responde directamente con el *array* de bytes que forman la imagen de profundidad codificados en JSON. Este *array* de puntos se utiliza tanto para crear la imagen de profundidad como

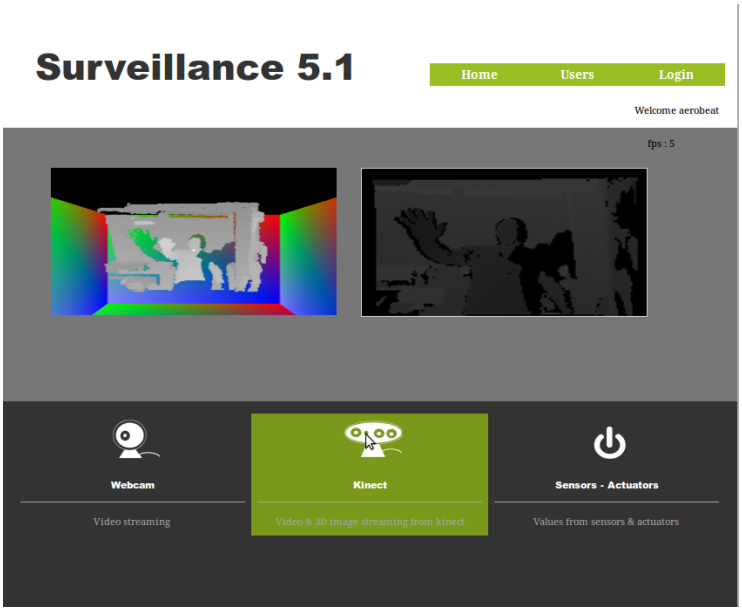

Figura 4.13: *Pagina de kinect Surveillance 5.1*

para el dibujo de la nube de puntos en 3D. Así se consigue con un solo envío de datos dos funcionalidades distintas, ahorrando ancho de banda una vez más.

Dibujo de las imágenes de profundidad

Cuando se recibe la respuesta AJAX con el *array* de bytes de la imagen de profundidad se llama a una función de *callback*. Esa función de *callback* dibuja desde JavaScript la imagen en la ventana del navegador y llama a la función necesaria para dibujar los puntos 3D.

```
function dibujar(data){
    var c=document.getElementById("myCanvas");
    var ctx=c.getContext("2d");
    var imgData=ctx.createImageData(160,120);
    for (var i=0;i<imgData.data.length/4;i++)
      {
      imgData.data[4*i+0]=(data[i]);
      imgData.data[4*i+1]=(data[i]);
      imgData.data[4*i+2]=(data[i]);
      imgData.data[4*i+3]=255;
      }
```

```
    ctx.putImageData(imgData,0,0);
}
```

Esta función hace uso del elemento *canvas* de HTML5. Como se puede observar se crea un *canvas* y se va rellenando con los valores del array de bytes. Se puede apreciar en el código que la imagen del *canvas* requiere 4 valores por píxel, ya que existe un valor para la transparencia de la imagen. Este valor se pone a 255 en todos los píxeles para que la imagen sea totalmente opaca. Como resultado se obtiene una imagen en blanco y negro donde los colores claros representan una distancia mayor que los colores oscuros.

Dibujo de la imagen 3D

El dibujo la imagen de puntos se realiza con WebGL. Para ello, en la función de *callback* de la petición AJAX se llama a la función `webGLStart()` que se encarga de todo lo necesario para dibujar una escena.

Esta función se debe llamar sólo una vez, ya que ella misma se encarga de refrescar la escena. Este refresco se consigue mediante la función `requestAnimFrame()` que dibuja la imagen 60 veces por segundo. De no ser por esta función, el movimiento de cámara por la escena 3D sería muy poco fluido.

La función `webGLStart()` prepara el *canvas* sobre el que se va a dibujar. También inicializa los *shaders* que se encargarán de que el sistema WebGL aplique las matrices modelo-vista y proyección a la escena.

Para representar una escena en WebGL, se deben definir los puntos de cada figura a dibujar e introducirlos en un *buffer*. Estos *buffers* antes de dibujarse pasan directamente a la memoria de la tarjeta gráfica donde se le aplican píxel a píxel todas las matrices de transformación que generarán la escena final.

Por último, se carga el *buffer* con los puntos de cada uno de los objetos a dibujar. Se aplican matrices de transformación para trasladarlos y rotarlos hasta que ocupen el lugar deseado y finalmente se dibujan.

```
    function webGLStart() {
    var canvas = document.getElementById("canvas");
    initGL(canvas);
```

```
    initShaders();
    initBuffers();
}
```

Creación de buffer de puntos

Como se explica en el apartado anterior, para dibujar en 3D un objeto es necesario guardar en un buffer todos los puntos que lo compongan. En este *buffer* se deben almacenar las coordenadas `x`,`y` y `z` de cada uno de los puntos a dibujar. Esta operación no es trivial ya que lo que se obtiene de `openni1Server` es un array de bytes con la distancia de cada punto en el eje `z`. Para calcular las coordenadas del eje `x` e `y` es necesario usar trigonometría básica. Estos cálculos se basan en el modelo de cámara *pin hole* que usa Kinect ilustrado en la figura 4.14

