ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN
UNIVERSIDAD POLITÉCNICA DE CARTAGENA

Proyecto Fin de Carrera

Aplicación de video-vigilancia multi-plataforma basada en tecnología JavaFX

AUTOR: Pablo Bernal Alcázar
DIRECTOR: Diego Alonso Cáceres

Julio / 2010

Autor	Pablo Bernal Alcázar
E-mail del Autor	bernalmurcia@hotmail.com
Director(es)	Diego Alonso Cáceres
E-mail del Director	diego.alonso@upct.es
Codirector(es)	
Título del PFC	Aplicación de video-vigilancia multi-plataforma basada en tecnología JavaFX
Descriptores	

Resumen

JavaFX, la nueva tecnología de Sun Microsystems para el desarrollo de aplicaciones RIA, promete dar un importante paso en la búsqueda de una mejor experiencia del usuario gracias a su capacidad multi-dispositivo que permite la manutención de una sola aplicación para los distintos entornos online, escritorio, móvil y TV.

Aprovechando esta notable innovación, el presente proyecto pretende llenar el importante vacío existente en el ámbito de la video-vigilancia al respecto de la capacidad de monitorización de las cámaras de seguridad en un escenario móvil, sin ataduras en cuanto a emplazamientos fijos.

Con este objeto, se ha creado una aplicación extensible capaz de soportar la monitorización de cámaras IP o web, utilizando la potencia sintáctica de JavaFX Script para crear una utilidad con capacidad multi-dispositivo y alta funcionalidad en materia de seguridad.

Titulación	Ingeniero Técnico de Telecomunicaciones Esp. Telemática
Intensificación	
Departamento	Tecnología de la Información y las Comunicaciones (TIC)
Fecha de Presentación	Julio - 2010

AGRADECIMIENTOS

Gracias, no sólo a los que con vuestro apoyo y ayuda me habéis permitido alcanzar esta meta, sino a todos los que habéis convertido el camino en lo mejor del viaje:

A mis padres, por darme la posibilidad de elegir mi futuro con su esfuerzo y creer en mí para conseguir lo que me proponga.

A mi hermana, por respetarme y cuidarme, por estar ahí.

A mis amigos, por hacerme disfrutar cada momento, por comprenderme y conocerme.

A Jaime, por compartir esos horizontes llenos de luz y música conmigo.

A Ana Cristina, por acompañarme a lo largo de estos 3 años y amenizar el trabajo en equipo.

A Diego, por su dedicación y ayuda en los momentos complicados de este proyecto.

A todos vosotros, y a los que en este tiempo han pasado por al lado enseñándome a ser mejor cada día, GRACIAS.

A mis padres y familia,
por los que este final es un comienzo

TABLA DE CONTENIDOS

1 Introducción 11

1.1 Objetivos del proyecto 12

1.2 Estructura del documento 13

2 Estado de la Técnica 15

2.1 Rich Internet Applications 16

2.1.1 Adobe Flash/Flex/Air 17

2.1.2 Microsoft Silverlight 17

2.1.3 AJAX 19

2.1.4 Google GWT 20

2.1.5 JavaFX 21

2.1.6 Comparativa uso de Frameworks en proyectos de Software 21

2.2 Aplicaciones video-vigilancia Desktop 22

2.2.1 uWatchIt 23

2.2.2 WebCam Surveillance Monitor 23

2.2.3 Vitamin D 23

2.3 Aplicaciones video-vigilancia Online 24

2.3.1 NetVision Premium 24

2.3.2 UGOlog 24

2.4 Aplicaciones video-vigilancia Móviles 25

2.4.1 QNAP VSMobile 25

2.4.2 Live2Phone 26

2.4.3 ConnectVu 26

3 Introducción a JavaFX 29

3.1 Componentes de la familia 30

3.1.1 JavaFX Script 30

3.1.2 JavaFX Mobile 32

3.1.3 JavaFX Production Suite 32

3.1.4 JavaFX SDK 33

3.1.5 Netbeans IDE 33

3.2 Historia y versiones de JavaFX 34

3.3 Sintaxis JavaFX Script 35

3.3.1 Variables 35

3.3.2 Tipos de datos 36
3.3.2.1 Boolean 36
3.3.2.2 Integer 36
3.3.2.3 Number 36
3.3.2.4 String 36
3.3.2.5 Duration 37
3.3.2.6 Void 37
3.3.2.7 Tipos de Datos Java 37
3.3.2.8 Pseudo-Variables 37
3.3.3 Binding 38
3.3.4 Binding bidireccional 39
3.3.5 Secuencias 39
3.3.6 Operadores 40
3.3.6.1 Operadores Aritméticos 40
3.3.6.2 Operadores de asignación 41
3.3.6.3 Operadores Unarios 41
3.3.6.4 Operadores Relacionantes 41
3.3.6.5 Operadores Lógicos 42
3.3.6.6 Operador Instanceof 42
3.3.7 Expresiones 42
3.3.7.1 Bloques 42
3.3.7.2 If 43
3.3.7.3 For 43
3.3.7.4 While 44
3.3.7.5 Break y Continue 44
3.3.8 Funciones 44
3.3.9 Clases y Objetos Literales 45
3.3.10 Modificadores de Acceso 46
3.3.11 Triggers 47
3.3.12 Timelines 47
3.3.13 "Hola Mundo" 48
3.4 Creación de la Interfaz Gráfica 49
3.4.1 Stage 49
3.4.2 Scene 50
3.4.3 Node 50

3.4.3.1 Group y Container 51
3.4.3.2 CustomNode 51
3.4.3.3 Shape 51
3.4.3.4 ImageView 51
3.4.4 Componentes de Interfaz de Usuario 52
3.5 Otras Utilidades 52
3.5.1 HttpRequest 52
3.5.2 Resource y Storage 53
4 Desarrollo del Proyecto 55
4.1 Primeros Pasos 56
4.1.1 Primeras impresiones JavaFX 56
4.1.2 Instalación JavaFX Mobile 57
4.1.3 Elección del adecuado IDE 57
4.2 Objetivos y requisitos de la aplicación 58
4.3 Evolución global del proyecto y problemas encontrados 59
4.4 La Aplicación 62
4.4.1 Núcleo de la aplicación 64
4.4.1.1 ***Main:*** perfiles de ejecución, *Stage* y *Scene* 64
4.4.1.2 ***Controller:*** Controlador de datos y escenarios gráficos 65
4.4.2 VistaPrincipal - Inicio de la Aplicación 68
4.4.3 VistaConfig - Configurar la Aplicación 69
4.4.4 VistaListado - Listado de las Cámaras 71
4.4.5 VistaMapa - Ver las Cámaras en el Mapa 71
4.4.6 VistaVisor - Monitorizar la Cámara 75
4.4.7 VistaError - Manejo de errores 78
4.4.8 Creación de la GUI 79
4.5 Software complementario 81
4.5.1 Servidor Apache 81
4.5.2 Mplayer 83
4.6 Funcionamiento de la aplicación 84
5 Conclusiones 87
Anexo A – Código Completo 91
Bibliografía 125

ÍNDICE DE FIGURAS

Figura 2.1: Tecnologías agrupadas bajo el concepto de AJAX 19
Figura 2.2: Tendencia de uso de frameworks en proyectos según SimplyHired 21
Figura 2.3: Tendencia uso de frameworks en proyectos según Indeed 22
Figura 2.4: Esquema básico sistemas de video-vigilancia 22
Figura 2.5: QNAP VSMobile 25
Figura 2.6: Live2Phone 26
Figura 2.7: Sistema ConnectVU 27
Figura 3.1: Situación tecnología JavaFX 30
Figura 3.2: Patrón Model-View-Controller 31
Figura 3.3: Jerarquía de Interfaz Gráfica 49
Figura 3.4. Jerarquía de nodos 50
Figura 4.1: Diagrama de Clases 63
Figura 4.10: Selección y carga del mapa 85
Figura 4.11: Desplazamiento y Zoom 86
Figura 4.12: Listado y Visor 86
Figura 4.2: Diagrama de Secuencia de Inicialización 64
Figura 4.3: Diagrama de secuencia de cambio a *VistaMapa* 67
Figura 4.4: Diagrama de secuencia de selección de cámara 68
Figura 4.5: Error 78
Figura 4.6: Pestañas de Interfaz Gráfica 79
Figura 4.7: Uso de *Text*, *TextBox*, *Button* y *ListView* en interfaz gráfica 81
Figura 4.8: Pantalla de bienvenida 84
Figura 4.9: Configurar IP y mapa 85

ÍNDICE DE CÓDIGOS

Código 1: Variables de perfil 65
Código 2: Uso de variables de perfil 65
Código 3: *Stage* y *Scene* 65
Código 4: Interfaz *Vista* 66
Código 5: *Contents* 66
Código 6: Pestaña para mostrar la *VistaMapa* 66
Código 7: Variable *cam_actual* en el controlador 67
Código 8: Variable *cam_actual* en la *VistaVisor* 68
Código 9: Mostrar *VistaPrincipal* 69
Código 10: Introducir IP 69
Código 11: Botón para introducir IP 69
Código 12: Almacenar información de IP 70
Código 13: Extraer información almacenada IP 70
Código 14: Petición Http de lista de mapas 71
Código 15: Listado de Cámaras 71
Código 16: Icono de cámara en el mapa 72
Código 17: Contenedor del mapa 72
Código 18: Variables para gestión de escalado y desplazamiento del mapa 73
Código 19: Zoom Out 74
Código 20: Flecha de Desplazamiento 74
Código 21: Desplazamiento basado en *dragging* 75
Código 22: Variables en *VistaVisor* 75
Código 23: Imagen para visualización 76
Código 24: *Timeline* en *VistaVisor* 76
Código 25: Función *renovarUrl* 77
Código 26: Función *showVistaVisor* en el *Controller* 77
Código 27: Función *iniciarVisor* 77
Código 28: Función setTextoCamara 77
Código 29: Función *cerrarVisor* 77
Código 30: Función *showErrorScreen* 78
Código 31: Función *volverVistaAnterior* 79
Código 32: Barra de desplazamiento 80
Código 33: Pestaña de Mapa 80
Código 34: *Hbox* contenedor de las pestañas 80
Código 35: Archivo de configuración *config.txt* 81
Código 36: Archivo de mapa *mapa0.txt* 82
Código 37: Archivo html *index.html* 82

1 Introducción

Las RIA (Rich Internet Applications) suponen una nueva generación de aplicaciones que aúnan las propiedades de las aplicaciones Web con las ventajas de las tradicionales aplicaciones de escritorio en una búsqueda por mejorar la experiencia del usuario. En este ámbito, JavaFX representa la contrapartida de Sun Microsystems para competir con grandes tecnologías ya consolidadas en este marco como son Adobe Flash o Microsoft Silverlight.

A pesar del hándicap que para JavaFX supone su retraso en la aparición en el escenario RIA, esta tardanza puede verse justificada y compensada por las grandes innovaciones y mejoras tecnológicas que este *framework* presenta. Con su novedosa sintaxis declarativa y capacidades de enlace y animación como abanderados, y con su soporte multi-dispositivo que permite la ejecución de una misma aplicación, no solo en todas las plataformas de escritorio o navegadores web existentes, sino también en dispositivos móviles e incluso TV, JavaFX presenta su candidatura al dominio de la atmósfera RIA y, por tanto, al dominio de gran parte del futuro escenario web.

Por otra parte, el ámbito de la video-vigilancia presenta un amplio número de posibilidades de desarrollo para todo tipo de utilidades software. A pesar de la gran cantidad de aplicaciones de video-vigilancia existentes en el estado actual de la técnica, ninguna presenta las características que se darían en un enlace entre la nueva tecnología de Sun y el desarrollo de software dedicado al respecto.

Con este objeto, se ha creado una aplicación capaz de soportar la monitorización de cámaras IP o web desde el escritorio de las actuales plataformas de sobremesa, desde cualquier navegador web para su ejecución online sin necesidad de instalación previa y desde un dispositivo móvil que se transformaría en todo un punto de vigilancia sin ataduras de emplazamiento. Todo ello utilizando la potencia sintáctica JavaFX Script para un resultado reutilizable con alta conectividad web y multimedia que además resulte gráficamente potente.

1.1 Objetivos del proyecto

En su línea más básica, este proyecto supone un estudio de las capacidades reales de la tecnología JavaFX, comparándolas con el resto de *frameworks* existentes en el estado actual de la técnica, para comprobar el auténtico alcance que esta familia tecnológica pueda tener en la escena de desarrollo de RIA.

Por otra parte, en materia de video-vigilancia, este trabajo busca completar un vacío existente en cuanto a la necesidad de una aplicación capaz de ejecutarse en un escenario móvil que permita a los vigilantes de seguridad controlar las imágenes de las cámaras a su disposición mientras hacen una ronda de vigilancia. De forma que, sin la existencia de ningún tipo de ataduras de emplazamiento, estos vigilantes de seguridad puedan mantener su control sobre toda una instalación sin necesidad de encontrarse en un lugar determinado.

Desde el punto de vista del desarrollo, este proyecto tendrá como objetivo principal la creación de una aplicación de video-vigilancia que, cumpliendo los objetivos anteriores de evaluación de JavaFX y capacidad de ejecución móvil, mantenga una serie de requisitos básicos obtenidos del estudio de las principales utilidades de video-vigilancia existentes en el estado actual de la técnica.

Considerando todo la anterior, los requisitos y objetivos de la aplicación final se resumen en:

- Soporte para cámaras web y cámaras IP.
- No instalación de software.
- Manutención de una sola aplicación para los distintos entornos de ejecución.
- Soporte Desktop multi-plataforma.
- Monitorización en dispositivo móvil.
- Monitorización online.
- Capacidad para soportar varias cámaras.
- Mapa de cámaras.
- Adaptabilidad respecto a una posible utilización en circunstancias variables de uso de tecnologías de vídeo o mapas físicos.
- Capacidad para ser configurable por un usuario local o de forma remota a través de un servidor.

1.2 Estructura del documento

El presente documento se divide en varios capítulos temáticos para recoger las bases del estudio realizado al caso de la familia tecnológica JavaFX, así como del estado actual de la técnica en cuanto a aplicaciones RIA y de video-vigilancia. Todo completado con la creación de una aplicación multiplataforma capaz de monitorizar imágenes desde cámaras web o IP a través de un servidor configurado a tal efecto. La descripción del estudio se subdivide en:

- **Introducción:** una presentación al grueso del estudio.
- **Estado de la Técnica:** un resumen básico de los *frameworks* existentes para el desarrollo de aplicaciones RIA y de las utilidades actuales para la monitorización de seguridad y video-vigilancia.
- **Introducción a JavaFX:** una completa descripción de la familia tecnológica JavaFX que nos acercará a la comprensión de sus verdaderas capacidades y de las herramientas que presenta para ello. Los elementos que la convierten en una plataforma multi-dispositivo, la sintaxis declarativa JavaFX Script y un repaso por su historia.
- **Desarrollo de la Aplicación:** la completa descripción de la aplicación desarrollada, partiendo desde los objetivos que la fundamentan y la línea temporal que la ha movido desde un inicio y llegando a la exposición de la solución implementada y un ejemplo de funcionamiento.
- **Conclusiones:** tras el estudio teórico y el desarrollo de una aplicación práctica se presentan una serie de conclusiones generales sobre la tecnología JavaFX y su aplicación en el campo de la video-vigilancia. Así como futuros trabajos que puedan tomar como base los elementos que se han estudiado y trabajado en este proyecto.
- **Anexos y bibliografía:** un anexo que recoge la totalidad del código empleado para la construcción de la aplicación y las referencias bibliográficas que se han utilizado a lo largo del estudio.

2 Estado de la Técnica

En este capítulo se recogen y se describen las diferentes tecnologías que rodean el entorno de este proyecto. El concepto RIA de aplicaciones supone un importante paso para la experiencia web del usuario, un paso que podría ser definitivo para un posible cambio desde el escritorio a la web. Es importante conocer los frameworks para el desarrollo de aplicaciones RIA que compiten con JavaFX en este mercado, dominado en sus inicios por un Adobe Flash que cada vez cede más terreno ante sus competidores Microsoft Silverlight y Google GWT.

Este capítulo proporciona una visión global de lo que significa la creación de una RIA desde los distintos entornos de desarrollo disponibles y más usados en el mercado, así como las diferencias y novedades que la tecnología JavaFX puede aportar en este sentido.

En cuanto a la problemática de la video-vigilancia, encontramos varias aplicaciones que tratan de abordar el tema desde los diversos escenarios: online, escritorio y móvil. Soluciones profesionales como NetVision Premium o VSMobile suponen herramientas muy potentes y funcionales, pero requieren de grandes desembolsos de dinero a causa de la necesidad de apoyarse en hardware exclusivo para su sistema. Otras soluciones más económicas, con versiones Free o de bajo coste como uWatchIt o Live2Phone pueden suponer un mayor ejemplo de aplicación sencilla y funcional de video-vigilancia, pero siempre se hallan atadas a un escenario o plataforma concreta.

El estudio de las distintas aplicaciones de video-vigilancia disponibles online y para escritorio y móvil aportará un mayor conocimiento sobre este objetivo y sobre los requisitos que deben ser satisfechos por este tipo de aplicaciones, además de justificar la necesidad de la creación de una aplicación que, cumpliendo estos requisitos, sea capaz de ejecutarse tanto en escritorio como en navegador web y dispositivos móviles, todo con la ventaja añadida que supone el hecho de crear y mantener una sola aplicación para todos estos escenarios.

2.1 Rich Internet Applications

RIA, acrónimo de Rich Internet Applications (Aplicaciones de Internet Enriquecidas) fue concebido en 2002 por Jeremy Allaire de Macromedia, ahora Adobe. Las RIA suponen una nueva generación de aplicaciones que aúnan las propiedades de las tradicionales aplicaciones de escritorio con las propiedades de las aplicaciones Web en una búsqueda por mejorar la experiencia del usuario. La siguiente información sobre las RIA puede ser ampliada en [Montero, 2009], [Rapoza, 2008] y [Stamos, 2008].

En contraposición con las aplicaciones Web tradicionales, en los entornos RIA, no se producen recargas de página, ya que desde el principio se carga toda la aplicación, y sólo se produce comunicación con el servidor cuando se necesitan datos externos como datos de una Base de Datos o de otros ficheros externos. De esta forma, se evita una recarga continua de páginas cada vez que el usuario pulsa sobre un enlace. Una característica habitual en las aplicaciones Web y que produce un tráfico muy alto entre el cliente y el servidor, llegando muchas veces, a recargar la misma página con un mínimo cambio. A pesar de que el desarrollo de aplicaciones multimedia para navegadores web está mucho más limitado y es más difícil que otro tipo de aplicaciones de escritorio, los esfuerzos se justifican por varios motivos:

- No necesitan instalación (solo es necesario mantener actualizado el navegador web), por lo que son más fáciles de mantener y reducen los procesos y tiempos de ejecución.
- Las actualizaciones hacia nuevas versiones son automáticas.
- Se pueden utilizar desde cualquier ordenador con una conexión a Internet sin depender del sistema operativo que este utilice.
- Más capacidad de respuesta, ya que el usuario interactúa directamente con el servidor, sin necesidad de recargar la página y utilizando una comunicación asíncrona.
- Ofrecen aplicaciones interactivas que no se pueden obtener utilizando solo HTML, incluyendo arrastrar y pegar, cálculos en el lado del cliente sin la necesidad de enviar la información al servidor.
- Evita la problemática del uso de diferentes navegadores al abstraerse de ellos a través de un framework.

Con todo, los usuarios finales son los que experimentan las grandes ventajas de las RIA. Cabe destacar que una RIA bien estructurada y diseñada disminuye el tiempo en que el usuario encuentre la información que necesita sin esperar a que una página nueva se cargue. Podemos afirmar que una RIA combina la rica interactividad de una aplicación de escritorio, con el amplio alcance de una aplicación Web, utilizando un "navegador web" estandarizado para ejecutarse y agregando las características adicionales por medio de un plugin o independientemente una virtual machine o sandbox.

Entre los diversos frameworks existentes para el desarrollo de RIA, destacan Adobe Air, Microsoft Silverlight, Google GWS, AJAX y Sun JavaFX. Por lo que a continuación se introducirá cada uno de ellos con una descripción general de sus características.