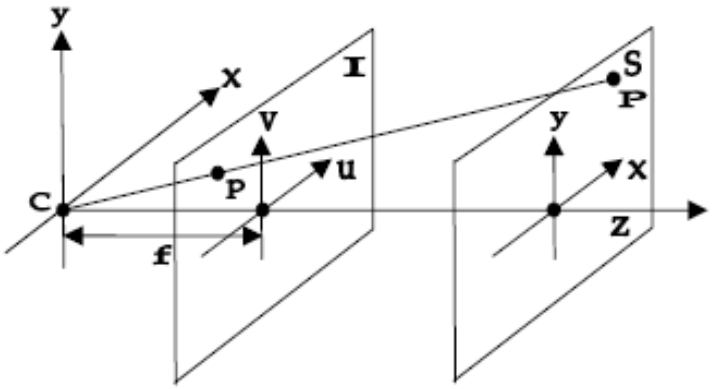

Figura 4.14: *Representación del modelo de cámara pin hole*

En la figura 4.15 se pueden ver las fórmulas trigonométricas usadas para el cálculo de las coordenadas cartesianas de cada píxel. Los ángulos `alpha` y `theta` son los ángulos de apertura horizontal y vertical del sensor Kinect. Los valores numéricos concretos se encuentran en el capítulo 5 de esta memoria.

Creación y movimiento de la Cámara

La cámara de observación, desde la cual se observa la escena en 3D, se puede mover por dicha escena.

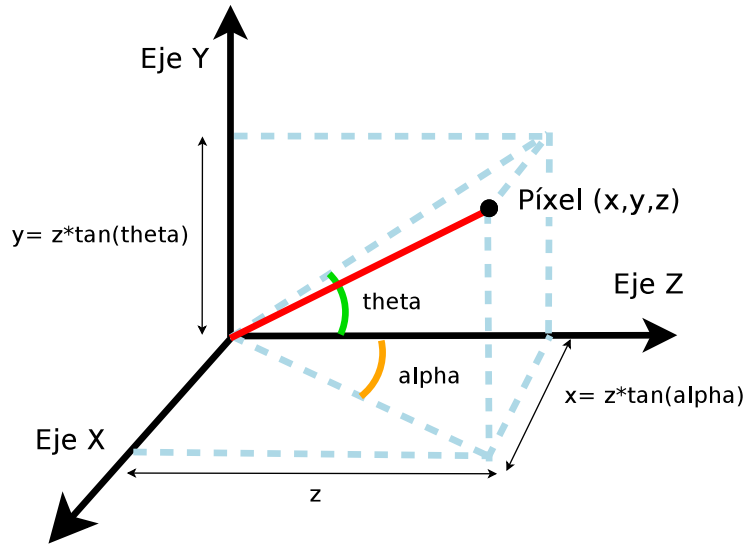

Figura 4.15: *Trigonometría aplicada en el cálculo de las coordenadas de cada píxel*

WebGL no implementa un objeto cámara como tal. Para simular una cámara y poder moverla lo que se hace es rotar y trasladar toda la escena respecto al punto (0,0,0). Ésto se debe a que WebGL ofrece siempre una vista desde este punto hacia el eje `z`.

El movimiento de la cámara es controlado por el usuario de la aplicación. Para conseguirlo se capturan los eventos del teclado para saber qué tecla se está pulsando en cada momento. La pulsación de estas teclas modifica los valores de las variables de velocidad de translación y velocidad de rotación que estaban inicializadas a cero. Estas variables de velocidad se multiplican por el tiempo transcurrido entre cada *frame*. De esta forma, se consiguen los valores exactos que hay que rotar y trasladar la escena para conseguir un movimiento fluido y realista.

Cuando se dejan de pulsar las teclas de movimiento, las variables de velocidad vuelven a cero con lo que no se produce ninguna rotación ni translación de la escena.

4.4.4. Página de sensores y actuadores domóticos

Por último, en la página de sensores y actuadores aparecen de nuevo dos ventanas. La ventana de la izquierda tiene dos botones para apagar y encender el actuador y en la ventana de la izquierda se observa si existen o no alarmas provenientes del sensor de humedad. En caso de alarma aparece el mensaje *Alarm!!!* con el fondo en rojo. Este mensaje permanecerá en pantalla mientras que el sensor de humedad entregue el valor de alarma. La figura 4.16 muestra esta pantalla. Además, se hace uso del elemento audio de HTML5 para reproducir un sonido de alarma.

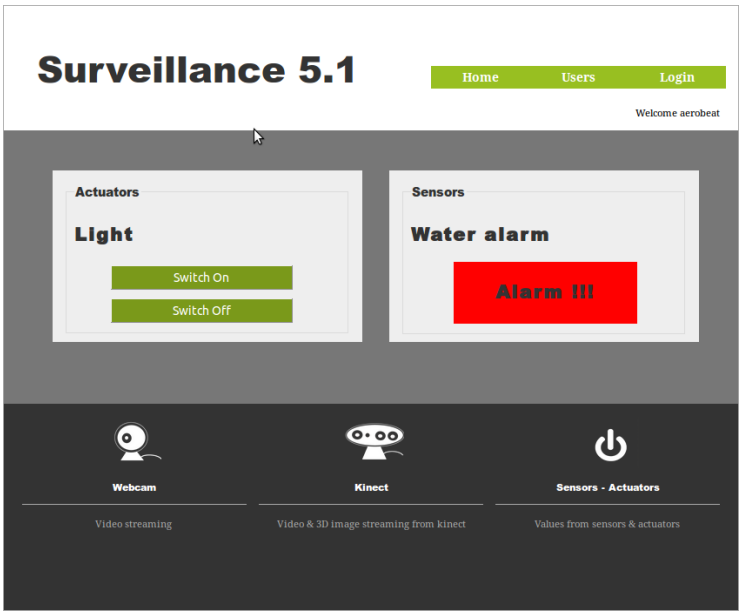

Figura 4.16: *Pagina de sensores Surveillance 5.1*

Capítulo 5

Pruebas

En este capítulo se presentan las pruebas realizadas al sistema y el hardware necesario para la reproducción de las mismas.