2.1.1 Adobe Flash/Flex/Air

Adobe Flash es una herramienta creada por Macromedia (después adquirida por Adobe) en 1996 que, aproximándose al concepto RIA fue concebida para permitir la inclusión de animaciones vectoriales, sonido e interacción con el usuario dentro de una página web. En la actualidad, se ha convertido en una herramienta muy potente para la realización de interfaces de usuario muy completas, siendo considerada la más relevante y puntera de las tecnologías que permiten la creación de aplicaciones RIA, teniendo como principal inconveniente un entorno de desarrollo privativo y de pago (existe una versión reducida gratuita) y la necesidad de instalación de un plugin en el cliente.

Flash, en sus últimas versiones hace uso del lenguaje ActionScript para dar a los desarrolladores la posibilidad de programar el comportamiento de su producto mediante un completo lenguaje de scripting. Este lenguaje permite la integración con diversas fuentes de datos externas a través de los estándares más importantes de internet (XML, Webservices, etc.). A pesar de brindar esta serie de facilidades, Flash ha sido durante muchos años un entorno hostil para los desarrolladores de software familiarizados al desarrollo en lenguajes orientados a objetos ya que basa su desarrollo en una línea de tiempo.

Adobe Flex se presenta como una novedad incorporando a la tecnología Flash la posibilidad de programar en forma declarativa u orientada a objetos permitiendo a través de una liberaría de controles de datos (grillas, listas, campos de texto, paneles, solapas, etc.) la construcción de aplicaciones de características similares a las desarrolladas en entornos orientados a objetos. Sin embargo, debido a los servicios que este framework requiere, las aplicaciones desarrolladas en Flex pueden presentar una importante disminución en su rendimiento respecto a aplicaciones de similares características desarrolladas en Flash.

Adobe Air supone un entorno de ejecución versátil que permite desarrollar y ejecutar las RIA como aplicaciones de escritorio. Manteniendo todo el alcance web de este tipo de aplicaciones y añadiendo soporte para la instalación de las mismas en los diversos sistemas operativos e incluso dando acceso a los sistemas de archivos para gestionar mejor el almacenamiento de la información. Siendo un entorno gratuito, permite desarrollo y ejecución, no sólo a aplicaciones Flash o Flex, sino también AJAX, aprovechando los conocimientos de los desarrolladores a la hora de crear aplicaciones web para desarrollar aplicaciones multimedia para el escritorio, mejorando el rendimiento de estos sistemas a través de una serie de herramientas exclusivas.

Para más información, consultar [Web Adobe Flash], [Web Adobe Flex] y [Web Adobe Air].

2.1.2 Microsoft Silverlight

Microsoft Silverlight es, según la biblioteca en español de la web oficial [Web Silverlight], una implementación multiplataforma de .NET Framework que se puede ejecutar en distintos exploradores para crear y proporcionar la nueva generación de experiencias multimedia y aplicaciones interactivas enriquecidas para la Web. Silverlight unifica las funciones del servidor, la Web y el escritorio, del código administrado y de los lenguajes dinámicos, de la programación declarativa y la tradicional, así como la eficacia de Windows Presentation Foundation (WPF).

Los desarrolladores web y diseñadores de gráficos pueden crear aplicaciones basadas en Silverlight de diversas maneras. Se puede utilizar el marcado de Silverlight para crear elementos multimedia y gráficos y manipularlos con lenguajes dinámicos y código administrado. Silverlight también permite utilizar herramientas de calidad profesional, como Visual Studio para la codificación y Microsoft Expression Blend para la disposición y el diseño gráfico.

Silverlight combina varias tecnologías en una sola plataforma de desarrollo que permite seleccionar las herramientas y el lenguaje de programación apropiados según las necesidades del usuario. Silverlight ofrece las características siguientes:

- WPF y XAML. Silverlight incluye un subconjunto de la tecnología Windows Presentation Foundation (WPF), que extiende en gran medida los elementos en el explorador para crear la interfaz de usuario. WPF permite crear gráficos, animaciones y elementos multimedia fascinantes, así como otras características de cliente enriquecidas, extendiendo la interfaz de usuario basada en explorador más allá de lo que está disponible únicamente con HTML. El Lenguaje XAML proporciona una sintaxis de marcado declarativa para crear elementos.
- Extensiones a JavaScript. Silverlight proporciona extensiones al lenguaje de scripting de explorador universal que permiten controlar la interfaz de usuario del explorador, incluida la capacidad para trabajar con elementos WPF.
- Compatibilidad con varios exploradores y plataformas. Silverlight se ejecuta de la misma manera en todos los exploradores conocidos (y en las plataformas conocidas). Es posible diseñar y desarrollar aplicaciones sin tener que preocuparse del explorador o de la plataforma de los usuarios.
- Integración con aplicaciones existentes. Silverlight se integra perfectamente con el código JavaScript y ASP.NET AJAX existente de modo que complementa la funcionalidad ya creada.
- Acceso al modelo de programación de .NET Framework y a las herramientas asociadas. Se pueden crear aplicaciones basadas en Silverlight mediante lenguajes dinámicos, como IronPython, y lenguajes como C# y Visual Basic. Se pueden utilizar herramientas de desarrollo como Visual Studio para crear aplicaciones basadas en Silverlight.
- Compatibilidad de red. Silverlight incluye compatibilidad con HTTP sobre TCP. Se puede conectar a los servicios WCF, SOAP o ASP.NET AJAX y recibir datos XML, JSON o RSS.
- LINQ. Silverlight incluye Language Integrated Query (LINQ), que permite programar el acceso a datos utilizando una sintaxis nativa intuitiva y objetos con establecimiento inflexible de tipos en los lenguajes de .NET Framework.
- Las aplicaciones basadas en Silverlight se ejecutan en el explorador. Silverlight garantiza que las aplicaciones se pueden ejecutar en todos los exploradores modernos, sin tener que crear código específico del explorador.
- Para ejecutar una aplicación basada en Silverlight, los usuarios requieren un pequeño complemento en su explorador.

Se pueden crear aplicaciones basadas en Silverlight mediante cualquier lenguaje compatible con .NET Framework (incluidos Visual Basic, C# y JavaScript). Visual Studio 2008 y Expression Blend admiten el desarrollo de aplicaciones basadas en Silverlight. Con Silverlight, se pueden crear páginas web con elementos HTML y WPF. Al igual que HTML, XAML permite crear la interfaz de usuario de las aplicaciones basadas en web mediante una sintaxis declarativa, pero XAML proporciona elementos mucho más eficaces.

2.1.3 AJAX

AJAX no es una tecnología en sí mismo. En realidad, se trata de varias tecnologías independientes que se unen para dar una nueva experiencia al usuario (ver figura 2.1), el término fue acuñado en 2005 por Jesse James Garrett y es un acrónimo de Asynchronous JavaScript + *XML*.

Las tecnologías que forman AJAX son:

- XHTML y CSS, para crear una presentación basada en estándares.
- DOM, para la interacción y manipulación dinámica de la presentación.
- XML, XSLT y JSON, para el intercambio y la manipulación de información.
- XMLHttpRequest, para el intercambio asíncrono de información.
- JavaScript, para unir todas las demás tecnologías.

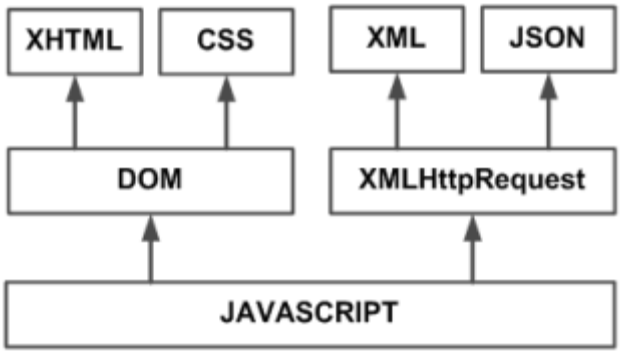


Figura 2.1: Tecnologías agrupadas bajo el concepto de AJAX

Desarrollar aplicaciones AJAX requiere un conocimiento avanzado de todas y cada una de las tecnologías anteriores, lo que constituye un objetivo bastante complejo.

Las aplicaciones construidas con AJAX eliminan la recarga constante de páginas mediante la creación de un elemento intermedio entre el usuario y el servidor. La nueva capa intermedia de AJAX mejora la respuesta de la aplicación, ya que el usuario nunca se encuentra con una ventana del navegador vacía esperando la respuesta del servidor. Las peticiones HTTP al servidor de las aplicaciones web tradicionales se sustituyen por peticiones JavaScript que se realizan al elemento encargado de AJAX. Las peticiones más simples no requieren intervención del servidor, por lo que la respuesta es inmediata. Si la interacción requiere una respuesta del servidor, la petición se realiza de forma asíncrona mediante AJAX. En este caso, la interacción del usuario tampoco se ve interrumpida por recargas de página o largas esperas por la respuesta del servidor. Se dispone de información más detallada sobre AJAX en [Eguíluz, 2008].

2.1.4 Google GWT

Google Web Toolkit (GWT) [Web GWT] permite crear aplicaciones AJAX en el lenguaje de programación Java que son compiladas posteriormente por GWT en código JavaScript ejecutable optimizado que funciona automáticamente en los principales navegadores. Durante el desarrollo de una aplicación, es posible repetir rápidamente el mismo ciclo "editar - actualizar - ver" típico de JavaScript y aprovechar la ventaja añadida de poder depurar y recorrer una a una todas las líneas de código Java. Una vez la aplicación se encuentre preparada para la implementación, GWT compilará el código fuente Java en archivos JavaScript optimizados independientes. Google Web Toolkit permite crear fácilmente tanto una sección animada para una página web como una aplicación completa.

A diferencia de los minimizadores de JavaScript, que solo funcionan con texto, el compilador de GWT realiza optimizaciones y un análisis estático completo de toda la base de código de GWT y, frecuentemente, genera código JavaScript que se carga y ejecuta con mayor rapidez que el código JavaScript equivalente creado de forma manual. Por ejemplo, el compilador de GWT suprime de forma segura todo el código no utilizable (mediante una exhaustiva tarea de eliminación de clases, métodos, campos, e incluso parámetros, que no se utilizan) para asegurarse de que el archivo de secuencias de comandos compilado sea lo más pequeño posible. Otro ejemplo: el compilador de GWT realiza una sustitución selectiva de llamadas a funciones en los métodos, lo que permite optimizar el rendimiento de las ejecuciones de métodos.

Durante el desarrollo de una aplicación, es posible visualizar inmediatamente los cambios que hayan sido realizados en el código mediante el navegador de modo alojado de GWT. No es necesario que volver a compilar el código en JavaScript ni implementarlo en un servidor. Solo realizar los cambios y hacer clic en "Actualizar" en el navegador de modo alojado.

Durante la producción, el código se compilará como JavaScript sin formato; sin embargo, durante el desarrollo de la aplicación, ésta se ejecutará en el equipo virtual de Java como código de bytes. Eso significa que, cuando el código realice una acción como gestionar un evento de ratón, se obtendrá una depuración de Java normal completa. Todo lo que pueda hacer el depurador de Java se aplicará también al código GWT. Lo que permitirá la disponibilidad de algunas funciones, como la depuración paso a paso y con puntos de interrupción. La compilación cruzada ofrece la modularidad y la abstracción necesarias para desarrollar aplicaciones sin sufrir una disminución del rendimiento durante la ejecución del programa.

Las aplicaciones GWT admiten automáticamente los navegadores IE, Firefox, Mozilla, Safari y Opera sin necesidad de detectar el navegador ni utilizar un formato especial en el código. Solo es necesario escribir el código una vez y GWT lo convertirá al formato JavaScript más adecuado para el navegador de cada usuario.

2.1.5 JavaFX

Siendo la última plataforma en dar su aparición en esta escena, JavaFX presenta un ambicioso proyecto que trata de unir la experiencia RIA con la capacidad multi-dispositivo que permita disfrutar de una misma aplicación en entornos tan distintos como son los navegadores web, los dispositivos móviles, aplicaciones de escritorio y TV.

Las aplicaciones de navegador web y escritorio se pueden ejecutar en cualquier ordenador que posea el Java Runtime Environment, JavaFX se coloca justo por encima del JRE de manera que es compilado en bycode de Java para poder ser interpretado por JRE: JSE en escritorio o JavaME en el caso de los dispositivos móviles.

Una descripción en profundidad sobre esta nueva tecnología de Sun en [Weaver, 2007], [Alberola, 2009] y en el capítulo 3 de este proyecto: *Introducción a JavaFX*.

2.1.6 Comparativa uso de Frameworks en proyectos de Software

Una vez estudiados los distintos frameworks utilizados para el desarrollo de RIAs, podemos comprobar la repercusión de cada uno de ellos gracias a diferentes estudios sobre la cantidad de proyectos que han sido realizados en estos entornos a lo largo de la escala temporal. En las figuras 2.2 y 2.3 se muestran las gráficas de tendencia temporal que resumen los estudios realizados por Simply Hired e Indeed, respectivamente.

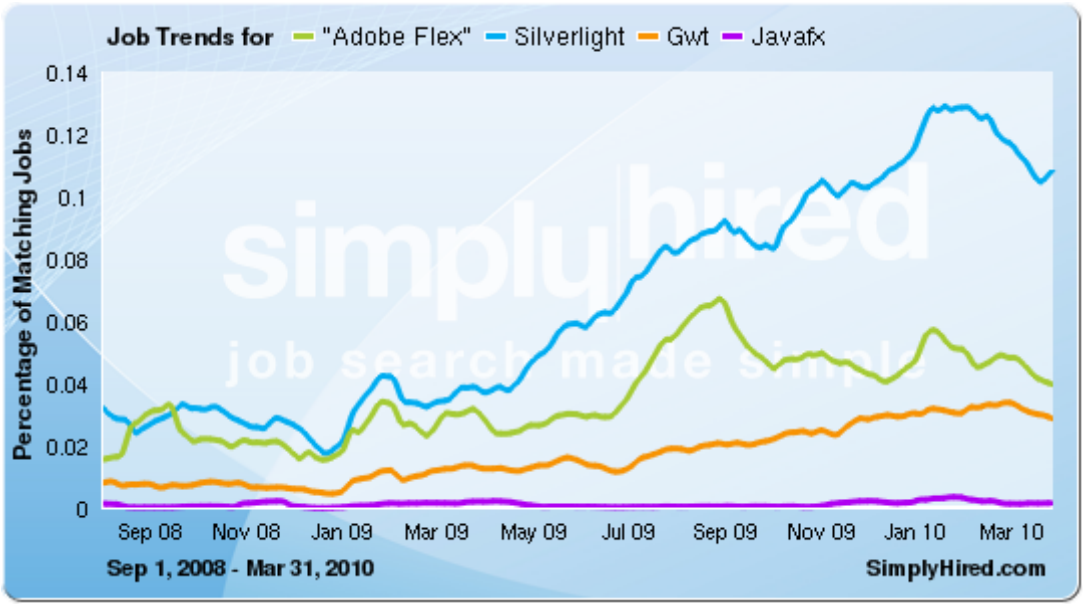


Figura 2.2: Tendencia de uso de frameworks en proyectos según SimplyHired

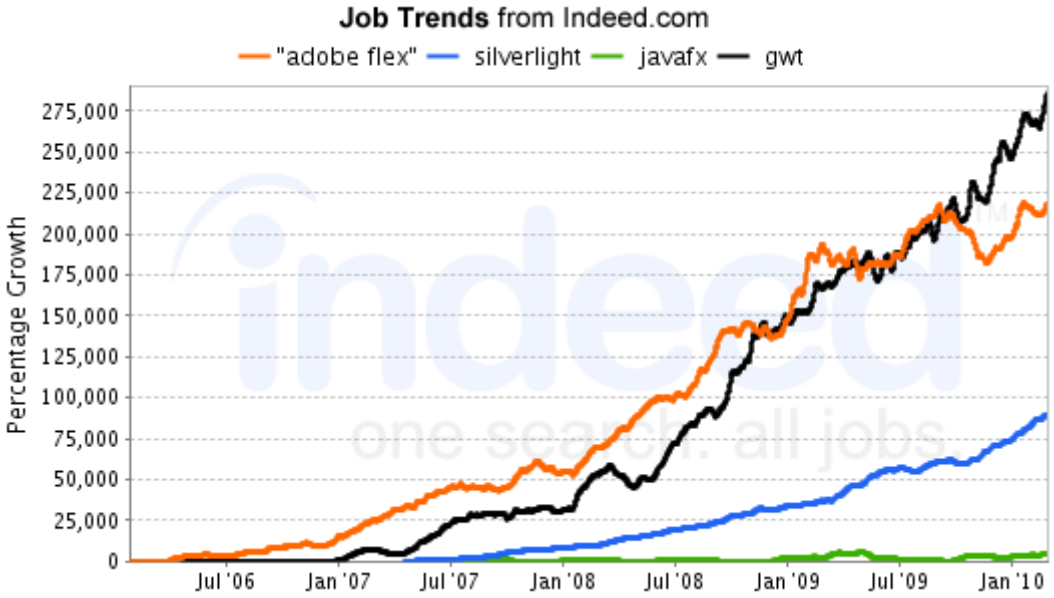


Figura 2.3: Tendencia uso de frameworks en proyectos según Indeed

En ambos casos, a pesar de la contradicción patente entre ambas figuras sobre la distribución y tendencia de las tecnologías Silverlight y GWT y en menor medida Adobe Flex; cabe destacar la baja repercusión de JavaFX, que a pesar de haber sido respaldada por grandes proyectos como la realización del medallero en la web de los juegos olímpicos de invierno Vancouver 2010 [Web Vancouver 2010], sigue sin despegar como tecnología trascendente para la realización de proyectos RIA. Tendencia que, sin embargo, debería cambiar según los planes de Sun con el lanzamiento de nuevas actualizaciones de JavaFX.

2.2 Aplicaciones video-vigilancia Desktop

Las aplicaciones de video-vigilancia que encontramos en el estado de la técnica actual buscan la monitorización mediante la conexión remota a un servidor conectado a las cámaras. Dependiendo del tipo de escenario, podemos encontrar aplicaciones online, móviles o de escritorio. Todas con el esquema básico descrito en la figura 2.4.

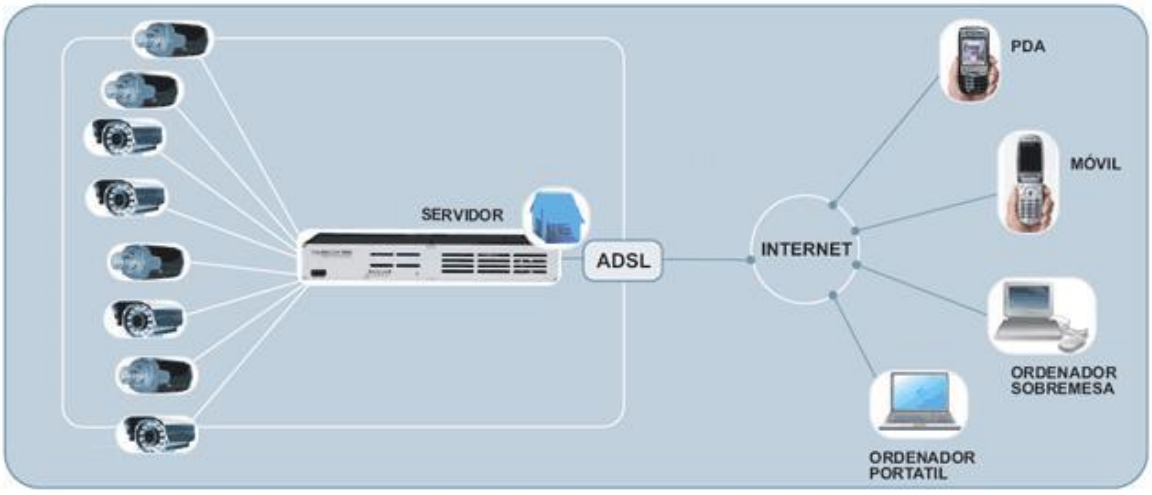


Figura 2.4: Esquema básico sistemas de video-vigilancia

Actualmente, no es complicado encontrar aplicaciones de escritorio muy potentes y con gran funcionalidad para gestionar la video-vigilancia a través de webcam o cámara IP. Estas aplicaciones, por su propia naturaleza de escritorio, están respaldadas por una gran cantidad de herramientas que permiten una excelente funcionalidad. Sin embargo, es también esa naturaleza lo que las restringe a un emplazamiento concreto de la aplicación, con todo lo que conlleva la instalación y manutención de un software en cada punto de utilización de la misma, además de las restricciones propias de una aplicación que no tiene carácter portátil.