Las pruebas consisten en el despliegue real del sistema colocando cada una de sus partes en una localización distinta. Se han definido tres localizaciones con distintas conexiones a Internet y distintos anchos de banda. Las características de dichas conexiones han sido medidas con la herramienta web *Speed Test* [7]

- **Universidad** La primera localización es el campus de Fuenabrada de la Universidad Rey Juan Carlos. La Universidad dispone de una conexión con un ancho de banda que supera los 100 Mbps característicos de la tecnología Fast Ethernet, por lo que tomaremos ese valor de 100 Mbps como el ancho de banda disponible para las pruebas.
- **Domicilio 1** La segunda localización es un domicilio particular con una conexión de fibra óptica. Es una conexión asimétrica con un ancho de banda de 5 Mbps de subida y 50 Mbps de bajada.
- **Domicilio 2** La tercera localización es también un domicilio particular con una conexión de fibra óptica. Esta conexión ofrece 3 Mbps de subida y 30 Mbps de bajada.

Los ordenadores usados para las pruebas y su función dentro del sistema son los que se especifican en el apartado siguiente.

En las pruebas se va a medir la latencia y los fotogramas por segundo del *streaming* de vídeo. Sólo se ha considerado para las pruebas este *streaming* porque es el que tiene una mayor tasa de refresco, y por lo tanto, es el que será más sensible a restricciones de ancho de banda.

5.1. Hardware usado en las pruebas

Este apartado se especifica todo el hardware necesario para la reproducción de las pruebas presentadas en el siguiente apartado. Este Trabajo de Fin de Grado necesita para su realización de varias computadoras dada su arquitectura distribuida y su modelo cliente-servidor. En la figura 5.1 se puede observar una fotografía de cada una de las computadoras usadas y la función que desempeñan en esta aplicación web.

Figura 5.1: *Computadoras usadas en este TFG y su función*

Además, han sido necesarios sensores y actuadores de distinto tipo y de distinto ancho de banda. En la figura 5.2 se observa una fotografía de cada uno de estos sensores y actuadores.

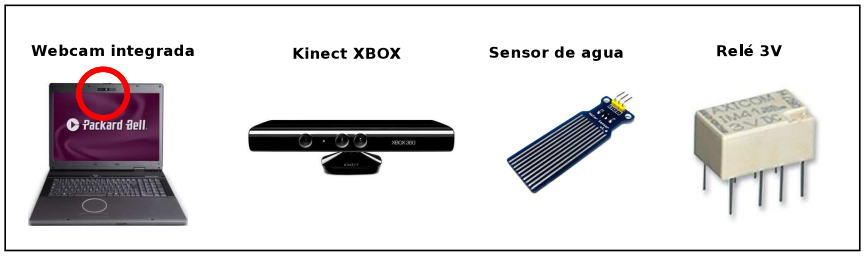

Figura 5.2: *Sensores y actuadores usados en este TFG y su función*

5.1.1. Sensores y actuadores domóticos

Servidor `cameraserver`

Como servidor de imágenes ICE se utilizó un ordenador portátil Packard Bell Easynote con un procesador Pentium(R) Dual Core 2.00GHz con 4GbiB de memoria RAM. En este pórtatil se instaló la distribución Debian de Jderobot 5. Esta máquina se usó sólamente para albergar el componente `cameraserver`.

Servidor `GPIOServer`

Para el control del sensor de humedad y el actuador se usa una Raspberry Pi. Esta computadora ofrece una serie pines de propósito general o GPIO (*General Purpose In/Out*. Manejando estos pines se consigue interactuar con los sensores y actuadores.

Raspberry Pi es un ordenador de placa reducida o (placa única) (SBC) de bajo costo, desarrollado en Reino Unido por la Fundación Raspberry Pi [18]. El diseño incluye un System-on-a-chip Broadcom BCM2835, que contiene un procesador central (CPU) ARM1176JZF-S a 700 MHz, un procesador gráfico (GPU) VideoCore IV, y 512 MiB de memoria RAM. El sistema operativo usado en la Raspberry pi es Raspbian, un derivado del sistema operativo Debian.

Sensor de humedad

Como ejemplo de que el sistema Surviellance 5.1 puede interactuar con sensores, se ha elegido un sensor de humedad de la marca Grove. Éste es un sensor muy simple que consta de 3 pines. Dos de ellos suministran una corriente continua de 3,3V al sensor y el tercero sirve para llevar la información del sensor: Manda 3,3 V cuando no hay humedad y 0 V cuando si la hay.

Actuador

Para la integración de actuadores en el sistema se utilizan los pines GPIO (*General Purpose In/Out*). Uno de estos pines se conecta a un relé de 3V de la marca Axicom [14] modelo IM41.

Este relé tiene 4 pines. En dos de ellos se conecta la Raspberry Pi y los otros dos sirven como interruptor para una corriente de 220V. La función de este relé es dejar pasar la corriente de 220V en función de la salida de los pines de la Raspberry Pi. Este sistema hace que cualquier aparato eléctrico que tengamos en un hogar sea susceptible de ser controlado.

Cámara

Para este Trabajo de Fin de Grado se ha usado una webcam integrada en el ordenador portátil Packard Bell. Es una cámara de uso doméstico. Su resolución es de 320x240 y tiene una tasa de refresco de 30fps.

Kinect

Kinect para Xbox o simplemente Kinect es un periférico que ofrece mediante una conexión USB una imagen RGB , una imagen de profundidad y audio. En la figura 5.3 están reflejadas las especificaciones oficiales distribuidas por Microsoft del sensor Kinect. Aunque la tabla refleje que el rango de actuación del sensor está entre 0.8 y 3.5 m realmente es bastante mayor, llegando hasta los 8-10 metros. Microsoft lo acotó a ese rango porque es donde tiene mayor precisión su algoritmo de reconocimiento gestual, y por lo tanto, es donde desempeña correctamente su función de interfaz de interacción con la consola.

Property	Value
Angular Filed-of-View	57° horz., 43° vert.
Framerate	approx. 30Hz
Nominal spatial range	640x480 (VGA)
Nominal Spatial resolution (at 2m distance)	3mm
Nominal depth range	0.8m - 3.5m
Nominal depth resolution (at 2m distance)	1 cm
Device connection type	USB (+ external power)

Figura 5.3: *Tabla de carácteristicas del sensor Kinect*

Para este TFG sólo se ha usado la Kinect como sensor de profundidad. Este sensor devuelve una imagen de profundidad de 640x580 píxeles de resolución y un refresco de 30 fps. Cabe destacar que el consumo de este sensor es alto y por ello no se puede alimentar directamente con un USB. A la hora de decidir la topología de la arquitectura de Surveillance 5.1 este alto consumo puede tener un papel importante.

5.1.2. Servidor Web

Para el desarrollo de este TFG se ha utilizado un ordenador portátil Lenovo Ideapad Z500 con procesador Intel I7 2.20GHz con 8 Gbits de memoria RAM. En él se ha instalado la plataforma JdeRobot que incluye todos los componentes usados como servidores de imágenes.

En este ordenador portátil también se ha instalado la versión de Ruby 2.0 y la versión de Rails 4.0.

Aunque para el desarrollo de la aplicación se usó el servidor que viene integrado en Rails, el servidor Webrick, todo el capítulo de pruebas está realizado con el servidor Apache2.

5.1.3. Cliente Web

Como cliente web para este proyecto se ha usado un ordenador portátil Apple Macbook 13' 2.00 GHz con 4 Gbits de memoria RAM. En él se instalaron los navegadores Chrome, Firefox y Safari para poder probar las prestaciones de cada uno de ellos. Los resultados se recogen en el capítulo de Pruebas. Además se probaron otros navegadores en dispositivos móviles y tabletas.

5.2. Prueba con todos los elementos dentro de una red local

La primera prueba se realiza en una subred local. Todas las máquinas se conectan entre sí usando un *switch* Fast Ethernet. En esta subred se tomará como ancho de banda el máximo permitido por el cable. Al estar usando un cable Fast Ethernet, el máximo ancho de banda permitido por dicho estándar son 100 Mbits por segundo.

Con esta primera prueba se pretende medir las prestaciones del sistema en un entorno ideal con conexiones de alto ancho de banda y sin latencia apreciable. La figura 5.4 representa un esquema del montaje realizado.

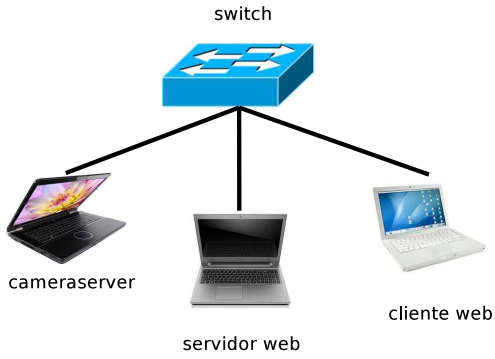

Figura 5.4: *Montaje para la prueba con todos los elementos en una misma subred*

El resultado obtenido en esta prueba es un *streaming* con una tasa de refresco de imagen de alrededor de 12 fotogramas por segundo y 500 ms de latencia.

5.2.1. Caracterización de los anchos de banda necesarios

Se quiere caracterizar ahora los anchos de banda necesarios entre las distintas partes de el sistema. Para ello, se parte del montaje descrito anteriormente y se van limitando los

anchos de banda disponibles entre partes del sistema. Estos límites se consiguen gracias al software `Wondershaper`. Este programa limita tanto el ancho de banda de subida como el de bajada de cada máquina.

Siguiendo este procedimiento, se caracteriza la conexión entre el servidor y el cliente web. Los resultados se muestran en la tabla de la figura 5.5

Ancho de banda (Mbps)	Refresco de la imgen (fps)
2	6
3	8
4	11
4,5	12

Figura 5.5: *Resultados de la caracteriación del ancho de banda entre servidor y cliente web*

La caracterización del ancho de banda entre `cameraserver` y el servidor web no es concluyente. El problema de caracterizar este ancho de banda es que si éste no es suficiente para la tasa de refresco nominal (12 fps) el sistema de autoajuste cree que debe ir más rápido en las peticiones ICE. Este comportamiento empeora la situación ya que se tienen más peticiones ICE por segundo en un ancho de banda ya de por sí congestionado.