2.2.1 uWatchIt

uWatchIt se ha diseñado como plataforma para controlar varias cámaras de seguridad a la vez en una misma pantalla. Además de recoger la imagen de las cámaras, si éstas están preparadas para ello, detecta el movimiento y se pone en modo grabación. Si fuese necesario, es posible configurar el programa para que envíe un aviso vía correo electrónico o transmita las últimas imágenes captadas a un servidor FTP.

uWatchIt tiene cuatro modos de funcionamiento: Visión en tiempo real, visión y grabación continua; visión y grabación cuando se detecte movimiento, y visión y grabación por secuencias para poder editar el vídeo sin que la grabación haya concluido.

Es posible visitar la web del fabricante: [Web uWatchIt].

2.2.2 WebCam Surveillance Monitor

Webcam Surveillance Monitor es una aplicación avanzada para vigilancia en video que cuenta con video y grabación de imágenes activados por movimiento y envío de instantáneas en momentos clave con alertas personalizables. La aplicación da soporte para la modificación de los parámetros de sensibilidad para realizar las alertas según el nivel de seguridad que se quiera dar a cada cámara. Las principales características de esta aplicación son:

- Soporte para vigilancia en vídeo basada en cámara web y cámara IP.
- Captura de instantáneas en los momentos clave.
- Alertas personalizables.
- Parámetros de configuración para la sensibilidad en entornos de vigilancia complejos

Para un mayor información de la aplicación se recomienda visitar [Web WSM].

2.2.3 Vitamin D

Vitamin D es una herramienta para Windows y Mac para la grabación de vídeos de seguridad muy más fáciles de visualizar, gracias a su sistema de reconocimiento de formas que evita el constante uso del avance rápido para localizar momentos concretos de la grabación. La aplicación utiliza webcams o cámaras IP corrientes para obtener las imágenes que después serán procesadas en busca de cualquier movimiento, identificando si se trata de una persona o no. Además de esto, es posible introducir unas sencillas reglas que permiten mostrar únicamente los fragmentos relevantes del clip omitiendo el resto de escenas.

Una herramienta muy profesional y asequible, que va un paso más allá en materia de video-vigilancia doméstica, evitando caer en la simple detección de movimientos para pasar a hacer un reconocimiento de personas o animales. Más información en [Web Vitamin D].

2.3 Aplicaciones video-vigilancia Online

Con quizá una menor capacidad de uso de recursos que las aplicaciones de escritorio, las aplicaciones web de video-vigilancia introducen al usuario a otras posibilidades, facilidades y funcionalidades para la gestión de la seguridad. En este tipo de aplicaciones no es necesaria una instalación o actualizaciones en el PC, tampoco se restringe la capacidad de visualizar las cámaras a un solo computador, pudiendo disponer de cualquier PC conectado a la red para acceder al servicio.

2.3.1 NetVision Premium

La solución profesional de Guardall consiste en un completo sistema de video-vigilancia compuesto por un servidor de vídeo conectado a las cámaras, este servidor permite la monitorización remota de las cámaras a través de una aplicación web. Al tratarse de un sistema profesional, NetVision Premium cuenta con una gran serie de características que lo convierten en una herramienta muy potente:

- Entradas auditables protegidas por contraseña
- Niveles de acceso configurables
- Hora y fecha incrustadas en imágenes
- Control de ancho de banda
- Monitoreo de audio
- Grabación continua y controlada por eventos
- Notificación de eventos

Es posible consultar las especificaciones del servidor de video y de la aplicación en la web [Web Netvision]

2.3.2 UGOlog

UGOlog [Web UGOlog] está 100% basado en la web, por tanto no es necesaria la descarga de ningún tipo de software, requiriendo únicamente el uso de una cámara web o IP. Sus principales características son:

- Detección de movimiento
- Grabación continua o a intervalos
- No es necesaria la instalación de software
- Historial de video
- Sencilla interfaz de usuario
- Soporte para cámaras web y cámaras IP.

2.4 Aplicaciones video-vigilancia Móviles

Si bien las aplicaciones de video-vigilancia para escritorio suponen una gran ventaja en cuanto a capacidades del sistema computacional y herramientas para la visualización de las cámaras IP, las aplicaciones para móviles convierten estos dispositivos en puestos de vigilancia totalmente portátiles que permiten una monitorización continua sin la dependencia de un puesto determinado de trabajo, acabando con las restricciones de emplazamiento del servidor de visualización o almacenamiento.

2.4.1 QNAP VSMobile

VSMobile es una utilidad de video-vigilancia de QNAP que permite monitorizar las cámaras del dispositivo VioStor NVR, un sistema hardware también de QNAP y basado en Linux que recoge la información enviada por una cámara de vigilancia IP para habilitar su visualizado. VSMobile supone una extensión a este sistema dando soporte a la visualización de las cámaras IP de forma remota a través de una PDA con Windows Mobile. Soportando la notificación de eventos que el sistema VioStor NVR dispone.

Figura 2.5: QNAP VSMobile

VSMobile pretende aprovechar la gran demanda existente en el campo de las aplicaciones de video-vigilancia móvil con esta aplicación que, si bien complementa su sistema de video-vigilancia VioStor NVR (sistema descrito en la figura 2.5), por otra parte, resulta absolutamente dependiente del mismo.

Las principales características de VSMobile son:

- Modo de Visualización ajustable.
- Notificación instantánea de eventos
- Control PTZ de las cámaras
- Fotogramas instantáneos
- Control manual de la grabación
- Instalable en dispositivos PDA Windows Mobile

Más información sobre QNAP VSMobile en la web oficial de QNAP [Web QNAP]

2.4.2 Live2Phone

Live2Phone [Web Live2Phone] es una aplicación de video-vigilancia cliente-servidor con la posibilidad de acceder a las cámaras mediante un teléfono móvil. El servidor, componente central de la aplicación, es instalado en un PC y se encarga de captar y procesar las fuentes de video desde una cámara web USB u otra fuente Direct Show como por una capturadora de video o una tarjeta de televisión.

El cliente, la aplicación que se instala en el teléfono móvil, se ofrece en dos versiones. Una compatible con móviles Symbian 3ª generación (mostrada en la figura 2.6) y otra para móviles con soporte JAVA MIDP 2.0. Es un simple visor con el cual es posible conectar a cualquiera de las cámaras controladas por el servidor.

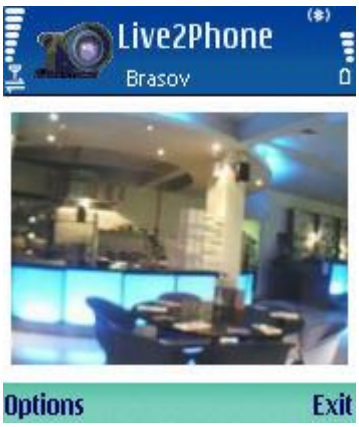


Figura 2.6: Live2Phone

Caracteristicas clave del sistema Live2Phone:

- Soporte para multiples cámaras
- Detección de movimiento
- Reducido uso de ancho de banda
- Velocidad 4-6 fotogramas por segundo

2.4.3 ConnectVu

La firma CCTVMobile ofrece un sistema de vigilancia avanzado: ConnectVu permite observar lo que están grabando las cámaras de seguridad ConnectVu Cam 3G (figura 2.7) a través de un dispositivo móvil Windows Mobile, Symbian, Android o iPhone.

Figura 2.7: Sistema ConnectVU

El programa instalado en el terminal permite ver en tiempo real las imágenes de las cámaras. Además, desde el móvil es posible girar la cámara para cambiar el ángulo de visión, efectuar un zoom sobre los detalles y cambiar la vista de una cámara a otra. También ofrece la posibilidad de modificar la calidad del vídeo que se está grabando. El sistema dispone de detección de movimiento y alertas automáticas cuando se produce actividad en las regiones clave de la imagen. En cuanto se detecta algún movimiento recibimos la pertinente notificación en el móvil. Es recomendable visitar la web de ConnectVU: [Web ConnectVU]

3 Introducción a JavaFX

JavaFX es una familia de productos y tecnologías de Sun Microsystems pensados para el desarrollo de RIAs (Rich Internet Applications), aplicaciones web que tienen características y capacidades similares a las aplicaciones de escritorio, incluyendo aplicaciones multimedia interactivas. Este producto es la contrapartida de Sun para las herramientas de desarrollo de este tipo de aplicaciones, como Flash de Adobe y Silverlight de Microsoft.

Una de las características principales de esta tecnología es la posibilidad del desarrollo de interfaces visuales para escritorio, web, dispositivos móviles y TV bajo un perfil común que permite utilizar la misma aplicación en todos estos ambientes sin la necesidad de reescribir código. JavaFX también favorece el trabajo conjunto de desarrolladores y diseñadores con una serie de herramientas que permiten crear diseños tan complejos como atractivos de una forma sencilla y coordinada. De esta manera los desarrolladores serán más productivos, centrándose en la lógica del programa y abstrayéndose del trabajo de diseño, reservado para los profesionales del campo.

Las principales características de JavaFX son:

- Script declarativo para creación de interfaces gráficas.
- Capacidad de enlace y dependencia Bind.
- Multi-dispositivo: escritorio, navegador web, móvil, TV.
- Capacidad de animación Timeline.
- Gran conectividad web y multimedia.

A continuación se realizará un estudio detallado de la familia tecnológica JavaFX, sin embargo, para una mayor profundidad en cuanto al uso de esta tecnología es siempre recomendable complementar esta información consultando: [James L. Weaver, 2007], [Alejandro Alberola, 2009] y [Web Oficial JavaFX], además de otra bibliografía disponible respecto a la sintaxis JavaFX Script.

3.1 Componentes de la familia

Las tecnologías principales incluidas bajo la denominación JavaFX son JavaFX Script y JavaFX Mobile, aunque dentro de la familia también contamos con el JavaFx SDK, el JavaFX Production Suite y el Java Runtime Environment. Las aplicaciones de navegador web y escritorio se pueden ejecutar en cualquier ordenador que posea el Java Runtime Environment, como se describe en la figura 3.1, JavaFX se coloca justo por encima del JRE de manera que es compilado en bycode de Java para poder ser interpretado por JRE: JSE en escritorio o JavaME en el caso de los dispositivos móviles.

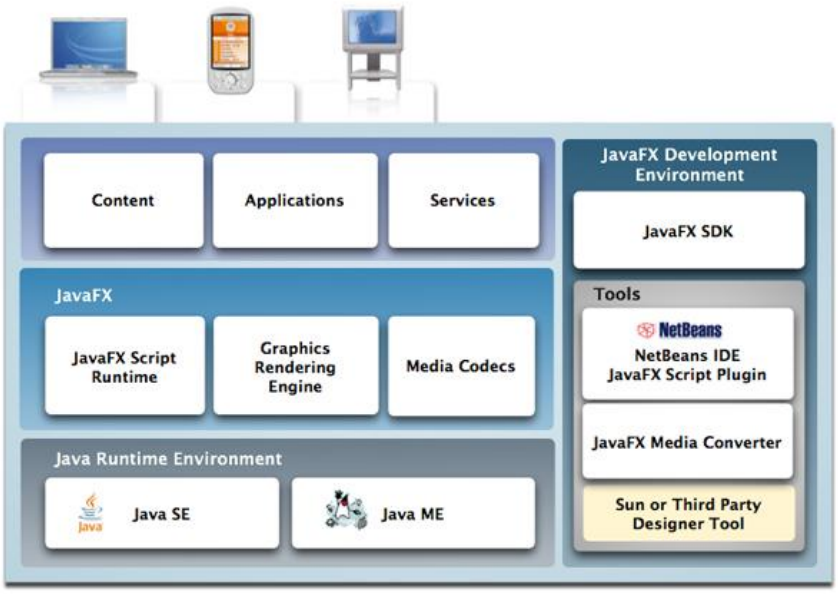


Figura 3.1: Situación tecnología JavaFX

Todo este conjunto de tecnologías tienen como objetivo satisfacer las necesidades por las que surgió el proyecto JavaFX creando un leguaje sencillo de carácter declarativo, soporte para el trabajo conjunto de desarrolladores y diseñadores de aplicaciones, capacidad para crear potentes interfaces gráficas y ejecutar todo esto en un ambiente multiplataforma a través de un perfil común a aplicaciones de escritorio, navegadores web, TV y móviles.

3.1.1 JavaFX Script

JavaFX Script es un lenguaje que se dirige a los autores de contenido web, independientemente de sus conocimientos de programación. Gracias a su uso sencillo, y a sus scripts de sintaxis declarativa, los autores de contenido web pueden crear interfaces de usuario muy potentes centrándose en la descripción de los objetos que van a ser mostrados y a los movimientos y animaciones que van a realizar.

JavaFX Script es un lenguaje estáticamente tipado y totalmente orientado a objetos, con métodos (llamados operaciones y funciones en JavaFX) y atributos. En este aspecto el lenguaje JavaFX mantiene una estrecha relación con Java, con quien comparte además toda la funcionalidad Swing gracias al gran acoplamiento posible entre estos dos lenguajes. La programación de interfaces de usuarios con las herramientas Swing que brinda Java permiten realizar una increíble cantidad de funcionalidad, pero a su vez, esta puede llegar a ser muy compleja. JavaFX impulsa todo el poder de Java, debido a que su código puede utilizar en su totalidad las librerías de este lenguaje. Muchas de las capacidades de la

interfaz de usuario de JavaFX hacen uso de las clases de Java. El efecto producido por esta nueva tecnología es que desarrolladores y diseñadores pueden utilizar un lenguaje simple y elegante que utilice todos los beneficios de Java y Java Swing.

La sintaxis declarativa es utilizada para declarar interfaces de usuario, incluyendo una amplia colección de herramientas que facilitan en gran medida los desarrollos de las interfaces de usuario. Los diseñadores de contenido pueden crear interfaces atractivas sin tener que ser expertos programadores. El lenguaje JavaFX Script ha sido diseñado especialmente para soportar el patrón Model-View-Controller, un modelo altamente utilizado en aplicaciones Web que se basa en la separación de los datos de aplicación, la interfaz de usuario y la lógica de control en tres componentes distintos. La interfaz de usuario esperará recibir cualquier evento y se lo comunicará al controlador, que automáticamente actualizará el modelo para que éste modifique adecuadamente la interfaz de usuario mostrada. Es posible encontrar más información sobre este patrón en [Steve Burbeck, 1992].

Gracias a su poderosa capacidad de enlace, JavaFX Script admitirá el modelo MVC como muestra la figura 3.2. Esta funcionalidad complementa y habilita la sintaxis de programación declarativa debido a que los atributos de los objetos, incluyendo los objetos de interfaz de usuario, pueden ser enlazados a valores contenido en el modelo de clase. Este enlace puede ser bidireccional.

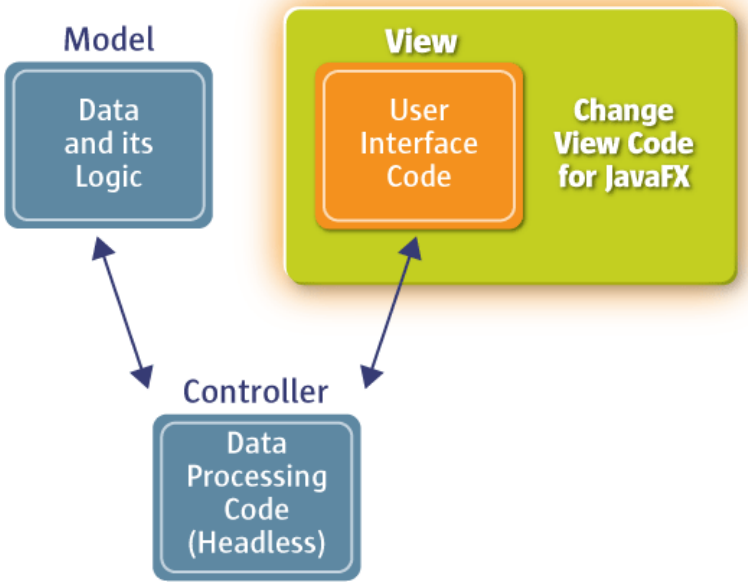


Figura 3.2: Patrón Model-View-Controller

El concepto de *triggers* habilita la sintaxis de programación declarativa, de la misma manera que vuelve los desarrollos de interfaces de usuario relativamente sencillos, los métodos *set* y *get* son reemplazados por *triggers* que son invocados automáticamente cuando el valor de un atributo es modificado.

Otra ventaja de este lenguaje reside en su poderosa sintaxis para definir, modificar, y consultar las secuencias (arrays). Los programas realizados con JavaFX podrán ejecutarse en cualquier lugar que pueda correr una aplicación Java, debido a que se ejecutan dentro del contexto de la **JVM** (*Java Virtual Machine)***.**

3.1.2 JavaFX Mobile

Está basado en la plataforma Java ME, aprovechando su potencial para dispositivos móviles. Actualmente la plataforma JavaME se ejecuta en más de 2 mil millones de teléfonos, y Sun quiere aprovechar la ventaja que aporta esta situación. Uniendo estas dos tecnologías, JavaFX podría confirmarse como la principal tecnología para el desarrollo de RIA en dispositivos móviles.

Lo más beneficioso de todo esto para los desarrolladores, es que, el contenido creado con la tecnología de JavaFX tiene la capacidad de ejecutarse en casi cualquier pantalla (ordenadores, aplicaciones Web, móviles, TV). Esto significa que el desarrollo de una interfaz de usuario para un tipo de pantalla será mucho más fácil de trasladar a otra pantalla que con cualquier otra tecnología. JavaFX Mobile no es sólo la implementación de JavaFX para móviles, sino que consta de dos partes, un SO basado en Linux y el entorno para construir aplicaciones para este SO o para otros ya existentes.

Como SO JavaFX Mobile constituye un sistema software completo para teléfonos móviles, el sistema está construido en torno al kernel Linux y a tecnologías Java (todo bajo licencia GPL), y pretende proporcionar a los fabricantes de teléfonos móviles un entorno software completo (con sus herramientas de desarrollo incluidas) en el que basar sus productos, de forma que estos se puedan centrar en el desarrollo de servicios que los diferencien de sus competidores y no en reinventar las mismas aplicaciones continuamente para adaptarse a los medios. Sun está trabajando con otros miembros en el ecosistema de la telefonía móvil para construir una completa solución compatible con la mayoría de dispositivos de la actualidad. Además, el contenido desarrollado con JavaFX Mobile será completamente compatible con los sistemas cada vez más complejos de las nuevas generaciones, volviendo menos complicado la migración hacia estos últimos. Esta tecnología no fue lanzada al mercado hasta la versión 1.1 de JavaFX, en la cual se incluyó un emulador para móviles, permitiendo a los desarrolladores crear aplicaciones para este tipo de dispositivos.

Por tanto, además de sobre el sistema operativo JFX Mobile, las aplicaciones JFX pueden correr sobre miles de dispositivos, todos aquellos basados en Android, Symbian o Windows Mobile. Sin embargo, en la actualidad solo está disponible el SDK de Windows Mobile, por lo que habrá que esperar para poder disfrutar de la tecnología JavaFX sobre el resto de Sistemas Operativos móviles.

3.1.3 JavaFX Production Suite

JavaFX Production Suite es un conjunto de herramientas pensadas para diseñadores gráficos profesionales que no se quedan en ser sólo un conjunto de plugins, sino que se convierten en herramientas desarrolladas explícitamente para este objetivo de coordinar y facilitar el trabajo conjunto de diseñadores y desarrolladores abstrayendo a estos últimos del trabajo que sin duda corresponde a los primeros.

Un ejemplo de estas herramientas es Project Nile, que aporta una interfaz para la conversión de gráficos a un formato de almacenamiento (FXD) que puede ser invocado por JavaFX, ya que posee una sintaxis declarativa para su manejo. Project Nile está compuesto por cinco componentes:

- Illustrator Plugin: Adobe Illustrator CS3 plugin, permite la exportación de archivos a formato FXD o código JavaFX.
- Photoshop Plugin: Adobe Photoshop CS3 plugin, permite la exportación de archivos a formato FXD o código JavaFX.
- SVG Converter: Herramienta que permite la conversión de SVG a gráficos FXD o código JavaFX, desde la línea de comandos o una interfaz.
- JavaFX Graphics Viewer: Herramienta que permite la visualización de gráficos generados por Project Nile.
- Librería: Una librería (archivo JAR) que debe adjuntarse a cualquier proyecto, para la utilización del formato de archivos FXD.

Desarrolladores y diseñadores trabajarán en gráficos y código como componentes separados, teniendo por seguro que ambos componentes podrán integrarse perfectamente en cualquier momento. El experto gráfico puede definir capas y elementos en sus diseños utilizando potentes aplicaciones de diseño (Adobe Illustrator ó Adobe Photoshop), y el desarrollador puede acceder a los mismos referenciándolos desde código JavaFX y animarlas.