Dados los resultados inconcluyentes de las pruebas, se calcula el ancho de banda necesario de forma teórica para esta conexión:

$$Mbps = \text{fps}\ x\ \text{resolución de la imagen}\ x\ \text{número de colores por píxel}\ x\ \text{nº de bits por color.} \tag{5.1}$$

Esto es:

$$12\ x\ 320\ x\ 240\ x\ 3\ x\ 8\ =\ 22{,}11 Mbps. \tag{5.2}$$

Por lo tanto, si tenemos en cuenta las cabeceras y el flujo de control, el ancho de banda mínimo disponible entre `cameraserver` y el servidor Web es de aproximadamente 25 Mbps.

5.3. Prueba con un ancho de banda medio entre `cameraserver` y el servidor Web

La segunda prueba se realiza con `cameraserver` y el servidor web dentro de la misma red local de la `Universidad` y el navegador en un domcilio.

Con esta prueba se pretende demostrar que el ancho de banda de bajada de un hogar convencional es suficiente para que el *streaming* funcione correctamente. Al tener `cameraserver` y el servidor web en la `Universidad` podemos asumir que las limitaciones de ancho de banda las pone el `domicilio 1`.

La figura 5.6 muestra un esquema del montaje realizado.

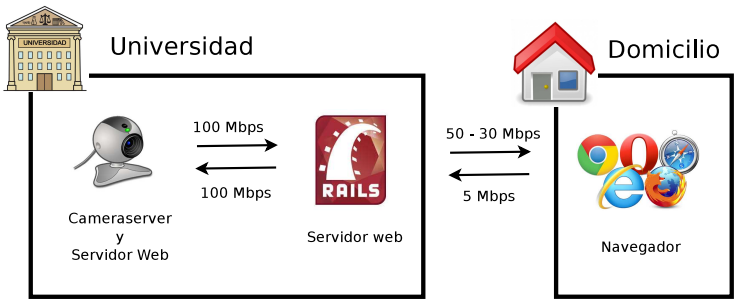

Figura 5.6: *Montaje para la prueba con un ancho de banda medio entre* `cameraserver` *y el servidor Web*

Los resultados de esta prueba son satisfactorios. Se consiguen tasas de refresco de imagen y latencia iguales a las de la prueba anterior con todos lo elementos dentro de la misma subred, unos 12 fps y 500ms de latencia..

Con esta prueba se puede concluir que el sistema es robusto siempre que haya un ancho de banda suficiente entre las distintas partes del sistema.

5.4. Prueba con un ancho de banda bajo entre `cameraserver` y el servidor Web

En la tercera prueba se quiere comprobar el funcionamiento del sistema si se coloca el componente ICE en un domicilio. Esta prueba resulta crítica ya que se intentará enviar una cantidad de datos muy grande por la conexión de subida de un domicilio convencional.

El `domicilio 1` tiene una capacidad de subida de 5 Mbps. Con el cálculo del apartado de caracterización de ancho de banda, se puede predecir a priori que el sistema no funcionará correctamente. La figura 5.7 muestra un esquema del montaje realizado.

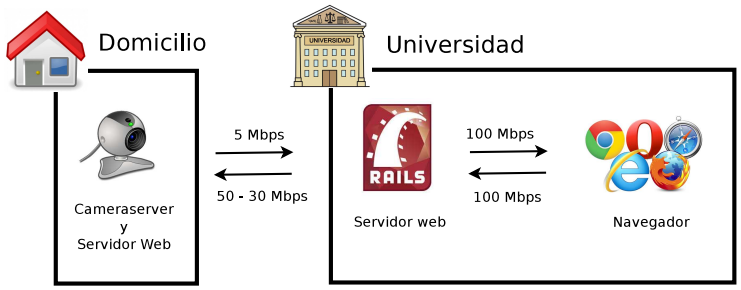

Figura 5.7: *Montaje para la prueba con un ancho de banda bajo entre* `cameraserver` *y el servidor Web*

El resultado de esta prueba no es satisfactorio. Como era de esperar la conexión entre *cameraserver* y el servidor web se satura. Se consigue un refresco de imágenes muy bajo, de menos de una imagen por segundo y una latencia de varios segundos.

5.5. Prueba con un ancho de medio entre todas las partes del sistema

Dados los resultados de la prueba anterior, se pretende demostrar con esta prueba que el sistema funciona siempre y cuando haya suficiente ancho de banda entre `cameraserver` y el servidor web. En esta prueba se ha colocado `cameraserver` en la `Universidad`, el servidor web en el `domicilio 1` y el navegador en el `domicilio 2`. La figura 5.8 muestra un esquema del montaje realizado.

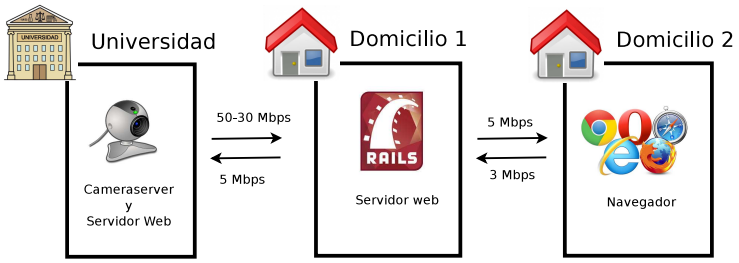

Figura 5.8: *Montaje para la prueba con un ancho de medio entre todas las partes del sistema*

El resultado de esta prueba es satisfactorio. Se recibe en el receptor un *streaming* con una calidad aceptable. La tasa de refresco de imágenes es de 12 *frames* por segundo y una latencia del orden de 500 ms.

5.6. Prueba con un teléfono móvil conectado con conexión 3G como cliente web

Con esta prueba se pretende demostrar que el sistema funciona en dispositivos móviles. Éste es el escenario típico de un sistema realista: Se accede a los sensores y actuadores de la casa desde un teléfono móvil desde, por ejemplo, desde el lugar de vacaciones.

En este montaje `cameraserver` y el servidor web están en el `domicilio 1`. El navegador es un teléfono móvil Samsung Galaxy S2. En la figura 5.9 se muestra un esquema del montaje de esta prueba.

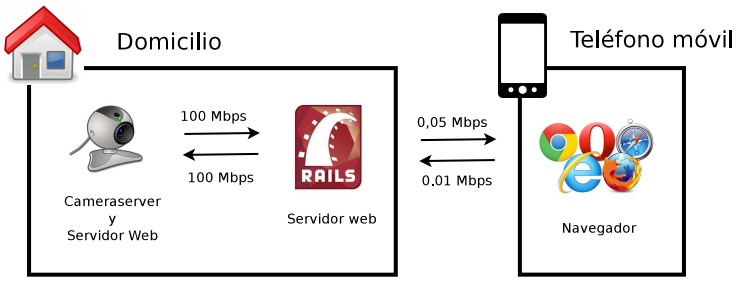

Figura 5.9: *Montaje para la prueba con el navegador web en un teléfono móvil*

Dado el ancho de banda que ofrece la conexión 3G, el refresco del flujo de vídeo no es tan alto como en otras pruebas. En este escenario se tiene un refresco de entre 6 y 8 imágenes por segundo. En la figura 5.10 se puede ver una captura de pantalla del navegador del móvil con el sistema en funcionamiento.

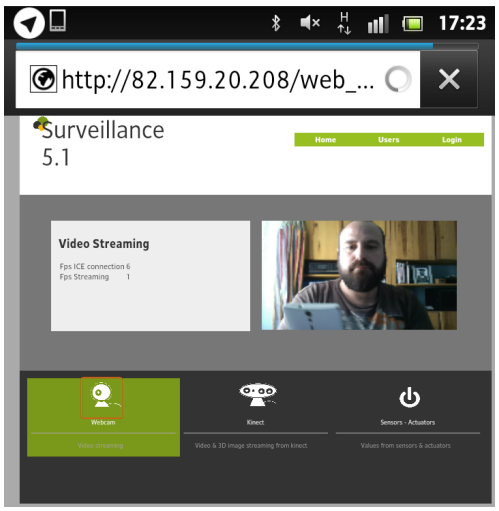

Figura 5.10: *Captura de pantalla del navegador de un dispositivo móvil*

5.7. Resumen de las pruebas

Surveillance 5.1 es un sistema robusto ya que funciona bien en situaciones reales. Se ha podido comprobar que la restricción principal tiene que ver con el ancho de banda entre `cameraserver` y el servidor web. Esto es lógico ya que `cameraserver` envía las imágenes sin comprimir y por lo tanto usa un gran ancho de banda. La tabla 5.11 resume las pruebas realizadas.

	CAMERASERVER		SERVIDOR WEB		NAVEGADOR		FPS	LATENCIA
	Ubicación	Mbps Subida / Bajada	Ubicación	Mbps Subida / Bajada	Ubicación	Mbps Subida / Bajada		
PRUEBA 1	Red local	100/100	Red local	100	Red local	100	12	500 ms
PRUEBA 2	Universidad	100/100	Domicilio 1	50/5	Domicilio 1	50/5	12	500 ms
PRUEBA 3	Domicilio 1	50/5	Universidad	100/100	Universidad	100/100	1	2 s
PRUEBA 4	Universidad	100/100	Domicilio 1	50/5	Domicilio 2	30/3	2	500 ms
PRUEBA 5	Domicilio 1	100/100	Domicilio 1	50/5	Teléfono móvil	3/0,3	8	1 s

Figura 5.11: *Tabla comparativa con los resultados de las distintas pruebas realizadas*

Capítulo 6

Conclusiones

En los capítulos anteriores se ha hecho una descripción del problema y se ha presentado la solución desarrollada. Además, se ha argumentado la elección final de diseño y se han presentado las pruebas realizadas. En este capítulo se analizarán las conclusiones a la luz del trabajo realizado y se proponen posibles lineas futuras de desarrollo.

6.1. Conclusiones

Tras analizar el trabajo realizado se puede apreciar que con Surveillance 5.1 se ha conseguido el objetivo general. Se ha logrado una aplicación web de interfaz sencilla y atractiva que puede integrar fácilmente cualquier componente desarrollado que implemente una interfaz ICE para acceder a diferentes sensores y actuadores. El diseño elegido consta de tres bloques: servidores ICE de sensores y actuadores, servidor web con Rails y cliente Web.

Además, se han conseguido todos los subobjetivos:

- Se ha realizado un flujo de vídeo por medio del envío consecutivo de imágenes. Estas imágenes se obtienen a través de una interfaz ICE que se conecta al componente de JdeRobot `cameraserver`, que se emplea en numerosas aplicaciones. Todas las funcionalidades del sistema han sido integradas dentro del servidor web sin utilizar ningún software paralelo.

  Se han implementado hilos de ejecución dentro del propio servidor Rails para incrementar la tasa de refresco del vídeo con continuas peticiones asíncronas.

Como mejora adicional también se ha conseguido procesar la imagen dentro del servidor web, para comprimirla a PNG con el consiguiente ahorro de ancho de banda en la transmisión al cliente web.

Se ha usado tecnología AJAX para el refresco de los valores de los sensores desde el cliente al servidor web. De esta forma, se ahorra ancho de banda en la comunicación entre ellos y se mejora la tasa de refresco de los flujos multimedia.