3.1.4 JavaFX SDK

El kit de desarrollo de software contiene el compilador JavaFX y herramientas de ejecución, gráficos en 2D y bibliotecas de medios para crear aplicaciones interactivas para el escritorio y el navegador, así como tutoriales y documentación API para crear aplicaciones RIA para escritorio, navegadores de Internet y plataformas móviles.

La API JavaFX manejará el desarrollo según los distintos ambientes de ejecución a través de los perfiles *common* y *desktop* (en JavaFX release 1.3). Las funciones que sólo estén disponibles para aplicaciones de escritorio tendrán perfil desktop, y no compilarán para el resto de modos de ejecución, que necesitarán que todas las funciones usadas en la aplicación tengan perfil *common* para poder compilar y ejecutarse.

3.1.5 Netbeans IDE

Siendo el IDE utilizado para el desarrollo de este proyecto, NetBeans incorpora, a partir de su versión 6.7, un plugin para JavaFX que incluye soporte drag-and-drop para añadir objetos, además, se puede ejecutar una preview del funcionamiento de la aplicación sobre los distintos ambientes de ejecución:

Ejecución Estándar (por defecto)

- Java Web Start
- Navegador Web
- Emulador de dispositivo móvil

A partir de la versión 6.9 (adaptado a JavaFX Release 1.3) el plugin añade soporte para desarrollo de aplicaciones para TV, así como una preview de este ambiente de ejecución para comprobar su correcto funcionamiento. Es digno de mención que también está disponible un plugin JavaFX para desarrolladores que utilicen Eclipse IDE.

3.2 Historia y versiones de JavaFX

JavaFX comenzó como un proyecto de Chris Oliver llamado F3 que fue anunciado en la conferencia de desarrolladores JavaOne en mayo de 2007 y liberado en diciembre de 2008, desde entonces ha pasado por distintas versiones hasta la actual JavaFX Release 1.3. Poco a poco las nuevas versiones consiguen acercarse a los objetivos finales del proyecto JavaFX, para los que, sin embargo, faltan aún ciertas mejoras fundamentales y llamativas: soporte para la ejecución de aplicaciones JavaFX en TV y móviles con sistema operativo distinto a Windows Mobile o el completo desarrollo del sistema operativo JavaFX Mobile además de otras mejoras necesarias.

JavaFX Release 1.0: En diciembre de 2008, Sun lanzó la primer release de JavaFX, esta traía varias mejoras, y muchas modificaciones a la versión anterior (Beta). Se agregaron nuevas clases para el manejo de gráficos y animaciones, además de ciertas mejoras multimedia, mecanismos de control de acceso y soporte para web services. También hubo ciertas mejoras en el uso de memoria en tiempo de compilación y ejecución y se realizó el soporte para la API de reflection, lo que permite la llamada de las APIs de JavaFX desde Java y JavaME.

JavaFX Release 1.1: Lanzada en febrero de 2009, en esta versión contamos con un emulador para teléfonos móviles, permitiéndonos desarrollar aplicaciones para los mismos. Aunque la SDK de JavaFX aún no se encuentra disponible para los teléfonos. Se mejoró la performance y estabilidad de las aplicaciones de escritorio. Mejor soporte para la portabilidad de aplicaciones entre la plataforma Mobile y Desktop. Se incorporó soporte para aplicaciones en modo pantalla completa y se agregaron los tipos numéricos a los tipos definidos por el sistema.

JavaFX Release 1.2: Esta versión ha sido la utilizada para la realización de este proyecto por coincidir temporalmente con el estudio y posterior utilización de los servicios que ofrece la familia JavaFX. Fue lanzada en junio de 2009 y se ha mantenido como release principal hasta la llegada de la 1.3 en abril de 2010. Representa una mejora bastante grande con respecto a su versión predecesora, tanto que es incompatible con ella o cualquiera de las anteriores. Se han eliminado clases de la API. La SDK de JavaFX 1.2 no soporta los archivos binarios de la versión 1.1, esto significa que para que los desarrollos realizados con versiones anteriores funcionen, es necesario recompilarlos con la última versión. Cabe destacar como mejorías en esta versión, ciertos aspectos que han sido fundamentales en el desarrollo del proyecto y sin los cuales no habría sido posible su realización:

- Es posible salvar información de forma permanente gracias a las nuevas clases *Storage* y *Resource*, que la almacenan con un mecanismo similar al de las cookies en los navegadores web.
- Se crea la SDK para Windows Mobile versión 6.x, primer SO móvil que soportará la tecnología, y por tanto, el usado en este proyecto.

- Se define la clase *Properties* para facilitar el manejo de archivos de texto con información almacenada basada en pares para su posterior utilización.
- Se establece soporte beta para el desarrollo y utilización de tecnología JavaFX en SO Linux (Ubuntu 8.04 o superior)

JavaFX Release 1.3: Sin representar grandes incompatibilidades o cambios con la release anterior, se producen ciertas mejoras relevantes en los controles GUI y el rendimiento en tiempo de ejecución. Como novedad, se incorpora soporte de creación de aplicaciones para TV, aunque todavía no esté disponible una SDK que permita correr estas aplicaciones en un dispositivo de TV.

3.3 Sintaxis JavaFX Script

La sintaxis del lenguaje JavaFX Script es acorde con los fundamentos básicos JavaFX, así, el lenguaje tratará de ser lo más sencillo posible, dando prioridad a la descripción sobre la explicación algorítmica. JavaFX será, por tanto, un lenguaje de sintaxis declarativa, puramente funcional.

La potencia del lenguaje JavaFX radica entre otras cosas en su capacidad para actualizar el escenario gráfico en tiempo real según unas reglas temporales marcadas por el programador o por las propias interacciones del usuario. Podemos conseguir esto gracias al uso de herramientas como los *Timelines*, el *Binding* y los *Triggers*. Estas herramientas y demás por menores de la sintaxis JavaFX serán descritas a continuación, comenzando por los aspectos más básicos como variables o tipos de datos y llegando hasta la creación de objetos gráficos o el visionado de imágenes. No obstante, es posible ampliar la información sobre el lenguaje con: [Gail Anderson, 2009], [Simon Morris, 2010] y [Web JavaFX: Script]. Todos los ejemplos de código expuestos a continuación (excepto los creados exclusivamente para este proyecto) pertenecen a [Gail Anderson, 2009].

3.3.1 Variables

En JavaFX, se definen las variables con la palabra reservada *var*. Para las variables de sólo lectura se utiliza la palabra *def*. Los intentos de modificar variables *def* directamente darán como resultado un error en tiempo de compilación.

```
def maxLength = 100;          // Solo lectura
var count = 0;                // Variable de lectura-escritura
count++;                      // La variable count puede ser modificada
maxLength = 500;              // Error de compilación
```

Imprimir las variables por consola es muy sencillo con JavaFX, basta con usa la función *print(var)* o *println(var).*

```
var b = true;
println(b);                   // true
var i = 12;
println(i);                   // 12
var s = "javafx";
println(s);                   // javafx
print("javafx");              // javafx (sin línea nueva)
println("");                  // Línea vacía con retorno de carro
```

Además es posible utilizar llaves "{}" para imprimir variables insertadas en una cadena.

```
var i = 12; var s = "javafx";
println("i = {i}, s = {s}");      // i = 12, s = javafx
```

3.3.2 Tipos de datos

JavaFX es un lenguaje estáticamente tipado con los siguientes tipos integrados: *Boolean*, *Integer*, *Number*, *String*, *Duration* y *Void*. Echemos un vistazo a los tipos integrados JavaFX con mayores detalles.

3.3.2.1 Boolean

El tipo booleano es útil para variables que representen flags y estados internos de aplicación. Las palabras reservadas true y *false* se utilizan para darle a las variables los valores booleanos. El valor por defecto de las variables es *false*.

```
var isElement: Boolean;      // Tipo Boolean, false por defecto
var flag = true;             // Iniciación de booleano
isElement = 1;               // Error de compilación
flag++;                      // Error de compilación
```

3.3.2.2 Integer

El tipo *Integer* se utiliza para representar todos los posibles valores enteros en 32 bits, para inicializar o asignar estas variables se pueden utilizar valores decimales, octales (0) o hexadecimales (0x).

```
var counter: Integer;        // Tipo Integer (32 bits), por defecto 0
def length = 80;             // Integer inicializado
var byte = 0x3f;             // Valor hexadecimal
var word = 037;              // Valor octal
```

3.3.2.3 Number

Number se utiliza para aquellas variables que necesiten precisión de 32 bits coma flotante. El valor por defecto es 0.0.

```
var radius: Number;          // Variable tipo Number, por defecto 0.0
var value = 1.1;             // Number inicializado
var big = 1.56e10;           // Number valor alto
var small = 6.32e-25;        // Number valor bajo
```

3.3.2.4 String

El tipo *String* representa cadenas de caracteres de cualquier longitud. Puede usar comillas dobles (") o comillas simples (') para encerrar la de cadena de caracteres. El carácter de comilla simple saldrá escrito en una cadena encerrada entre comillas dobles y viceversa. El compilador concatena cadenas adyacentes en tiempo de compilación por lo que es posible definir las cadenas en varias líneas. Se utilizan llaves "{}" para integrar expresiones en las cadenas. El valor predeterminado es "".

```
var s: String;               // Tipo String, por defecto ""
var a = "datapoint";         // String inicializado
var b = "I can't figure it out, that's too hard";
def c = "Hello "             // Strings adyacentes
"and Goodbye";               // "Hello and Goodbye"
```

```
var s1 = 'duck';            // Comillas simples
var s2 = 'soup';            // Comillas simples
var s3 = "{s1} {s2}"; // "duck soup"
var s4 = s1 s2;       // Error de compilación
var s5 = "alpha" "bet";     // "alphabet"
```

3.3.2.5 Duration

El tipo *Duration* representa los valores de las unidades de tiempo. Las duraciones se denotan con literales de tiempo (*ms* para milisegundos, *s* para segundos, *m* para minutos, y *h* para horas). Normalmente se utilizan variables *Duration* para animaciones y *Timelines* en JavaFX. El valor predeterminado es 0 *ms*.

```
var timeSlice: Duration;    // Tipo Duration, por defecto 0ms
var period = 100ms;         // Duration inicializado, 100 milisegundos
var halfMinute = 30s; // 30 segundos
var halfHour = 30m;         // 30 minutos
var halfDay = 12h;          // 12 horas
var halfPast = .5h;         // Media hora
```

3.3.2.6 Void

Se usa el tipo *Void* para definir funciones que no retornan ningún valor.

```
function startSimulation(): Void {
simulate(100ms);
// no hay valor de retorno
}
```

3.3.2.7 Tipos de Datos Java

En JavaFX, y a pesar de que usualmente no será necesario, también es posible utilizar los tipos primitivos de Java. Estos tipos se utilizan principalmente para la interoperabilidad con las clases primitivas Java que puedan tener uso en las interfaces.

```
var ch: Character = 120;    // Carácter Unicode (16 bits): 'x'
var ts: Byte = 127;         // 8 bits
var data: Byte = 500; // Error de compilación: más de 8 bits
var ts: Short = 10;         // Short (16 bits)
var sdata: Short = 50000;   // Error de compilación: más de 16 bits
var tl: Long = 100000;      // Long (64 bits)
var tg: Double = 3.45;      // Double (64 bits)
var tf: Float = 1.23; // Float (32 bits): mismo que Number
```

3.3.2.8 Pseudo-Variables

JavaFX soporta varias variables predefinidas de sólo lectura para su uso en scripts.

Nombre	Descripción
__PROFILE__	Ambiente (“mobile”, “desktop”, “browser”)
__FILE__	Path y nombre completo del archivo Script
__DIR__	Path del archivo Script

3.3.3 Binding

Binding es una de las características más potentes de JavaFX, se trata de crear una dependencia de una variable del programa sobre algo que cambiará en la ejecución del programa. De esta forma el valor de esta variable se actualizará automáticamente en función de la dependencia creada. JavaFX permite crear dependencias de variables (*var* o *def*) con respecto a bloques, condicionales, funciones, bucles for y objetos literales. La sintaxis para realizar el binding es la siguiente:

```
def v = bind expression;
```

La variable v será actualizada cuando la expresión cambie. Para comprender mejor esta expresión podemos observar más ejemplos, en éste, se pretende que el producto de dos números varíe en función de los propios números:

```
var a = 10; var b = 2;
def product = bind a * b;    // bind con la expresión
println(product);            // 20
a = 100;
println(product);            // 200
b = 5;
println(product);            // 500
product = 10;                // Error de compilación
```

A pesar de poder declarar las variables dependientes como var, el compilador no nos permitirá su modificación directa.

```
var prod = bind a * b; // bind con la expresión
prod = 10;       // AssignToBoundException en tiempo de ejecución
```

Este otro ejemplo que llama a la función getNumber para actualizar la variable top nos permitirá asegurarnos de forma sencilla que el número obtenido no sea mayor que 100.

```
def top = bind if (num <= 100) num else 100; // bind a una condición
var num = getNumber();       // suponemos que el número es 50
println(top);                      // top valdrá 50
num = getNumber();                 // suponemos que el número es 500
println(top);                      // top valdrá 100
```

En este ejemplo se utiliza bind para crear una dependencia de la variable sum sobre un bloque:

```
a = 10; b = 20; var c = 30;
def sum = bind {                   // bind a un bloque
     def d = 40; def e = 50;
     a + b + c + d + e             // hace bind con esta sentencia
}
println(sum);                      // sum valdrá 150
b = 50; c = 50;
println(sum);                      // sum valdrá 200
```

3.3.4 Binding bidireccional

El binding bidireccional suele usarse en las interacciones con el usuario para asegurar que las variables guardadas y las mostradas se mantengan sincronizadas. El formato de esta expresión es:

```
var v = bind w with inverse;
```

Este formato permitirá actualizaciones en ambos sentidos. La variable v cambiará con w y viceversa. En este ejemplo vemos como se ambas variables se mantendrán sincronizadas:

```
var field = "one";
var name = bind field with inverse;     // name valdrá "one"
field = "two";                          // name valdrá "two"
name = "three";                    // field valdrá "three"
```

3.3.5 Secuencias

En JavaFX una secuencia es una lista ordenada de elementos, similar a los arrays de otros idiomas. Las secuencias son muy poderosas en JavaFX, ya que se pueden utilizar para almacenar todos los datos tipo, incluidos los objetos. Hay un gran número de posibilidades para manipular las secuencias. Para crear una secuencia basta con listar una serie de términos entre comas "," y encerrarlos entre corchetes "[]", el compilador sabrá interpretar el tipo de datos de la secuencia, aunque también es posible incluirlo explícitamente:

```
var brothers = ["groucho", "chico", "harpo"]; // Valores String
var brothers: String[] = ["groucho", "chico", "harpo"];
def primeNumbers: Integer[] = [2, 3, 5, 7, 11, 13];
```

JavaFX proporciona asistencia para la creación de secuencias en incremento de cualquier tipo.

```
def nineties = [1990..1999];      // [1990, 1991, 1992, ..., 1999]
var g = [1..5];        // mismo que [1, 2, 3, 4, 5]
var h = [1.1..5.1];          // mismo que [1.1, 2.1, 3.1, 4.1, 5.1]
var j = [1..<5];             // mismo que [1, 2, 3, 4];
var k = [1..5.0];            // mismo que [1.0, 2.0, 3.0, 4.0, 5.0]
var m = [4.5..7];            // mismo que [4.5, 5.5, 6.5]
var n = [5..1];        // valores no en incremento, mismo que []
```

El incremento por defecto es 1, pero esto puede modificarse aplicando la notación con *step*.

```
var p = [0..9 step 2];       // mismo que [0, 2, 4, 6, 8]
var q = [5..1 step -1];      // mismo que [5, 4, 3, 2, 1]
var r = [1..3 step .5];      // mismo que [1.0, 1.5, 2.0, 2.5, 3.0]
```

El operador *sizeof* devuelve el tamaño de la secuencia.

```
def primeNumbers = [2, 3, 5, 7, 11, 13];
var length = sizeof(primeNumbers);      // length valdrá 6
var length = sizeof primeNumbers; // lo mismo
```

El operador *reverse* da la vuelta a los elementos de una secuencia.

```
var brothers = ["groucho", "chico", "harpo"];
var revBros = reverse brothers; // ["harpo", "chico", "groucho"]
```

Es muy sencillo concatenar los elementos de varias secuencias en otra:

```
def boys = ["billy", "joey"];
def girls = ["mary", "susie"];
var kids = [boys, girls]; // ["billy", "joey", "mary", "susie"];
```

Para imprimir todos los valores no es necesario más que usar directamente *println()*.

```
def primeNumbers = [2, 3, 5, 7, 11, 13];
println(primeNumbers); // [ 2, 3, 5, 7, 11, 13 ]
println("{primeNumbers}"); // 23571113
```

Para imprimir los elementos uno a uno se puede utilizar un bucle *for*.

```
def primeNumbers = [2, 3, 5, 7, 11, 13];
for (n in primeNumbers) {
     print("{n} ");
}
println(""); // 2 3 5 7 11 13
```

El operador *indexof* devolverá el índice del elemento indicado en la secuencia, aunque este operador sólo puede ser utilizado en un bucle que recorra la secuencia:

```
var colors = ["red", "blue", "green"];
for (c in colors) {
     println("{c} is color #{indexof c}");
}
```

Cada iteración en este bucle mostrará un nombre de color y su número de índice en la secuencia.

```
red is color #0
blue is color #1
green is color #2
```

3.3.6 Operadores

JavaFX tiene operadores aritméticos, lógicos, relacionantes, unarios, y de asignación. El operador *Instanceof* determina el tipo de dato de una variable.

3.3.6.1 Operadores Aritméticos

Los operadores aritméticos son suma(+), resta(-), multiplicación(*), división(/) y módulo(mod). Podemos ver unos ejemplos:

```
var val = 25; var nt = 4;
var a = val + nt;           // a valdrá 29
var b = val - nt;           // b valdrá 21
var c = val * nt;           // c valdrá 100
var d = val / nt;           // d valdrá 6
var e = val mod nt;         // e valdrá 1
```

El operador efectuará la operación más razonable en el caso de que se mezclen tipos en una operación:

```
var fv = 6.5;                // fv es de tipo Number
var ic = 6;                  // ic es de tipo Integer
var td = 100ms;        // td es de tipo Duration
var tg = 10ms;               // tg es de tipo Duration
var gv = fv * ic;            // gv es de tipo Number
var tf = td / fv;            // tf es de tipo Duration
var hm = td / tg;            // hm es de tipo Number
var mm = td * tg;            // Error de compilación
```

3.3.6.2 Operadores de asignación

Además de las asignaciones convencionales con =, es posible realizar asignaciones con operaciones aritméticas:

```
var val = 25; var nt = 4;
val += nt;                   // val = val + nt; (val valdrá 29)
val -= nt;                   // val = val - nt; (val valdrá 25)
val *= nt;                   // val = val * nt; (val valdrá 100)
val /= nt;                   // val = val / nt; (val valdrá 25)
```

El operador de modulo no está disponible para asignación compuesta:

```
val = val mod nt;            // val valdrá 1
```

3.3.6.3 Operadores Unarios

Aunque la mayoría de los operadores en JavaFX son de dos operandos (binarios), existe algunos de un solo operando, como pre-incremento y post-incremento (++), pre-decremento y post-decremento (--), negación (-), y complemento lógico (not):

```
var data = 10;
var negate = -data;          // negate valdrá -10
var m; var n;
var s = 5; var t = 5;
m = s++;                     // m valdrá 5, s valdrá 6
n = ++s;                     // n valdrá 7, s valdrá 7
m = t--;                     // m valdrá 5, t valdrá 4
n = --t;                     // n valdrá 3, t valdrá 3
var enabled = false;
var start = not enabled;     // start valdrá true
```

3.3.6.4 Operadores Relacionantes

Los operadores relacionantes comparan valores (>,>=,<,<=) y comprueban igualdades (==,!=):

```
var f = 10.5; var g = 20.5; var h;
h = (f == g);                // h valdrá false
h = (f != g);                // h valdrá true
h = (f > g);                 // h valdrá false
h = (f < g);                 // h valdrá true
h = (f >= g);                // h valdrá false
h = (f <= g);                // h valdrá true
h = (f > 10);                // h valdrá true
```

La variable h obtendrá un valor de tipo booleano, el último ejemplo compara tipos diferentes, (*Integer* y *Number*) por lo que el *Integer* se transformará en *Number* para realizar la comparación. Cuando se trata de *Strings*, los operadores de igualdad(==,!=) realizarán comparaciones por el valor de la cadena (a diferencia de Java):

```
def input: String = getInput();    // obtener input String
var exit = (input == "quit");      // true si input string es "quit"
```

3.3.6.5 Operadores Lógicos

Los operadores *and* y *or* permiten combinar expresiones booleanas:

```
var i = 2; var j = 3; var k = 4;
var p = (j > i and j < k);         // p valdrá true
var q = (j > k or i < k);          // q valdrá true
```

Ambos operadores tienen comportamiento "short-circuit", esto es, si la primera expresión del operador and es falsa, la segunda no será evaluada. De la misma forma, la segunda expresión del operador *or* no será evaluada si la primera es *true*.

3.3.6.6 Operador Instanceof

El operador *instanceof* indicará si un objeto es de un tipo de dato específico, retornando *true* si el tipo coincide:

```
def u = "string of chars";         // String
var v = u instanceof String;       // v valdrá true
def w = 6.5; // Number (Float)
v = w instanceof java.lang.Float;// v valdrá true
```

Normalmente se hace uso de este operador para determinar si un objeto es de una clase específica en tiempo de ejecución.

3.3.7 Expresiones

Muchos de los bloques, bucles y condicionales en JavaFX son expresiones con valor por sí mismas. En algunos casos, por ejemplo, será posible utilizar el valor de retorno de una sentencia en una asignación o el valor de retorno de un bucle para crear una nueva secuencia.

3.3.7.1 Bloques

En JavaFX, un bloque es una lista de expresiones encerradas entre llaves "{}", el valor de la expresión de bloque será el valor de la última expresión. Los bloques pueden estar vacíos (devolverán *Void*), las sentencias de tipo de tipo *var* o *def* también se consideran expresiones (su valor es el valor de la nueva variable). Los bloques suelen usarse típicamente para el cuerpo de las funciones, para sentencias tipo *if* y para bucles *for* o *while*, los bloques pueden contener variables locales, con alcance solo para ese mismo bloque. Finalmente, los bloques también pueden retornar variables:

```
var a = 10; var b = 20; var c = 30;
     var sum = { def d = 40; def e = 50;      // inicio de bloque
     a + b + c + d + e       // valor de la última expresión
}                            // final de bloque
println(sum);                // sum valdrá 150
```

El tipo y valor de retorno en un bloque serán el tipo y valor de retorno de su última expresión, que no tiene por qué finalizar en punto y coma.

3.3.7.2 If

Una expresión *If* cambiará el flujo del programa en función de la expresión booleana que comprobará internamente:

```
var num;
var i: Integer = getNumber();
if (i >= 0 and i <= 9)
num = "single digit";
```

Está disponible también el uso de la expresión *else*:

```
var num;
var i: Integer = getNumber();
if (i >= 0 and i <= 9) {
     num = "single digit";
} else if (i >= 10 and i <= 99) {
     num = "two digits";
} else
    num = "three digits or more";
```

Ya que las expresiones JavaFX tienen un valor propio, es posible sintetizar todo este código de la siguiente forma (sin el uso de llaves):

```
var i: Integer = getNumber();
var num = if (i >= 0 and i <= 9) "single digit"
else if (i >= 10 and i <= 99) "two digits"
else "three digits or more";
```

La expresión *If* mantendrá el control de la ejecución de una sentencia, en caso de que sean necesarias más sentencias habrá que encerrar el código como un bloque.

3.3.7.3 For

La expresión *for* aporta una forma sencilla de realizar una acción un determinado número de veces:

```
for (i in [1..5]) {
     doSomething();    // llamada a la función cinco veces
}
```

La notación *[1..5]* genera una secuencia literal. También es recomendable el uso de un bucle *for* para recorrer una secuencia:

```
var strings = ["straight", "forward", "thinking"];
for (s in strings) {
     writeText(s);     // llamada a la función cinco veces con s
}
```

Este bucle asignará a la variable s cada uno de los elementos de la secuencia, y llamará a la función pasando en cada iteración el valor de la variable en ese momento.

3.3.7.4 While

Un bucle *while* iterará mientras la condición dada siga siendo true. Las expresiones *While* no retornarán un valor y serán adecuadas cuando sea posible expresar la condición de continuación como un booleano:

```
var u = 1234;          // contador del número de cifras del número
var numDigits = 0;
while (u != 0) {
     numDigits++;
     u /= 10;
}
println(numDigits);    // numDigits valdrá 4
```

3.3.7.5 Break y Continue

Las palabras reservadas *break* y *continue* deben aparecer sólo dentro de bucles *while* y *for*. *Continue* saltará a la siguiente iteración del bucle mientras que *break* saldrá del mismo.

```
var str: String = getString();
for (weekDay in ["Mon", "Tues", "Wed", "Thu", "Fri"]) {
     if (weekDay == "Wed") continue;     // salta Wed
     . . .
     if (str == weekDay) break;          // salir del bucle
}
```

3.3.8 Funciones

Las funciones ayudan a centralizar el código de diferentes lugares de los scripts tomando argumentos y devolviendo valores. Además es posible crear dependencia de funciones con bind e incluso estrechar esta relación con las funciones *bound*. Para pasar argumentos a funciones o especificar un posible retorno se pueden incluir los tipos en la firma de la función, sin embargo esto no es necesario, ya que JavaFX es capaz de deducir por sí mismo el tipo del valor de retorno y el de los argumentos:

```
function printStrings(a: String, b: String): Void {
     println("{a}{b}");
}
     printStrings("alpha", "bet");       // imprime "alphabet"

function printStrings(a, b) {
     println("{a}{b}");
}
printStrings("alpha", "bet");      // imprime "alphabet"
```

Un aspecto importante de los argumentos de las funciones en JavaFX es que estos serán siempre pasados por valor, lo que hace imposible crear ciertos tipos de funciones, como este ejemplo que pretende intercambiar el valor de los parámetros de entrada:

```
function swap(a, b) {
     var tmp = a;
     a = b;             // Error de compilación
     b = tmp;           // Error de compilación
}
```

Además de utilizar el binding en expresiones, es posible hacerlo en las llamadas a las funciones, de esta forma, cuando uno de los argumentos de la función cambie, la variable que pide realizar la llamada a la función será automáticamente modificada:

```
import java.lang.Math;
function hypot(a, b) {
     return Math.sqrt(a * a + b * b);
}
var base = 3; var height = 4;
def hypotenuse = bind hypot(base, height);    // bind a la función
println(hypotenuese);                         // 5.0
base = 30; height = 40;
println(hypotenuse);                          // 50.0
```

Quizá no sea suficiente para una dependencia el realizar la actualización cuando varíe alguno de los argumentos de entrada, también puede ser necesario realizar la actualización si cambia alguna de las variables en el cuerpo de la función. Para resolver este problema existen las funciones *bound*, en las que cada expresión debe ser definida como *var* o *def*. Podemos observar la sintaxis en el siguiente ejemplo:

```
bound function getTotal(): String {
     def item1 = if (itemOne.selected) .75 else 0;
     def item2 = if (itemTwo.selected) .50 else 0;
     def item3 = if (itemThree.selected) .25 else 0;
     def total = item1 + item2 + item3;
     return "$ {total}";
}
def finalOrder = Text {
     content: bind getTotal()      // debe hacerse el bind aquí
     font: Font {
          size: 18
     }
}
```

Si no se aplica el contexto bind en la llamada a la función, ésta funcionará como una función ordinaria.

3.3.9 Clases y Objetos Literales

Más allá de los tipos predefinidos, JavaFX permite crear tipos de datos en las aplicaciones. Esta potente herramienta es el concepto en el que se basan las técnicas orientadas a objetos que tan bien funcionan en soluciones complejas del "mundo real". La mayoría de lenguajes orientados a objetos usan funciones especiales de las clases llamadas constructores para crear instancias de las clases. Sin embargo, en JavaFX, para crear instancias de las clases se utilizan expresiones de objeto literal. En este ejemplo podemos ver la creación de la clase *Point* y su instanciación como un objeto literal:

```
public class Point {
     public var x: Number;
     public var y: Number;
     public function clear(): Void {
           x = 0;
           y = 0;
     }
}

// el objeto literal inicializa las variables de instancia
var p = Point { x: 10, y: 20 };
println("p = ({p.x}, {p.y})");    // p = (10.0, 20.0)
// un objeto literal vacío usará los valores por defecto para las
variables de instancia
var q = Point {};
println("q = ({q.x}, {q.y})");    // q = (0.0, 0.0)
p.clear();
println("p = ({p.x}, {p.y})");    // p = (0.0, 0.0)
```

Los Objetos Literales comienzan con un nombre de clase seguido por llaves conteniendo inicializadores con : separando las variables instanciadas de sus valores. Asignar un Objeto Literal a una variable permitirá más tarde llamar a funciones de ese objeto utilizando esa referencia. Un Objeto Literal vacío dará a las variables de instancia los valores por defecto.

Existen grandes ventajas creando los objetos como Objetos Literales: ver los valores de las variables de instancia en el mismo momento en que se construye el objeto, la posibilidad de usar un objeto literal o incluso una secuencia de objetos literales dentro de otro, crear objetos gráficos sin la necesidad de crear variables temporales innecesarias… Todo esto convierte a los objetos literales en una herramienta realmente útil para el diseño de interfaces gráficas o para otros problemas complejos.

3.3.10 Modificadores de Acceso

Para hacer los programas JavaFX de una forma modular y reutilizables es recomendable utilizar modificadores de acceso primarios en los scripts. Estos modificadores se aplicarán a todas las variables del script si no son sobrescritos mediante modificadores de acceso a variable.

Modificadores de Acceso Primarios:

- ***Package***: sólo accesible por dentro del paquete en que está definido (acceso lectura-escritura)
- ***Protected***: para funciones y variables de una clase, permiten el acceso a los herederos de la clase, para funciones y variables de un script, dentro del mismo paquete en que están definidas (acceso lectura-escritura)
- ***Public***: Accesible desde cualquier lugar (acceso lectura-escritura)

No existe un modificador de acceso *private*, sino que por defecto los scripts serán *script-private*.

Modificadores de Acceso a Variable:

- ***public-read***: acceso a escritura sólo en su propio script. Sólo lectura fuera del script.
- ***public-init***: acceso a escritura sólo en su propio script. Puede ser inicializado desde un objeto literal y leído desde fuera del script.
- ***protected public-read***: acceso a escritura sólo a través de subclases. En cualquier otro caso sólo lectura.
- ***protected public-init***: acceso a escritura sólo a través de subclases fuera del script. En cualquier otro caso sólo inicialización.
- ***package public-read***: acceso a escritura sólo en su mismo paquete. En otro caso sólo lectura.
- ***package public-init***: acceso a escritura sólo en su mismo paquete. En otro caso sólo inicialización.

3.3.11 Triggers

JavaFX dispone de otra herramienta que permite ejecutar código cuando las variables del programa cambian sus valores. Los *Triggers* son similares a las expresiones de binding, utilizan las palabras reservadas *on replace* y se pueden usar con variables, propiedades, secuencias y con bind. Entenderemos mejor el comportamiento y la sintaxis de los triggers en este ejemplo con variables en el que las sentencias del bloque se ejecutarán cuando se modifique el valor de la variable que llama al trigger:

```
var seq: Integer[];
var value = 0 on replace {
     if (value > 0) insert value into seq;
}
value = 12; value = -5;      // no hay valores negatives en seq
println(seq);                // [ 12 ]
value = 20;
println(seq);                // [ 12, 20 ]
```

Si fuese necesario utilizar el antiguo valor de la variable, esto puede lograrse incluyendo un nombre para la variable antes de la definición del bloque:

```
var number = 4 on replace old {
     println("old = {old} new = {number}");    // old = 0 new = 4
}
number = 8;                                    // old = 4 new = 8
```

3.3.12 Timelines

En JavaFX, modificar los valores de las propiedades de un objeto *Node*, actualizará automáticamente el escenario gráfico para ajustarse a esos nuevos valores. Para animar un *Node*, las propiedades comunes son *translateX* y *translateY* para mover el Node, *scaleX* y *scaleY* para escalar el tamaño de un *Node* en el eje X o Y, y *rotate* para realizar un giro del Node sobre su propio centro. Sin embargo es posible modificar y animar cualquier variable con acceso de escritura de un *Node*, para hacer esto, simplemente habrá que modificar las propiedades con los *Timelines*.

Un *Timeline* es, básicamente, una sucesión de *KeyFrames*, un *KeyFrame* especifica un retarde temporal (propiedad *time*) en que una acción o una animación (propiedad *action*) deben tener lugar. Es posible manejar el progreso de un *Timeline* mediante una serie de propiedades (*rate*, *autoReverse*…) y funciones básicas (*play*, *playFromStart*, *stop*, *pause*) que nos aporta la clase.

```
import javafx.animation.Timeline;
import javafx.animation.KeyFrame;
import javafx.animation.Interpolator;

Timeline {
keyFrames: [
     KeyFrame{
           time: 0s
           action: doSomething()
     },

     KeyFrame{
           time: 4s
           action: doSomething()
     }
]
}.play();
```

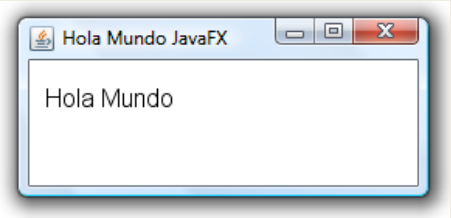
3.3.13 "Hola Mundo"

Un sencillo ejemplo demostrativo de la capacidad declarativa de JavaFX es el siguiente "Hola Mundo", comparativo entre Java Swing y JavaFX; en JavaFX es extremadamente sencillo, no solo crear objetos, sino también establecer sus propiedades.

Hola Mundo JavaFX:

```
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.text.Font;
import javafx.scene.text.Text;

Stage {
    title: "Hola Mundo JavaFX"
    scene: Scene {
        width: 250
        height: 80
        content: [
            Text {
                font : Font {
                    size : 16
                }
                x: 10
                y: 30
                content: "Hola Mundo"
            }
        ]
    }
}
```



Hola Mundo Java Swing:

```
import javax.swing.*;
import java.awt.*;

public class HolaMundoSwing {

    public static void main(String[] args) {
        JFrame frame = new JFrame("HolaMundoSwing");
        JLabel label = new JLabel("Hola Mundo");
        frame.getContentPane().add(label);

        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.pack();
           frame.setSize(new Dimension(300, 100));
        frame.setVisible(true);
    }

}
```

3.4 Creación de la Interfaz Gráfica

En la manipulación de objetos gráficos es donde encontramos el gran punto fuerte de JavaFX. Como se describe en la figura 3.3, en el nivel más alto para el manejo de los objetos gráficos se encuentra el *Stage*, el *Scene* contendrá los objetos mostrados en el escenario y la clase *Node* nos aportará toda una serie de herramientas que permitirán manejar los objetos gráficos de forma consistente.

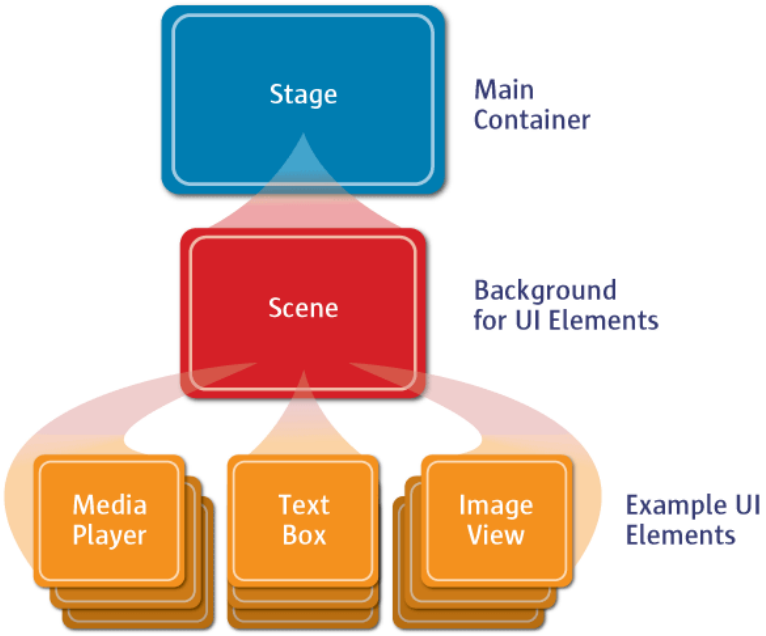


Figura 3.3: Jerarquía de Interfaz Gráfica

3.4.1 Stage

Todo programa JavaFX que pretenda manejar material gráfico debe incluir un *Stage* en su nivel más alto, el *Stage* contendrá un objeto *Scene*, que a su vez contendrá una secuencia de *Nodes*. Por defecto el *Stage* tiene estilo StageStyle.DECORATED, que será visto de distinta forma según el ambiente, (Windows Vista, Mac OS, Windows XP). StageStyle.UNDECORATED eliminará la decoración de la ventana.

3.4.2 Scene

La clase *Scene* es el root de todo el contenido en un escenario gráfico. El fondo de un *Scene* está especificado por la propiedad *fill* (por defecto Color.WHITE). La secuencia de nodos contenida por el *Scene* será mostrada en el escenario gráfico.

```
Stage {
title: "Stage Title"
width: 180
height: 150
style: StageStyle.UNDECORATED
scene: Scene {
          fill: Color.BISQUE
          content:                // contenido aquí
}
}
```

3.4.3 Node

La clase *Node* es la clase base para todos los objetos en el escenario gráfico. Se pueden añadir objetos *Node* (subclases de *Node*) al escenario, especificar sus propiedades y aplicarles transformaciones. Esta clase tiene muchas propiedades que permiten personalizar su aspecto y comportamiento, como la visibilidad, los manejadores de eventos, opacidad, rotación, visibilidad y otras muchas propiedades. Existen tres tipos diferentes de subclases de *Node*: Subclases predefinidas como *Shape* o *ImageView* con una funcionalidad predeterminada, la subclase *Group* aportará contenedores para más elementos tipo *Node*, y por último *CustomNode* nos permitirá personalizar nuestro propio tipo de *Node* según nuestras necesidades. Todo esto ajustado según la jerarquía de clases mostrada en la figura 3.4.

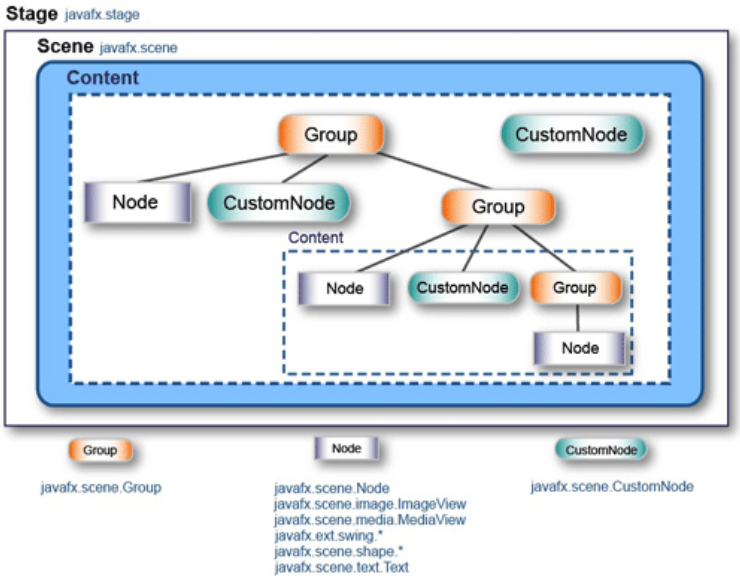


Figura 3.4. Jerarquía de nodos

3.4.3.1 Group y Container

Group es una clase contenedora de grupos de nodos que se unirán para tener unas características comunes. Lo más común es utilizar esta clase para manejar una porción del escenario gráfico que contendrá una serie de *Nodes* con una relación entre ellos. Una vez que la secuencia de *Nodes* ha sido insertada en el *Group*, es posible reposicionar el grupo sin afectar a la posición relativa de los *Nodes*. Además de construyendo el objeto *Group* con la secuencia de *Nodes* en el campo *content*, también es posible insertarlos de la siguiente forma:

```
var g1 = Group { };    // instancia el Group g1, con el content vacío
insert Rectangle {
     width: 40
     height: 20
} into g1.content;     // añade un Rectangle al content de g1
```

3.4.3.2 CustomNode

La clase *CustomNode* es una clase abstracta que extiende a la *Node* y que permite crear *Nodes* personalizados para los escenarios gráficos, al extender la clase *CustomNode* se debe proveer al programa de una función de instanciación que, normalmente retornara un nodo *Group* con una secuencia de *Nodes* en su campo *content*:

```
public class GuitarString extends CustomNode {
// propiedades, variables, funciones
. . .
protected override function create(): Node {
     // aquí cualquier código de inicialización
     return Group {
                 content: [
                 Rectangle { ... }
                 Rectangle { ... }
                 Rectangle { ... }
                 Text { ... }
           ]
     } // Group
}
} // GuitarString
```

3.4.3.3 Shape

Los objetos que hereden de la clase *Shape* heredarán todas las propiedades de un *Node* añadiendo unas nuevas como *fill* y *stroke*, que definirán propiedades de este elemento como el color de fondo, la opacidad, el gradiente, la línea del borde o la forma, de modo que tendrán todas las propiedades necesarias para convertirse en una figura 2D que podrá ser mostrada en un escenario gráfico.