- Se ha programado un flujo con la imagen de profundidad que ofrece en sensor Kinect de Microsoft. Estas imágenes de profundidad se consiguen mediante peticiones ICE síncronas al componente de JdeRobot `openni1Server`, que hace de driver. Este flujo de imágenes de profundidad se ha implementado usando una tecnología distinta al flujo de vídeo, entre otras razones, para explorar todas las posibilidades actuales que nos ofrece el desarrollo de aplicaciones web y el nuevo estándar HTML5.

  Se han realizado gráficos 3D en tiempo real a partir de las imágenes de profundidad en el navegador. Para ello, gracias a WebGL, se ha usado la aceleración gráfica de la tarjeta de vídeo. Además, se ha incluido un movimiento de cámara de observación de la escena 3D que hace la aplicación más atractiva y llamativa.

- Se ha conseguido recibir alarmas de un sensor de humedad. Estas alarmas se reciben a través de una conexión ICE con el componente `GPIOServer` desarrollado específicamente en este proyecto. Igualmente, se ha conseguido acceder a un actuador para encender y apagar una bombilla desde la aplicación web usando el mismo componente `GPIOServer` que corre en un Raspberry Pi

Los requisitos que se especificaban en el capítulo 2 también están satisfechos.

El primer requisito era que la aplicación fuese fácilmente escalable. Este requisito está satisfecho ya que en Surveillance 5.1 se puede integrar cualquier componente de JdeRobot que implemente una interfaz ICE.

En el capítulo 2 se especificaba que el flujo debía tener una tasa de refresco suficientemente alta como para dar sensación de movimiento. Se ha conseguido llegar a tasas de refresco de unas 12 imágenes por segundo, lo que proporciona una sensación de movimiento fluido. Desde un teléfono móvil con conexión 3G a Internet se han conseguido 8 fotogramas por segundo.

Se señalaba además que la aplicación debía ser intuitiva y fácil de manejar. Todo el desarrollo de la apariencia y estructura de la aplicación se ha hecho pensando en estos principios según se muestra en el capítulo 4.

Surveillance 5.1 es una última aproximación a un posible sistema domótico comercial de bajo coste. Esta nueva versión se diferencia de la versión 4.0 principalmente en que el propio servidor web crea conexiones ICE con los componentes para obtener los datos de los sensores. Mientras que la versión antigua usaba un software de terceros para el flujo de vídeo, la version presentada en esta memoria lo hace conectándose directamente al componente `cameraserver`.

Otra diferencia es el manejo de los sensores y actuadores: mientras que en la versión antigua se guardaban los valores en una base de datos para que posteriormente el servidor los leyera, en la versión actual, se toman esos valores directamente de `GPIOServer` mediante ICE.

Además, la nueva versión introduce flujo de imágenes de profundidad y representación de puntos en 3D.

La aportación principal de este trabajo más allá de la aplicación aquí presentada, es la integración de la tecnología usada por el proyecto JdeRobot en el entorno de las aplicaciones web.

Mediante la tecnología presentada en esta memoria, las interfaces ICE de cualquier componente desarrollado por este grupo pueden, de ahora en adelante, conectarse a una aplicación web con todas las ventajas que esto ofrece (multiplataforma, multilenguaje, no se necesita instalación ni actualización en el cliente, etc).

También se ha abierto una vía de investigación en cuanto a dibujo en 3D en la web. Hasta ahora todo lo que había desarrollado este grupo estaba hecho con aplicaciones de escritorio como OpenGL, GTK o QT.

Los conocimientos adquiridos durante la realización de este trabajo son muy numerosos debido a su carácter heterogéneo y las múltiples tecnologías manejadas.

- Desde el punto de vista de programación, se ha aprendido Ruby como lenguaje de programación y se ha usado Ruby sobre Rails como entorno de desarrollo de aplicaciones web. Se han aplicado conocimientos sobre Javascript y ampliado conocimientos sobre JQuery, JSON y AJAX.
- Se han adquirido conocimientos sobre el middelware ICE y se ha investigado su integración en aplicaciones Web.
- También se ha aprendido a representar figuras en tres dimensiones con WebGL.

- Se han asentado y ampliado conocimientos del nuevo estándar HTML5 y de sus nuevos elementos, como por ejemplo la etiqueta *canvas*.
- Se han adquirido conocimientos sobre el Hardware Raspberry Pi, su sistema operativo y el control de sus puertos GPIO.
- Se han adquirido los conocimientos necesarios sobre el sistema de composición de textos LaTeX para redactar esta memoria.

6.2. Trabajos futuros

Este TFG sienta las bases para posteriores trabajos en el ámbito de la domótica o robótica.

Por un lado, se puede seguir trabajando en el sistema domótico Surveillance implementando nuevas herramientas y funcionalidades hasta conseguir un sistema completo y de bajo coste para comercializar. Por ejemplo, la versión 4.0 desarrolló un componente llamado `zbserver` para la recepción de sensores mediante protocolo Zigbee. Un posible avance sería incluir una interfaz ICE para comunicarse con este componente.

Por otro lado, a la vista de las pruebas realizadas en el capítulo anterior, se pone de manifiesto la limitación de ancho de banda que existe entre `cameraserver` y el servidor web. Una posible mejora sería que `cameraserver` envíe las imágenes al servidor ya codificadas con algún tipo de compresión.