3.4.3.4 ImageView

Para manipular imágenes, JavaFX aporta dos clases principales: la clase *Image* permite definir una imagen desde una URL (local o remota), mientras que la clase *ImageView* permite visualizar un objeto *Image*.

ImageView es de tipo *Node*, por tanto, todas las manipulaciones y transformaciones disponibles podrán serle aplicadas.

```
// cargar la imagen
var image = Image {
    url: "{__DIR__}flower.png"
}

// iv1 muestra la imagen tal y como es

var iv1 = ImageView {
    image: image
}

// iv2 cambia el tamaño de la imagen para que tenga un ancho de 100
puntos pero siga manteniendo la relación de aspecto
// método de filtrado de alta calidad, además de uso de cache para
mejorar el rendimiento

var iv2 = ImageView {
    image: image
    fitWidth: 100
    preserveRatio: true
    smooth: true
    cache: true
}
```

3.4.4 Componentes de Interfaz de Usuario

Los objetos gráficos deben incluir una serie de componentes de interfaz de usuario, JavaFX nos aporta estos componentes de dos formas distintas:

- JavaFX UI Controls: clases nativas de JavaFX, entre las que se incluyen *TextBox, Button, ListView, CheckBox, ScrollBar…*
- Componentes Java-Swing: modificados para adaptarse y ser consistentes sobre un escenario gráfico JavaFX, son incompatibles con el perfil *common* de JavaFX, entre estos componentes tenemos: *SwingButton, SwingComboBox, SwingIcon, SwingListItem…*

3.5 Otras Utilidades

En la API de JavaFX encontramos ciertas librerías que, aportándonos una mayor funcionalidad, nos permiten llegar más allá a la hora de cumplir unos requisitos en nuestras aplicaciones. Sin duda dos grandes ejemplos de esta gran suma de funcionalidades son las librerías para trabajar con tráfico Http y para almacenar datos de forma local. Para una mayor extensión de la información, consultar la API JavaFX en [Web JavaFX: API].

3.5.1 HttpRequest

Sin duda, JavaFX debe estar preparado para trabajar con conexiones web. En este aspecto, uno de los requisitos fundamentales es el uso de servicios web y peticiones Http. La clase *HttpRequest* cumple esta función permitiendo realizar peticiones *GET* y *POST* y, por tanto, capacidad para invocar servicios web bajo REST. *HttpRequest* nos provee de una gran cantidad de funciones para gestionar nuestras peticiones Http; apreciaremos mejor todas estas funciones en el siguiente ejemplo de petición Http mediante *GET* [Web JavaFX: API]:

```
def getRequest: HttpRequest = HttpRequest {
    location: "http://www.wikipedia.org";
    onStarted: function() {
        println("onStarted - started performing method:
{getRequest.method} on location: {getRequest.location}");
    }
    onConnecting: function() { println("onConnecting") }
    onDoneConnect: function() { println("onDoneConnect") }
    onReadingHeaders: function() { println("onReadingHeaders") }
    onResponseCode: function(code:Integer) { println("onResponseCode -
responseCode: {code}") }
    onResponseMessage: function(msg:String) {
println("onResponseMessage - responseMessage: {msg}") }
    onReading: function() { println("onReading") }
    onException: function(ex: java.lang.Exception) {
        println("onException - exception: {ex.getClass()}
{ex.getMessage()}");
    }

    onDoneRead: function() { println("onDoneRead") }
    onDone: function() { println("onDone") }
}

getRequest.start();
```

3.5.2 Resource y Storage

En una aplicación, normalmente es necesario el almacenamiento de la información para poder darle uso cuando esta aplicación es reiniciada. Tanto es así, que se hace obligatoria la implementación de una clase que soporte este tipo de operaciones. En JavaFX, este problema se hace más importante considerando que tratamos con entornos que no deben tener acceso a archivos locales. Con un funcionamiento similar al de las cookies, encontramos una solución a este problema en las clases Resource y Storage de la API JavaFX. Estas clases serán utilizadas para almacenar información de forma local sin necesidad de acceder a archivos del dispositivo. Como podemos apreciar en el siguiente ejemplo, se hace uso de *InputStream* y *OutputStream* para almacenar y extraer la información de un *Resource*.

```
import javafx.io.Storage;
import javafx.io.Resource;
import java.io.InputStream;

var entry = Storage {
   source: "myfile.txt"
}
var resource = entry.resource;
var inputstream = resource.openInputStream();
// usamos un inputstream para extraer información
inputstream.close();
```

4 Desarrollo del Proyecto

Una vez estudiadas las distintas tecnologías disponibles en el estado actual de la técnica y familiarizados con la familia JavaFX, resulta mucho más sencillo comprender la línea temporal que ha movido este proyecto desde un comienzo para acabar logrando los objetivos propuestos.

Para facilitar una completa comprensión de la totalidad del trabajo realizado, este resumen se divide en diversos capítulos con el fin de abordar la explicación del proyecto de igual forma que se aborda un proyecto de software: dividiendo la complejidad innata a la totalidad del proyecto en diversos módulos que nos permitan abstraernos del conjunto del programa para poder centrarnos en la solución de cada uno de los problemas por separado, haciendo más sencilla y manejable la comprensión del proyecto en su conjunto. De esta forma, podremos observar la solución a los distintos problemas presentados según la clase en la que este problema ha sido abordado, comprendiendo el recurso específico de la API JavaFX que ha permitido la implementación de cada problema concreto.

Ahora bien, una descripción subdividida exclusivamente en la problemática podría resultar insuficiente si se cayese en el error de obviar la línea de ejecución temporal que se ha seguido para la consecución de los objetivos. Una línea temporal que nos será aportada a través de la sección 4.3 sobre la evolución global del desarrollo del proyecto y de la sección 4.1 sobre los primeros pasos seguidos.

La descripción de la aplicación se completa con un caso de utilización de la aplicación final en el que un usuario interactuará con la misma para obtener los resultados deseados.

La totalidad de la implementación de la aplicación se ha realizado con ayuda de la API JavaFX release 1.2.1. A pesar de que ya existe una versión más actualizada de esta API, la coincidencia temporal con el desarrollo de este proyecto ha forzado el uso de la versión 1.2. Una versión que es posible consultar en [API JFX 1.2].

4.1 Primeros Pasos

Una exploración a fondo de la web oficial de JavaFX [Web Oficial JavaFX] y de la gran cantidad de videos, tutoriales y aplicaciones de muestra que se exponen en ella es fundamental para una primera toma de contacto con la tecnología que se aborda en este proyecto. Y no sólo esto, también resulta muy necesario conocer ciertos pormenores que enmarcan el desarrollo de una aplicación en JavaFX. Pasos como la instalación de un adecuado IDE y el entorno de ejecución JavaFX Mobile en la PDA o como la realización de unas primeras pruebas para asegurar que "todo funciona bien" resultan fundamentales antes de comenzar el desarrollo de una aplicación definitiva.

En esta sección se describirán los pasos básicos que se han realizado para comenzar la construcción de la aplicación con la certeza de que ésta podrá ser llevada a cabo con éxito, minimizando al máximo aquellos posibles contratiempos que, sin duda, es inevitable encontrar.

4.1.1 Primeras impresiones JavaFX

Tras un completo estudio de la tecnología JavaFX a través de la visualización de los videos explicativos, de los tutoriales disponibles y de las aplicaciones demostrativas de la potencia del lenguaje, no resulta complicado obtener una primera impresión sobre las principales características y puntos fuertes de la tecnología, puntos que deben ser explotados para un correcto estudio de la misma. Según la web [Web Oficial JavaFX], JavaFX nos brinda un sistema orientado al desarrollo de aplicaciones RIA con capacidad multi-dispositivo, alta potencia gráfica mediante un lenguaje declarativo y amplio soporte para servicios web y multimedia.

Conocidas estas capacidades generales de la tecnología JavaFX parece adecuado adentrarse en la sintaxis JavaFX Script, una sintaxis con unas características muy interesantes:

- El uso de lenguaje declarativo para la creación de las interfaces gráficas podría facilitar y potenciar la creación de una UI impactante.
- La herramienta *Timeline* aporta una gran funcionalidad para la animación de los objetos que se muestren en el escenario gráfico.
- El *binding* debería resultar altamente útil para muchos de los aspectos del programa, incluyendo la modificación de los objetos gráficos a través de las interacciones del usuario.
- Perfil de desarrollo *common* como soporte para multi-dispositivo.
- La gran conectividad web nos acerca a muchos de los recursos disponibles en internet.
- La alta disposición multimedia permite cargar y mostrar con facilidad imágenes y vídeos tanto locales como remotos.

Con estas primeras impresiones sobre la tecnología nos acercamos un poco más a la realidad de la potencia del lenguaje, teniendo una mejor visión sobre las posibilidades que JavaFX nos brinda para realizar el proyecto.

4.1.2 Instalación JavaFX Mobile

La aplicación a desarrollar debe estar disponible en todos los escenarios en que el estado actual de la tecnología nos permita: escritorio, navegador Web y dispositivo móvil. Es en este último escenario donde quizá resulte más complicado llegar a tener la seguridad de que va a ser posible la ejecución real de la aplicación. A pesar de que el IDE Netbeans aporta un emulador de dispositivo móvil que puede ser muy útil a la hora de desarrollar la aplicación, parece fundamental la comprobación de que, realmente este escenario de ejecución es real sobre un dispositivo físico.

Según el objetivo final de Sun, JavaFX debería estar disponible para aquellos dispositivos móviles que cuenten con Windows Mobile, Symbian OS o Android. Sin embargo, gracias a la web oficial de JavaFX y al documento [JFX 1.2 Release Notes, 2009], descubrimos que, en la actualidad, JavaFX sólo se encuentra disponible para Windows Mobile en su versión 6.0 y 6.1.

Se muestra fundamental para el proyecto obtener una PDA con Windows Mobile 6 instalado, sin embargo, nuestro único recurso es una PDA Qtek S200 con sistema operativo Windows Mobile 5. Resulta, por tanto, necesaria la instalación del nuevo sistema operativo mediante una serie de operaciones sobre la ROM que ponen en riesgo la integridad del dispositivo y que se pueden encontrar descritas de manera informal en [Web ROM WM6]. Una vez actualizado el SO del dispositivo Windows Mobile 6.1, tan sólo queda instalar el entorno JavaFX Mobile para comprobar definitivamente su capacidad de ejecución móvil. Siguiendo los pasos descritos en [Win Mob Install Guide, 2009], JavaFX Mobile queda correctamente instalado, incluyendo además, una serie de aplicaciones de muestra que dan prueba definitiva del comportamiento de JavaFX sobre este escenario.

4.1.3 Elección del adecuado IDE

El sistema JavaFX dispone de soporte para desarrollo de aplicaciones mediante un plugin disponible en los dos entornos de desarrollo integrado más importantes: Eclipse y Netbeans. En un principio, se opta por el uso de Eclipse, una gran herramienta que muestra un comportamiento excelente en todos los aspectos necesarios para el desarrollo de aplicaciones JavaFX, incluyendo soporte drag-and-drop y emulador móvil.

Sin embargo, ciertos problemas a la hora de la creación de un archivo ejecutable JavaFX, debidos quizá a la falta de soporte técnico para el plugin de este IDE y al desconocimiento del mismo, resultaron determinantes para el cambio de la elección a Netbeans, el IDE que cuenta con todo el soporte de Sun y que resolvía todos estos problemas gracias, no a un mejor planteamiento de las soluciones, sino a una mayor información sobre su utilización.

4.2 Objetivos y requisitos de la aplicación

Partiendo de las aplicaciones de video-vigilancia ya existentes en el estado actual de la técnica recogemos una serie de requisitos que podrían formar parte de los objetivos a cumplir por nuestro proyecto:

- Soporte para cámaras web y cámaras IP.
- Reconocimiento de formas.
- Capacidad para soportar varias cámaras.
- Hora y fecha incrustadas.
- Detección de movimiento.
- Notificación de eventos.
- Grabación continua y controlada por eventos.
- Monitoreo de audio.
- No instalación de software.
- Historial de video.
- Niveles de acceso configurables.
- Control manual de la grabación.

Haciendo una valoración sobre la complejidad de estos objetivos para su desarrollo en JavaFX y sobre su impacto en el usuario final, recopilamos los objetivos finales de la aplicación. Objetivos que se basarán en buena parte en explotar las mayores ventajas y funcionalidades de JavaFX, para de esta forma lograr una aplicación con la funcionalidad básica de las anteriores, pero con una serie de mejoras y ventajas sobre ellas que justifiquen su desarrollo.

Dejando abierta la posibilidad de la posterior implementación del resto de objetivos, estos serán finalmente los requisitos a cumplir por nuestra aplicación final:

- Soporte para cámaras web y cámaras IP.
- No instalación de software.
- Manutención de una sola aplicación para los distintos entornos de ejecución.
- Soporte Desktop multi-plataforma.
- Monitorización en dispositivo móvil.
- Monitorización online.
- Capacidad para soportar varias cámaras.
- Mapa de cámaras.
- Adaptabilidad respecto a una posible utilización en circunstancias variables de uso de tecnologías de vídeo o mapas físicos.
- Capacidad para ser configurable por un usuario local o de forma remota a través de un servidor.

Una vez reconocidos los requisitos básicos de la aplicación, tendremos la ocasión de comprobar cómo se han cumplido estos objetivos a través del estudio de la solución final y del uso que ésta hace de las herramientas JavaFX.

4.3 Evolución global del proyecto y problemas encontrados

Tras la evaluación de los requisitos y tomar las medidas oportunas para la comprobación de los medios óptimos de desarrollo y ejecución parece adecuado comenzar con la producción de la aplicación. Durante todo el proceso de desarrollo se hace básico disponer de una conexión con un servidor web con distintos recursos disponibles, encontramos una solución en la instalación y configuración de un servidor Apache: Xampp para Linux [Web Oficial Xampp].

El problema fundamental que presenta el proyecto, y por tanto, el primero en ser abordado, es la monitorización de las cámaras. JavaFX presume de tener grandes capacidades multimedia, por lo que, en principio, no debería haber mayor problema en este aspecto. Las clases *media* y *mediaBox* proporcionadas por la API de JavaFX parecen idóneas para nuestro objetivo según puede verse en la aplicación de ejemplo [Web JavaFX: Media]. Por otra parte, el codificador de vídeo oficial para JavaFX, On2 Flix Encoder [Web Oficial On2] permite codificar los formatos más comunes de vídeo como AVI o MPEG a formatos aceptados por JavaFX: el formato Flash Video (FLV) y el oficial de JavaFX (FXM). Tras la descarga de una versión de prueba de 30 días del On2 Flix Encoder para realizar una codificación de vídeo en directo a formato FXM y reproducirla en una sencilla aplicación JavaFX encontramos los primeros problemas: Flix Encoder no codifica vídeos para streaming en directo, únicamente videos previamente guardados.

La solución podría estar en otra aplicación de On2: Flix Live, un programa capaz de realizar codificaciones para streaming de vídeo en directo, aunque eso sí, no a formato FXM sino a FLV. Este obligado cambio de formato podría parecer una simple limitación anecdótica pero, tras alguna prueba comprendemos que, lejos de ser una limitación, es sencillamente un síntoma de otro problema más importante: JavaFX no soporta streaming de vídeo en directo. Lo que a priori podría parecer un callejón sin salida para el proyecto acaba resultando en un simple cambio de enfoque: es posible tener la sensación de streaming de vídeo gracias a una recarga continua de imágenes consecutivas, lo cual, además abre una nueva serie de posibilidades como la elección de una determinada velocidad de fps según nuestra elección.

Salvado el inconveniente de obtener una transformación de vídeo en directo a imágenes consecutivas gracias al programa Mplayer [Web Oficial Mplayer], tan sólo queda la comprobación de que estas imágenes pueden realmente ser recargadas en una aplicación JavaFX obteniendo la sensación real de streaming de vídeo. Una sencilla aplicación de prueba basada en un *Timeline* dio el resultado deseado, dando como efectiva está solución al problema y dejando como último escollo a resolver el cambio consecutivo de nombre que Mplayer da a las imágenes que exporta.

Dos recursos podrían resolver esta situación:

- Hacer uso de un servicio web para que el servidor informe al cliente del número de secuencia que debe cargar en un principio, y modificar ese número de forma sucesiva para cargar la imagen adecuada en cada momento.
- Aprovechar la licencia GPL de Mplayer para recompilar el código fuente modificando las líneas necesarias para que el nombre de la imagen se mantenga constante y solo sea necesaria la continua recarga de una imagen con un nombre predeterminado.

Ya que la segunda de las posibles soluciones brindaría una mayor sostenibilidad a la aplicación y a todo el sistema, se decide efectuar la recompilación. Tras un estudio del código fuente se comprueba que sólo es necesaria la modificación de la línea (línea 269, archivo "vo_jpeg", carpeta "libvo" del código fuente de Mplayer) que cambia consecutivamente el nombre del archivo de imagen exportado:

```
snprintf(buf, BUFLENGTH, "%s/%s/%08d.jpg", jpeg_outdir, subdirname, framenum);
```

Por esta otra que exporta y sobrescribe todas las imágenes con nombre "cam.jpg":

```
snprintf(buf, BUFLENGTH, "%s/%s/cam.jpg", jpeg_outdir, subdirname);
```

Una vez recompilado el Mplayer y comprobado el correcto funcionamiento de todo el proceso de monitorización se da este problema por resuelto.

Un siguiente paso resulta de la creación de un mapa de selección de cámara, de manera que sea posible seleccionar la cámara a visualizar mediante su posición en el mapa. La creación de este tipo de mapas debería cumplir una serie de requisitos para satisfacer las necesidades del usuario final que haga uso de ellos:

- Cada cámara debe indicarse mediante un icono en el mapa, pero estos iconos no deben formar parte del plano de fondo, ya que podrían ser modificables según lo hagan las cámaras en su posición física real.
- Debe ser posible realizar un desplazamiento por el mapa para localizar la cámara adecuada, de forma que la imagen se moverá con las interacciones del usuario.
- Sería adecuada la disponibilidad de una herramienta de zoom para modificar la escala del mapa según la voluntad del usuario.

Estos requisitos traen consigo la necesidad de establecer cada icono de cámara como un objeto independiente en el mapa, con unas coordenadas que lo sitúen en el mismo. Pero todos estos objetos de imagen (tanto los iconos como el plano) deben desplazarse y escalarse en conjunto, objetivo que cumplimos situándolos como contenidos en un nodo *Container*, un contenedor que mantiene la posición relativa de los objetos permitiendo una transformación y desplazamiento de la totalidad de ellos como conjunto respecto al resto de nodos del escenario gráfico.

Para permitir el desplazamiento del mapa se hace uso de la herramienta de *binding* para modificar los parámetros que contienen la información de la posición del nodo respecto al resto del escenario. Siempre teniendo en cuenta el desplazamiento máximo que debe ser posible para cada imagen en función de su tamaño. En la solución se implementan dos maneras de crear desplazamiento del mapa:

- Unas flechas de dirección, que modifican estos parámetros en base a la pulsación de la flecha correspondiente a la dirección del desplazamiento deseado.
- Un mecanismo para pulsar y arrastrar la imagen, que también provocará el desplazamiento consecuente de la imagen gracias al evento de *dragging* que registra JavaFX. Una solución muy adecuada sobre todo para entornos táctiles.

El zoom de la imagen puede conseguirse gracias a las herramientas de escalado que tienen en común todos los nodos en JavaFX, aplicándolas adecuadamente al *Container* y a través del binding de las variables con respecto a las pulsaciones en los botones de zoom. Con esto, el mapa quedaría listo.

Sin duda, uno de los aspectos que mayor funcionalidad dan a una aplicación es su capacidad para ser personalizable y configurable según las circunstancias. Es, por tanto, en este aspecto donde se centraron a continuación los esfuerzos para el desarrollo.

En un principio podría parecer que un fichero *properties* de configuración podría resolver el problema de forma similar a como podría hacerse en Java, sin embargo, JavaFX, al estar orientado a la creación de RIAs no permite el acceso a archivos locales para garantizar la seguridad de los usuarios finales. Lejos de no prever la posibilidad de querer almacenar datos de forma local, JavaFX nos da un soporte magnífico para almacenamiento de información a través de las clases *Storage* y *Resource*. Gracias a estas dos clases es posible guardar una configuración en el cliente utilizando un sistema similar al de las *cookies*.

Una primera prueba de este mecanismo de almacenamiento consiste en la introducción mediante un *TextBox* de la IP del servidor de las imágenes. Prueba que resulta todo un éxito. Más difícil parece aprovechar este mecanismo de almacenamiento de una configuración para conseguir mapas personalizables, sin embargo, varias ideas podrían permitir alcanzar este objetivo:

- Crear un editor de mapas en la propia aplicación que permitiese cargar una imagen local y fijar la posición de cada cámara en este plano. De esta forma en cada copia del programa sería posible disponer del mapa necesario mediante una edición absolutamente local y descentralizada.
- Manejar unos archivos de configuración de mapa en el servidor de imágenes, estos archivos deberían contener toda la información de los mapas: la dirección URL de la imagen que serviría de plano, las coordenadas de la situación de las cámaras en el plano y un nombre descriptivo de cada cámara.

Esta última solución permite centralizar la edición de los mapas de modo que sólo sería necesario mantener o editar un mapa en el servidor para todas las aplicaciones cliente que quieran conectarse a dicho servidor. Fundamentalmente por este aspecto parece la solución más apropiada, a pesar de requerir una mayor conectividad con el servidor.

Ya comprobada la efectividad del uso de *Resource* y *Storage* para almacenar datos introducidos por el usuario de forma local, queda confirmar esta efectividad a la hora de almacenar datos desde un recurso remoto. Es aquí donde entra en escena la clase *HttpRequest* para peticiones http de JavaFX. Utilizando los métodos aportados por esta clase se hace posible la descarga de un fichero de pares alojado en el servidor, con toda la información sobre los mapas. Fichero que sería inmediatamente almacenado localmente y

leído para, a través de los pares dato-valor obtener la total descripción de un mapa y las cámaras que éste contiene. Aprovechando la viabilidad de este sistema, se puede introducir también un fichero de configuración general en el servidor que permitiría soporte, no sólo para un único mapa, sino para el número total de ellos que sea necesario utilizar.

En este punto, tenemos la certeza de que nuestra aplicación tendrá soporte para monitorización, mapa de cámaras y configuración general. Encontramos un siguiente paso a seguir en la unificación de ambos aspectos en una misma aplicación que, evidentemente, debería permitir un cambio total de los objetos mostrados según las interacciones del usuario. Para esto, se define un controlador similar al que se nos describe en el modelo Model-View-Controller [Steve Burbeck, 1992], con la diferencia de que este controlador gestionará también los datos de las diferentes vistas.

Probada la eficacia de este controlador para modificar las vistas y los objetos gráficos con los que el usuario interacciona se hace un diseño de la totalidad de la aplicación, incluyendo las diversas opciones disponibles según las diferentes vistas: se decide que la aplicación definitiva dispondrá de un visualizador, un menú de configuración, un listado de las cámaras disponibles y un mapa para la selección de la cámara a visualizar. También formarán parte de la aplicación una pantalla de bienvenida que prepare las configuraciones y cargue las diferentes opciones al inicio y una pantalla de visualización de errores.

Para el control de estos errores se hace uso del manejo de excepciones que presenta JavaFX, incorporando ciertas excepciones que JavaFX no es capaz de detectar. Debido a la propia naturaleza de la aplicación y al uso de recursos externos, se hace necesario un control sobre la auténtica disponibilidad de estos recursos, alertando de forma controlada al usuario en caso de producirse un error de este tipo.

Todo este proceso de desarrollo lógico y funcional se ve completado mediante la creación de la interfaz gráfica de usuario. JavaFX no sólo aporta una gran cantidad de herramientas para diseñadores y facilita su unión con los desarrolladores, sino que también simplifica la creación gráfica a través de elementos propios del lenguaje, como los objetos gráficos de figuras básicas. Se decide hacer uso de estos objetos para dar interactividad al usuario, pero además, estas formas primitivas estarían cubiertas por otra capa que daría el verdadero aspecto final del programa, de esta forma sólo sería necesario diseñar ciertas imágenes que se adapten a las formas básicas. Una solución que, siendo extremadamente sencilla, permite una total actualización de toda la interfaz gráfica con un esfuerzo mínimo. Dejando a la aplicación con un aspecto totalmente distinto según nuestra voluntad en el diseño y rediseño de la misma.

En los siguientes capítulos podemos encontrar una explicación ampliamente detallada de la solución final que se ha dado a cada problema de forma independiente, dejando a un lado el eje temporal para centrarnos en los pormenores sintácticos y técnicos que han permitido la solución final.

4.4 La Aplicación

La descripción de la aplicación se ha subdividido utilizando las mismas clases del programa que ya de por sí hacen un correcto reparto del trabajo para la consecución de los objetivos. De esta forma, podremos observar la solución a los distintos problemas presentados según la clase en la que este problema ha sido abordado:

VistaConfig nos permite comprender cómo se hace uso de peticiones al servidor web para obtener la adecuada configuración y de las clases *Storage* y *Resource* para el almacenamiento de dicha configuración de forma local.

VistaVisor podría ser la clase más importante para el correcto funcionamiento de la aplicación ya que de ella depende el visionado de las cámaras. En esta clase comprenderemos el uso de un *Timeline* para poder hacer una recarga continua de la imagen a mostrar de forma que tengamos una auténtica sensación de streaming de vídeo.

VistaError gestiona de forma controlada todos aquellos errores que puedan producirse durante la ejecución de la aplicación, la mayoría de ellos debidos a errores externos como la no disponibilidad de un recurso en el servidor.

VistaMapa permite la visualización de un mapa en el que se muestran todas las cámaras disponibles en su punto concreto de disposición en el mapa para poder seleccionar la adecuada a monitorizar. Esta clase muestra el uso de *ImageView* mediante la utilización de herramientas como escalado de la imagen para conseguir efecto zoom y translaciones para el desplazamiento a través del mapa.

VistaListado hace uso de sintaxis propia de las secuencias JavaFX para introducir los datos descriptivos de las cámaras disponibles en un objeto *ListView* que muestre un listado de las distintas cámaras para la selección de una concreta para su monitorizado.

VistaPrincipal carga todas las vistas anteriores desde el inicio del programa para un mejor rendimiento del mismo.

A su vez, el núcleo de la aplicación está compuesto por la clase ***Main***, primera en cargarse y responsable del *Stage* principal y los perfiles de ejecución, la clase ***Controller*** maneja los datos y la muestra en pantalla de la vista adecuada según las interacciones del usuario. También existe una clase abstracta ***Vista*** de la que heredaran todas las vistas anteriormente descritas. Podemos ver la relación existente entre las clases y su inicialización en ejecución en los diagramas de las figuras 4.1 y 4.2 respectivamente

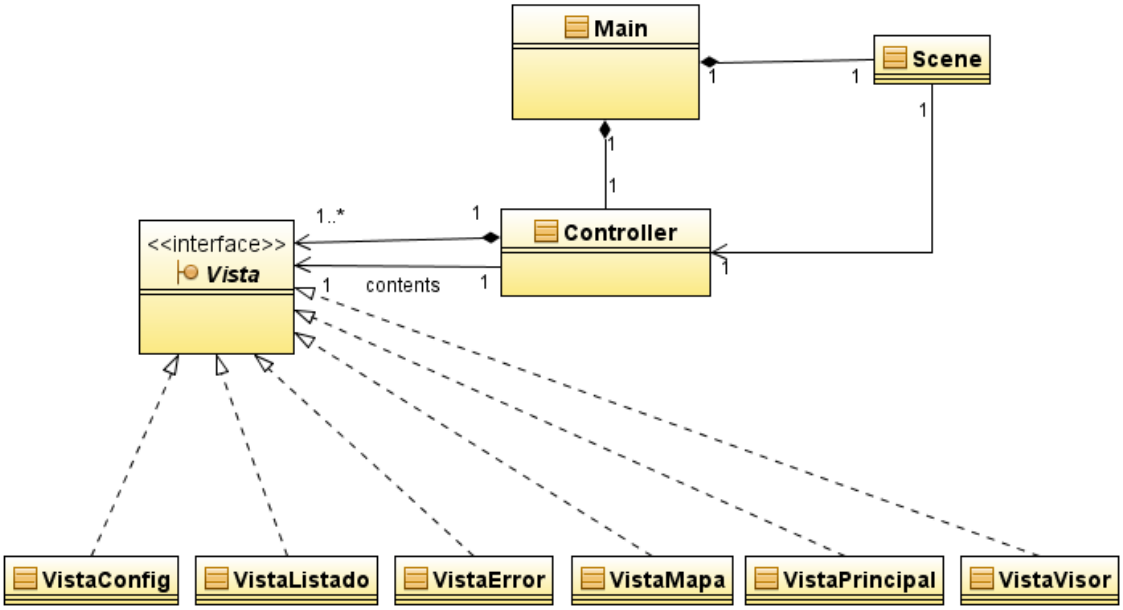


Figura 4.1: Diagrama de Clases

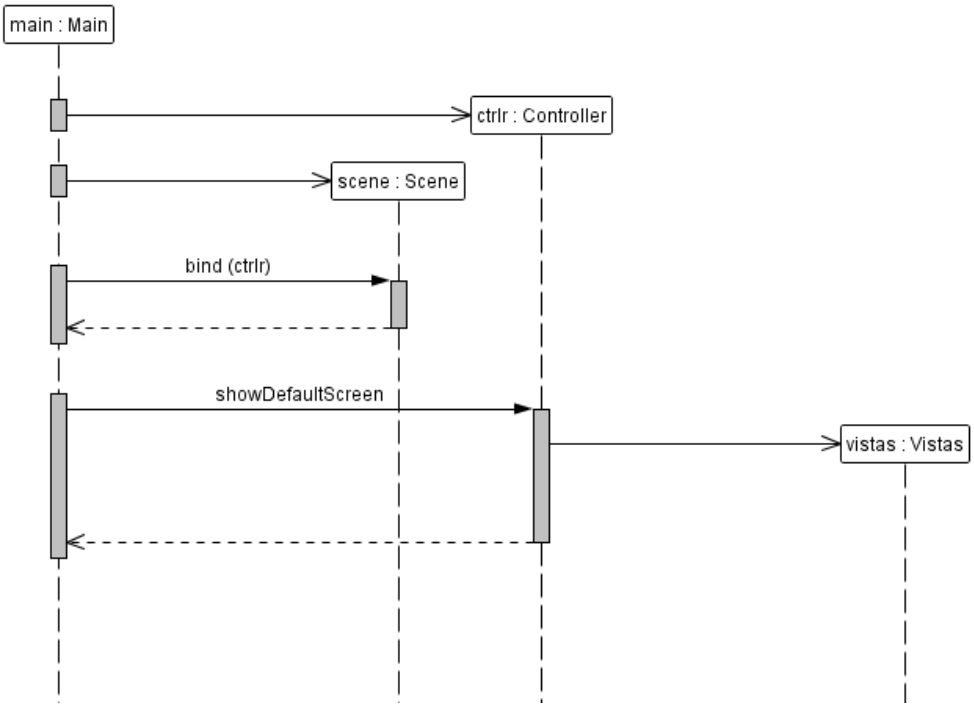


Figura 4.2: Diagrama de Secuencia de Inicialización

Resulta igualmente necesaria una descripción detallada de la creación de la interfaz gráfica de usuario que maneja el programa, además de un pequeño repaso de la utilización de los recursos externos a la aplicación como el servidor web Apache y el alojamiento de las imágenes de vídeo en el servidor a través de Mplayer. La totalidad del código de la aplicación se encuentra recogida para su consulta en el Anexo A.

4.4.1 Núcleo de la aplicación

El núcleo de la aplicación da forma a la estructura interna de la misma, una implementación adecuada de este núcleo es fundamental para el correcto desarrollo del resto de la aplicación. Esta estructura se basa en un manejador de datos y escenarios gráficos ***Controller***, que será capaz de cargar las diferentes vistas requeridas según las interacciones del usuario. Para asegurar una correcta conexión de los escenarios gráficos con este controlador, se crea una interfaz ***Vista***, con la función básica de cargar los objetos gráficos adecuados en un ***Panel*** contenido por el ***Controller***. Como todas las aplicaciones JavaFX, la clase ***Main*** contiene el *Stage* principal de la aplicación y mantiene la posición en pantalla de la aplicación. Además se encarga de llamar al controlador para que éste cargue la vista principal de la aplicación ***VistaPrincipal***.

4.4.1.1 ***Main:*** perfiles de ejecución, *Stage* y *Scene*

Al ser la primera clase en cargarse, la clase ***Main*** se encarga de mantener ciertos aspectos fundamentales de la aplicación como el *Stage* y *Scene* principales. Sin embargo, tras llamar al ***Controller***, deja a éste todo el control sobre la aplicación y se mantiene en un segundo plano. Las primeras variables a crear en la aplicación (*mobile* y *browser*) que vemos en el código 1, se utilizarán posteriormente para comprobar el escenario en que se ejecuta la aplicación:

```
var mobile: Boolean = "{__PROFILE__}" == "mobile";
var browser: Boolean = "{__PROFILE__}" == "browser";
```

Código 1: Variables de perfil

Esto permite que el comportamiento de la aplicación pueda ser diferente según las capacidades de los distintos ambientes o dispositivos en que ésta sea ejecutada, en el código 2, el *dragging* o arrastre de uno de los objetos gráficos sólo provocará el cambio en las variables *X* e *Y* si se ha producido desde un entorno de escritorio, ya que estas variables contienen la información sobre la posición en pantalla de la aplicación y no tiene sentido el desplazamiento de ésta en un dispositivo móvil o un navegador web:

```
onMouseDragged: function(me: MouseEvent) {
    if (not controller.browser and not controller.mobile ) {
        controller.X = me.screenX - me.dragAnchorX;
        controller.Y = me.screenY - me.dragAnchorY;
    }
}
```

Código 2: Uso de variables de perfil

El *Stage* (código 3) tan sólo contiene información sobre su título y posición en la pantalla, posición que, como hemos visto en el código 2, puede ser modificada por el usuario mediante un arrastrado que recogerán las variables *X* e *Y* del ***Controller***. Mediante una operación de *binding*, la posición del *Stage* variará en tiempo de ejecución. El *Scene* cargará un fondo con las medidas adecuadas de la aplicación y el contenido que el controlador le indique también en tiempo de ejecución.

```
var stage: Stage = Stage {
    x:bind controller.X;
    y:bind controller.Y;
    title: "Videovigilancia UPCT"
    style: StageStyle.UNDECORATED
    scene: Scene {
        width: Vista.screenWidth
        height: Vista.screenHeight
        content: bind [fondo, controller.contents]
    }
}
```

Código 3: *Stage* y *Scene*

4.4.1.2 ***Controller:*** Controlador de datos y escenarios gráficos

Para manejar los datos y los diversos escenarios gráficos, el controlador hace uso de la interfaz ***Vista***, una clase abstracta que sirve como base para todos los escenarios gráficos que necesitemos en la aplicación, de forma que implemente las variables y funciones comunes a todas las vistas para que el controlador pueda cargarlas adecuadamente. Como vemos en el código 4, no necesita más que una función que sobrescribe la de creación propia de un CustomNode (superclase de ***Vista***) de manera que devolverá al controlador un nodo contenedor de todos los objetos gráficos que la vista contenga.

```
public abstract class Vista extends CustomNode {
    protected var controller: Controller;
    protected var view: Node;
    protected abstract function createView(): Void;
    public override function create(): Node {
        createView();
        return view
    }
}
```

Código 4: Interfaz *Vista*

El contenido que será mostrado se almacena en la variable *contents* (contenida en el *Scene* del código 3). Una serie de funciones del controlador permiten la modificación en tiempo de ejecución de la variable como se observa en el código 5:

```
public var contents: Node;
 …
public function showVistaMapa() {
    if (vistaMapa == null) {
        vistaMapa = VistaMapa {
            controller: this
        }
    }
    vistaMapa.crearMapa();
    contents = vistaMapa;
    vistaActual=vistaMapa;
}
```

Código 5: *Contents*

Todas las vistas poseen una serie de pestañas que permiten el cambio de vista modificando los *contents* del controlador según las decisiones del usuario. En el código 6 podemos ver una de estas pestañas, en concreto la que al ser pulsada invocaría a la función del código 5 cambiando el valor de la variable *contents* del controlador mostrando la ***VistaMapa***.

```
var mapa: Rectangle = Rectangle {
    width: 240/4
    height: 28
    onMouseClicked: function(me: MouseEvent) {
        controller.showVistaMapa();
    }
}
```

Código 6: Pestaña para mostrar la *VistaMapa*

De esta forma, cuando un usuario presiona una pestaña para el cambio de vista, la vista que en ese momento se encuentre activa indicará al controlador que debe modificar la que debe mostrar mediante la invocación de la función correspondiente, vemos un ejemplo en ejecución en el diagrama de la figura 4.3.

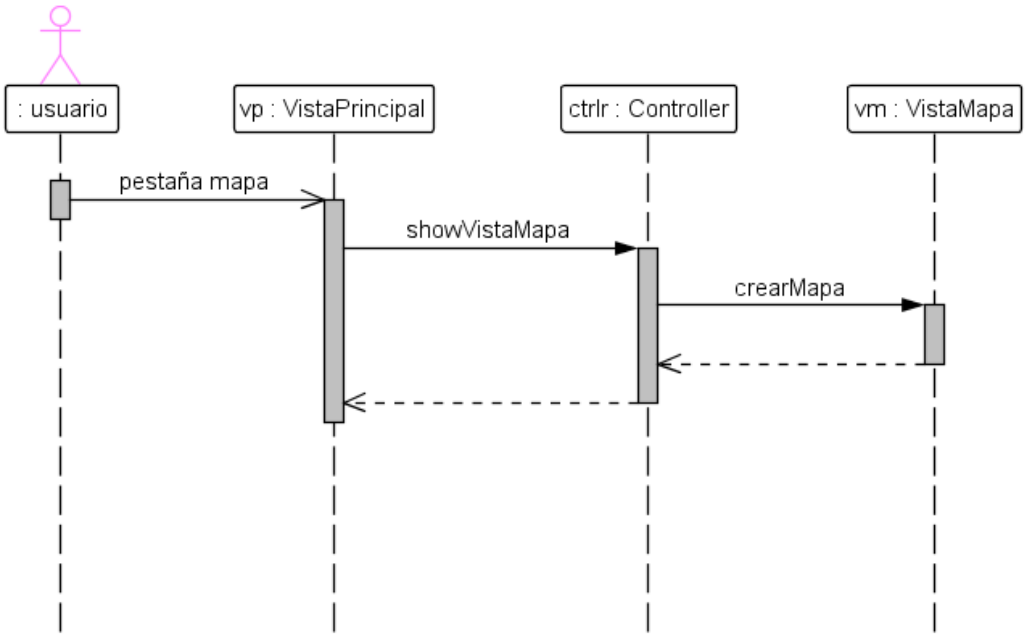


Figura 4.3: Diagrama de secuencia de cambio a *VistaMapa*

Como manejador de datos el ***Controller*** mantiene una serie de variables que cada **Vista** irá modificando y obteniendo según sus necesidades de modo que la comunicación entre las distintas vistas se efectúa a través de este controlador. Por esto se aportan una serie de variables necesarias para las vistas y sus métodos de obtención y actualización, encontramos un ejemplo en la variable *cam_actual*, que guarda la cámara en uso por el visor según lo que decidamos en el mapa o el listado de cámaras. En la figura 4.4 vemos el correspondiente diagrama de secuencia mientras que en el código 7 se muestra esta variable en el controlador:

```
public var cam_actual: Integer = -1;
 …
public function setCam(cam: Integer):Void{
    cam_actual = cam;
}
public bound function getCam():Integer{
    return cam_actual;
}
```