Además, se puede investigar sobre cómo conseguir un *streaming* de vídeo más fluido y de mayor resolución. Esto se puede conseguir aplicando un preprocesado a las imágenes que consiga codificar el flujo de vídeo de forma eficiente, consiguiendo un *streaming* de calidad que requiera poco ancho de banda.

Dado que ya existen dentro del grupo de robótica algoritmos de inteligencia dedicados, por ejemplo al seguimiento y detección de caídas de personas mediante el sensor Kinect, se ve muy viable la integración de este tipo de inteligencia en la aplicación web.

Otro avance muy interesante sería que el propio navegador implementara el middleware ICE para JavaScript y se conectara directamente con los componentes de sensores y actuadores.

Por último, se podría investigar en lo que puede aportar la tecnología WebRTC al proyecto Surveillance.

Bibliografía

[1] ¿Qué es el Diseño Universal? http://www.ciudadaccesible.cl/que-es-el-diseno-universal/ Consultado en Junio de 2014.

[2] Proyecto JdeRobot http://jderobot.org/ Consultado en Junio de 2014.

[3] Página oficial WebGL http://get.webgl.org/ Consultado en Junio de 2014.

[4] Página oficial WiringPi http://wiringpi.com/ Consultado en Junio de 2014.

[5] Rails Guideshttp://guides.rubyonrails.org/action_controller_overview.html Consultado en Junio de 2014.

[6] Página Chunky_png en RubyGems https://rubygems.org/gems/chunky_png Consultado en Junio de 2014.

[7] Herramienta web SpeedTest http://www.speedtest.net/es/ Consultado en Junio de 2014.

[8] Mediawiki Daniel Castellano (Surveillance 4.0) http://jderobot.org/D.castellanob-pfc Consultado en Junio de 2014.

[9] Home automation system with web interface in the JdeRobot frameworkhttp://gsyc.es/jmplaza/papers/waf2014-surveillance4.pd Consultado en Junio de 2014. f

[10] Página oficial Nest https://nest.com/ Consultado en Junio de 2014.

[11] Página oficial Zigbee http://www.zigbee.org/ Consultado en Junio de 2014.

[12] Mediawiki Edgar Barrero (Surveillance 5.1) http://jderobot.org/Aerobeat-colab Consultado en Junio de 2014.

[13] Repositorio Edgar Barrero (Surveillance 5.1) `https://svn.jderobot.org/users/aerobeat/colab/` Consultado en Junio de 2014.

[14] Página oficial Axicom `http://www.axicom.com/` Consultado en Junio de 2014.

[15] Página oficial Ruby `https://www.ruby-lang.org/es/` Consultado en Junio de 2014.

[16] Página oficial Ruby on Rails`http://rubyonrails.org/` Consultado en Junio de 2014.

[17] Página oficial ICE`http://www.zeroc.com/` Consultado en Junio de 2014.

[18] Página oficial Raspberry Pi `http://www.raspberrypi.org/` Consultado en Junio de 2014.

[19] Componente `cameraserver` JdeRobot `http://jderobot.org/Interfaces#Camera` Consultado en Junio de 2014.

[20] Componente `openni1Server` JdeRobot `http://jderobot.org/index.php/Components#OpenniServer` Consultado en Junio de 2014.

[21] Página oficial GStreamer `http://gstreamer.freedesktop.org/` Consultado en Junio de 2014.

[22] Página oficial Open CV `http://opencv.org/` Consultado en Junio de 2014.

[23] Openni `http://en.wikipedia.org/wiki/OpenNI` Consultado en Junio de 2014.

[24] Sam Ruby, Dave Thomas, David Heinemeier(2013): *Agile Web Developement with Rails 4*. Dallas: The Pragmatic Bookshelf.

[25] Dave Thomas, Chad Fowler, Andy Hunt(2011): *Programming Ruby 1.9*. Dallas: The Pragmatic Bookshelf.

[26] Ice 3.5.1 Documentation

[27] ICE spanish PDF. David Vallejo Fernández

[28] Wesley Hales (2013): *HTML5 and JavaScript Web Apps*. O'Reilly Media By:

[29] Monográfico sobre el estándar X10. Historia, protocolo, implementaciones, etc. de este estándar. `https://forja.rediris.es/docman/view.php/414/761/X10.pdf`

[30] Página web del estándar KNX. `http://www.knx.org/es/` Consultado en Junio de 2014.

[31] Norma UNE-EN 50065-1:2012 para la transimisión de sañales por la red eléctrica. `http://www.aenor.es/aenor/normas/normas/fichanorma.asp?tipo=N&codigo=N0048827`

[32] Página web del la alianza empresarial para la difusión de *Z-wave*. `http://www.z-wavealliance.org/`

[33] Sistema de seguridad domótica de la empresa *Securitas Direct*. `http://www.securitasdirect.es/myverisure.html`

[34] Alarma para el hogar *Prosegur Proview +*. `http://www.alarmahogarprosegur.com/?o=web`

[35] Sistema domótico textitDelta Dore. `http://www.deltadore.com/`

[36] Página web de la compañía *Libelium*. `http://www.libelium.com/company/`

[37] Distributed Video surveillance System based on Android, Roberto Calvo Palomino, Master project, 2010. `http://jderobot.org/index.php/Rocapal_Enabling_Technologies_LS`

[38] Orellana Galloso, Rubén,*Sistema de monitorización y control de riego e iluminación basado en tecnologías Arduino, Zigbee, Android y Bluetooth para un emplazamiento en Cádiz*, España, 2013.

[39] Detección de caídas en hogar basado en cámaras y kinects. `http://jderobot.org/index.php/ElderCare`