Código 7: Variable *cam_actual* en el controlador

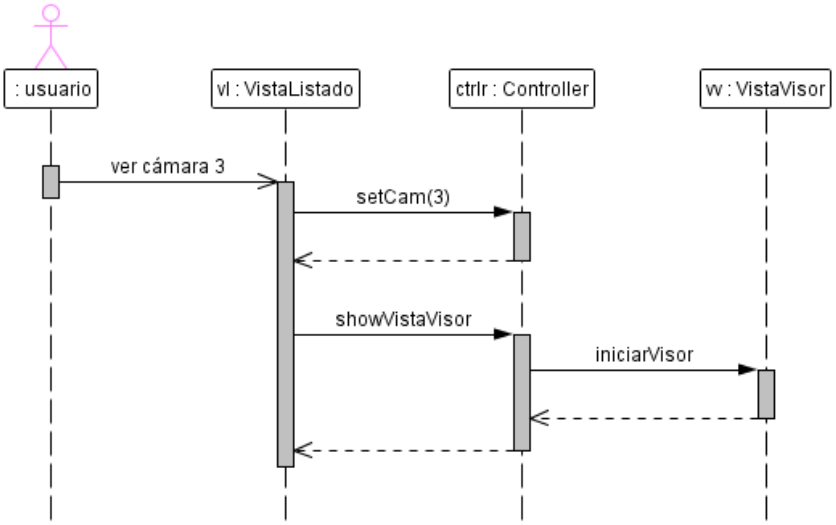


Figura 4.4: Diagrama de secuencia de selección de cámara

La variable podrá ser igualmente modificada por el mapa o por el listado de cámaras y será actualizada automáticamente en el visor (gracias a su condición de función *bound*) para monitorizar la cámara adecuada como muestra el código 8.

```
var cam_actual: Integer= bind controller.getCam();
```

Código 8: Variable *cam_actual* en la *VistaVisor*

4.4.2 VistaPrincipal - Inicio de la Aplicación

La pantalla de bienvenida de la aplicación no tiene ninguna funcionalidad práctica de cara al usuario, pero sí logística de cara del rendimiento de la aplicación: como recoge el código 5, el controlador llama a la función de creación de una vista sólo si ésta no ha sido ya creada. Para mejorar el rendimiento en la ejecución del programa, desde un principio se crean todas las vistas y mientras se muestra esta pantalla de bienvenida que da acceso a la selección de todas las demás. En el código 9 vemos cómo el ***Controller*** carga las vistas y muestra la ***VistaPrincipal*** mediante su función *showDefaultScreen* (invocada por el ***Main*** del programa):

```
public function showDefaultScreen() {

    vistaPrincipal = VistaPrincipal {
        controller: this
    }

    vistaConfig = VistaConfig {
        controller: this
    }

    vistaMapa = VistaMapa {
        controller: this
    }

    vistaListado = VistaListado {
        controller: this
    }
```

```
        vistaVisor = VistaVisor {
            controller: this
        }
        contents = vistaPrincipal;
    }
```

Código 9: Mostrar *VistaPrincipal*

4.4.3 VistaConfig - Configurar la Aplicación

El escenario de configuración de la aplicación nos ofrece la posibilidad de almacenar una IP con la que conectar remotamente con el servidor de vídeo, una vez que esta dirección IP ha sido almacenada una vez, la aplicación la carga en su inicio y sólo será necesario modificarla si también lo hace la dirección IP del servidor. Podremos introducir esta IP en el campo de texto reservado para ello según el código 10:

```
    public var argumento_ip: TextBox = TextBox {
        translateY:47.5
        translateX:72
        onMouseClicked: function(e: MouseEvent):Void{
            argumento_ip.text = argumento_ip.promptText;
        }
    }
```

Código 10: Introducir IP

Este *TextBox* muestra la IP previamente introducida, y, una vez clicado, permitirá su modificación. Para activar la IP introducida sólo será necesario pulsar el botón destinado para ello:

```
    var ok_ip: Button = Button {
        cursor: Cursor.HAND
        text: "OK"
        translateY:47.5
        translateX:176
        action: function() {
            storeIp(false);
        }
    }
```

Código 11: Botón para introducir IP

Al pulsar el botón del código 11 se llama a la función *storeIp*, una función que hace uso de las herramientas *Storage* y *Resource* que JavaFX propone como sistema de almacenamiento local de información. En el código 12 vemos como el texto introducido en el *TextBox* del código 10 se guarda como un par dato-valor en el *Resource* a través de un *OutputStream* que lo almacena como si de un archivo *Properties* se tratara.

```
    var storage_ip: Storage;
    var res_ip: Resource;
    res_ip = storage_ip.resource;
     …

    public function storeIp(clearold : Boolean): Void {
        if (checkStoreIp() == true){loadIp();}
        var outp: OutputStream = res_ip.openOutputStream(clearold);
        argumento_ip.commit();
        var str: String = argumento_ip.text;
        var properties = Properties{};
```

```
        if(str.trim().length() == 0 ) {
            outp.close();
            return;
        }
        properties.put("ip", str);
        properties.store(outp);
        outp.close();
        argumento_ip.text = "";
        loadIp();
    }
```

Código 12: Almacenar información de IP

La función *checkStoreIP* sencillamente comprueba si ya ha sido almacenada previamente una IP, mientras que *loadIp* carga los datos almacenados desde el *Resource* y los guarda en el ***Controller*** para poder utilizarse desde todas las vistas como muestra el código 13:

```
    public function loadIp() : Void {
        var inp: InputStream = res_ip.openInputStream();
        var properties = Properties{};
        var ip;
        properties.load(inp);
        if(properties.get("ip") != null){
            ip = properties.get("ip");
        }
        inp.close();
        controller.setIp(ip);
        ip_actual = ip;
        location lista="http://{ip}/config/config.txt";
        argumento_ip.promptText = controller.getIp();
    }
```

Código 13: Extraer información almacenada IP

Una vez se ha establecido la IP del servidor, el siguiente paso consiste en extraer el archivo de configuración remoto con la lista de los mapas disponibles, la función *iniciarPeticionLista* mostrada parcialmente en el código 14 realiza una petición Http al servidor indicado en la variable location_lista (modificada en el código 13) pidiéndole el archivo a través de un objeto *HttpRequest*. Una vez haya obtenido respuesta almacenará los datos de forma similar al almacenamiento de la dirección IP.

```
    function iniciarPeticionLista():Void{
        var getRequest_config: HttpRequest = HttpRequest {
            location: bind location_lista
            method: HttpRequest.GET
             …
            onInput: function(is: InputStream) {
                var properties = Properties{};
                checkStoreLista();
                var outp: OutputStream =
                    res_lista.openOutputStream(false);
                properties.load(is);
                properties.store(outp);
                outp.close();
                is.close();
                loadLista();
            }
        }
```

```
        getRequest_config.start();
    }
```

Código 14: Petición Http de lista de mapas

Una vez disponemos de la información sobre los mapas disponibles, podremos seleccionar uno de ellos. Mediante otra petición Http el mapa escogido será descargado y almacenado en un *Resource* disponible para su gestión desde la ***VistaMapa*** y la ***VistaListado***, clases encargadas de proveer un método de selección de cámara.

4.4.4 VistaListado - Listado de las Cámaras

La clase ***VistaListado*** nos propone una manera de seleccionar la cámara que queremos visualizar basada en una lista descriptiva de todas ellas. El archivo descargado de mapa nos aporta información sobre la situación de las cámaras, dando a cada una un nombre que la identifique sobre las demás. Como explica el código 15, ***VistaListado*** aprovecha esta descripción para exponer en un objeto *ListView* todas las cámaras disponibles para su selección. El contenido del *ListView* se indica en el atributo ítems, este código hace uso de la potencia *bind* y de las secuencias JavaFX para mantener actualizada la lista de ítems a mostrar.

```
    var list: ListView = ListView {
        cursor: Cursor.HAND
        translateY:46
        translateX: 9
        layoutInfo: LayoutInfo {
            width: 223
            height: 224
        }
        items: bind for(x in[0..numero_camaras]){
            "Camara {x}: {controller.descCam[x]}";
        }
        onMouseClicked: function (me: MouseEvent) {
            controller.setCam(list.selectedIndex);
            controller.showVistaVisor()
        }
    }
```

Código 15: Listado de Cámaras

Cuando el usuario selecciona una de las cámaras el ListView actualiza el número de cámara a visualizar en el controlador. Recordemos que, según los códigos 7 y 8, esta variable será inmediatamente actualizada en el visor. Por tanto, una vez invocada la función del ***Controller*** *showVistaVisor* el usuario podrá inmediatamente visualizar la cámara seleccionada por pantalla.

4.4.5 VistaMapa - Ver las Cámaras en el Mapa

En el mapa, cada icono de cámara se sitúa en el plano mediante unas coordenadas que han sido previamente descargadas de un archivo de mapa alojado en el servidor y almacenadas localmente. Este archivo nos indica la dirección URL de la imagen que formará el plano sobre el que se situarán las cámaras y el número de cámaras que están presentes en este plano.

JavaFX Script, mediante su sintaxis declarativa que prohíbe la existencia de constructores de clases, nos obliga a crear cada objeto describiéndolo directamente en el código, razón por la que la aplicación soporta un máximo extensible de 15 cámaras, siendo necesaria la declaración en el código de nuevas cámaras en su caso. Por tanto, cada icono de cámara constituye un objeto independiente con unas coordenadas que lo sitúan en el mapa. Según el código 16, estas coordenadas se guardan en una secuencia que mantiene la posición de cada una de ellas mediante un enlazado *binding*. La cantidad de cámaras disponibles se gestiona mediante el atributo de visibilidad que también será gestionado por una secuencia. Al pulsar sobre un icono de cámara, esta se comportará de forma similar a un ítem de la *ListView* del código 15: indicando al ***Controller*** la nueva cámara y llamando a la directa monitorización de la cámara.

```
var Cam0:ImageView = ImageView {
    cursor: Cursor.HAND
    x: bind xCam[0]
    y: bind yCam[0]
    image: Image {url: icoCam}
    onMouseClicked: function(e: MouseEvent):Void{
        controller.setCam(0);
        controller.showVistaVisor();
    }
    visible:bind visibleCam[0];
}
```

Código 16: Icono de cámara en el mapa

Todos estos objetos de imagen se desplazan y escalan en conjunto con el plano, ya que se encuentran contenidos en un nodo *Container:* un contenedor que mantiene la posición relativa de los objetos permitiendo una transformación y desplazamiento de la totalidad de ellos como conjunto respecto al resto de nodos del escenario gráfico. El código 17 muestra parcialmente la implementación de este contenedor:

```
var group : Container= Container {
    scaleX : bind escala_fondo
    scaleY : bind escala_fondo
    translateX:bind xFondo + 9;
    translateY:bind yFondo + 46;
     …
     content: [
         fondo,Cam0,Cam1,Cam2,Cam3,Cam4,Cam5,Cam6,
         Cam7,Cam8,Cam9,Cam10,Cam11,Cam12,Cam13,Cam14
     ]
 }
```

Código 17: Contenedor del mapa

Mientras que los atributos *scaleX* y *scaleY* del *Container* permiten el escalado del mapa simulando un zoom, los atributos *translateX* y *translateY* gestionan su desplazamiento. Para un correcto funcionamiento de estas funcionalidades ha sido necesario establecer una serie de valores máximos y mínimos de escalado y desplazamiento. El código 18 muestra cómo se obtienen estos valores máximos y mínimos en función del resto de variables de las que estos valores dependen.

```
var xFondoMax = bind fondo.image.width - 224 +xFondoMin;
var yFondoMax = bind fondo.image.height - 224+yFondoMin;
var xFondoMin:Number = bind fondo.image.width *(1+escala_fondo)/2
    - fondo.image.width;
```

```
var yFondoMin:Number = bind fondo.image.height *(1+escala_fondo)/2
    - fondo.image.height;
var xFondo:Number = 0;
var yFondo:Number = 0;
var escala_fondo:Number = 1.0;
```

Código 18: Variables para gestión de escalado y desplazamiento del mapa

Cada una de estas variables es imprescindible para el correcto funcionamiento del sistema de escalado y desplazamiento en su totalidad, para una mayor explicación sobre el funcionamiento de estas varaibles:

- **xFondo**: valor que tomará el fondo para su coordenada X.
- **yFondo**: valor que tomará el fondo para su coordenada Y.
- **escala_fondo**: valor entre 0 y 1 que dará escalado al *Container*.
- **xFondoMin**: valor mínimo que puede tomar la variable *xFondo* para mantenerse en su posición correspondiente en el escenario gráfico. Dependerá del tamaño horizontal de la imagen del mapa y de la escala actual que se esté tomando.
- **yFondoMin**: valor mínimo que puede tomar la variable *yFondo* para mantenerse en su posición correspondiente en el escenario gráfico. Dependerá del tamaño vertical de la imagen del mapa y de la escala actual que se esté tomando.
- **xFondoMax**: valor máximo que puede tomar la variable *xFondo* para mantenerse en su posición correspondiente en el escenario gráfico. Dependerá del tamaño horizontal de la imagen del mapa y del valor mínimo *xFondoMin*.
- **yFondoMax**: valor máximo que puede tomar la variable *yFondo* para mantenerse en su posición correspondiente en el escenario gráfico. Dependerá del tamaño vertical de la imagen del mapa y del valor mínimo y*FondoMin*.

El escalado de la imagen se producirá con la pulsación de los botones *zoomIn* o *zoomOut*. Estos botones modificarán el valor de la variable *escala_fondo* de modo que siempre se mantenga entre 1.0 y el valor mínimo permitido para que la imagen se mantenga en su posición correspondiente en la escena. El código 19 muestra la implementación de la herramienta de *zoomOut*.

```
var zoomOut:ImageView = ImageView {
    cursor: Cursor.HAND
    x: 40
    y: 272
    image: Image {
        url: "{__DIR__}images/MapZoomOut.png"
    }
    onMouseClicked: function(e: MouseEvent):Void{
        escala_fondo= escala_fondo - 0.05;
        if (escala_fondo < 224 / imagen_fondo.height) {
            escala_fondo = 224 / imagen_fondo.height;
        }
        if (escala_fondo < 224 / imagen_fondo.width) {
            escala_fondo = 224 / imagen_fondo.height;
```

```
            }
            if (xFondo>xFondoMin){xFondo=xFondoMin;}
            if (yFondo>yFondoMin){yFondo=yFondoMin;}
            if (xFondo<-xFondoMax){xFondo=-xFondoMax}
            if (yFondo<-yFondoMax){yFondo=-yFondoMax}
        }
    }
```

Código 19: Zoom Out

El desplazamiento del mapa se conseguirá mediante dos métodos:

- Unas flechas de dirección, que modifican los parámetros en base a la pulsación de la flecha correspondiente a la dirección del desplazamiento deseado.
- Un mecanismo para pulsar y arrastrar la imagen, que también provocará el desplazamiento consecuente de la imagen gracias al evento de *dragging* que registra JavaFX.

El código 20 describe la implementación de una de las flechas de desplazamiento:

```
    var flechaDerecha:ImageView = ImageView {
        cursor: Cursor.HAND
        x: 205
        y: 274
        image: Image {
            url: "{__DIR__}images/arrow.png"
        }
        onMouseClicked: function(e: MouseEvent):Void{
            xFondo = xFondo-50;
            if (xFondo<-xFondoMax){
                xFondo=-xFondoMax;
            }
        }
    }
```

Código 20: Flecha de Desplazamiento

Un poco más compleja resulta la implementación de la funcionalidad de desplazamiento en base al arrastrado. Esta herramienta se consigue mediante las funciones descritas en el código 21 y que pertenecen al *Container* del código 17.

```
    onMousePressed: function(me: MouseEvent) {
        startX = xFondo;
        startY = yFondo;
    }
    onMouseDragged: function(me: MouseEvent) {
        if (not flechaAbajo.hover
        and not flechaArriba.hover
        and not flechaIzquierda.hover
        and not flechaDerecha.hover
        and not barra.pressed
        and not imagenPestañas.hover){
            if( xFondo<=xFondoMin
            and xFondo>=-xFondoMax
            and yFondo<=yFondoMin
            and yFondo>=-yFondoMax){
                xFondo = startX + me.dragX;
                yFondo = startY + me.dragY;
```

```
                if (xFondo>xFondoMin){xFondo=xFondoMin;}
                if (yFondo>yFondoMin){yFondo=yFondoMin;}
                if (xFondo<-xFondoMax){xFondo=-xFondoMax}
                if (yFondo<-yFondoMax){yFondo=-yFondoMax}
            }
        }
    }
```

Código 21: Desplazamiento basado en *dragging*

Al clicar con el ratón o pulsar la pantalla táctil sobre el *Container*, éste registra la posición de esa pulsación en las variables *startX* y *startY* para después utilizarlas en la modificación de la posición que se realizará en base al desplazamiento producido por el puntero. En ningún caso se producirá desplazamiento si el puntero se sitúa sobre otros objetos del escenario gráfico.

4.4.6 VistaVisor - Monitorizar la Cámara

Sin duda el visor representa una funcionalidad esencial en la aplicación, ninguna otra herramienta tendría sentido si no fuese posible una correcta monitorización de las cámaras de vigilancia. Dada su extrema importancia se hace necesaria una completa explicación de la implementación que ha sido llevada a cabo para lograr una solución satisfactoria.

Las variables "primitivas" utilizadas en esta clase se muestran en el código 22 y son:

- **ipserv:** mantendrá actualizada la IP del servidor gracias a la función *bound* disponible en el ***Controller*** (código 7).
- **cam_actual:** mantendrá actualizada la cámara que debe mostrarse según le indiquen la ***VistaMapa*** (código 15) o ***VistaListado*** (código 16) a través del ***Controller***.
- **mapa_actual:** mantendrá actualizado el mapa del que debe obtener la imagen de la cámara correspondiente según le indique el ***Controller***.
- **cámara_detectada**: una variable para previsión de errores indica que la cámara a monitorizar esté disponible.
- **no_camara:** la dirección URL de la imagen a mostrar si no se está visualizando ninguna.
- **texto_camara:** un texto descriptivo sobre la cámara que se está visualizando.
- **url_imagen:** la dirección URL de la imagen que se mostrará en el escenario gráfico, mientras que en un principio mostrará la imagen de cámara no disponible, posteriormente tomará el valor adecuado para la visualización de la cámara.

```
    var ipserv = bind controller.getIp();
    var cam_actual: Integer= bind controller.getCam();
    var mapa_actual: Integer= bind controller.getMapa();
    var camara_detectada: Boolean = false;
    var no_camara = "{__DIR__}images/nodisponible2.jpg";
    var texto_camara: String;
    var url_imagen = no_camara;
```

Código 22: Variables en *VistaVisor*

El objeto imagen que permite la monitorización se muestra en el código 23:

```
var imagen: Image = Image{
    url: url_imagen;
}
var cam: ImageView = ImageView {
    cache:false
    translateY:72;
    translateX:9;
    fitWidth:223;
    preserveRatio:true;
    image: imagen
}
```

Código 23: Imagen para visualización

La sensación de video *streaming* se produce por la continua recarga de esta imagen que siempre se actualizará desde una misma dirección URL. Para conseguir esta animación de duración indefinida en la imagen, se hace uso del *Timeline* mostrado en el código 24. Según el escenario de ejecución de la aplicación (código 1), el *Timeline* llama a la función *renovarUrl* cada 25 centésimas de segundo en escritorio o navegador ó 75 centésimas de segundo en un dispositivo móvil, de forma que este refresco de la imagen se adapte a las capacidades reales del dispositivo.

```
var timeline: Timeline = Timeline{
    repeatCount: Timeline.INDEFINITE
    keyFrames:[
        KeyFrame{
            time : if (controller.mobile == true){0.75s}
                   else{0.25s}
            action : renovarUrl
        }
    ]
}
```

Código 24: *Timeline* en *VistaVisor*

Cada vez que la función renovarUrl del código 25 sea llamada por el Timeline, ésta creará un nuevo objeto de imagen que sustituirá a la anterior (código 23) y será mostrada en el visor. La URL de esta imagen necesita información sobre la dirección IP del servidor, el mapa que se está manejando y la cámara que se desea visualizar. Esta información la obtiene de las variables correspondientes del código 22. Si no ha habido un error en la carga de la imagen se tomará la cámara como detectada, si se comprueba que si que ha habido un error y no se había detectado una cámara se llamará al control de errores (ver capítulo 1.3.8 VistaError para más información).

```
function renovarUrl() : Void{
    var imagen2: Image = Image{
        url: "http://{ipserv}/mapa{mapa_actual}/
            camara{cam_actual}/cam.jpg";
    }
    cam.image=imagen2;
    if(imagen2.error==false){camara_detectada=true}
    if(imagen2.error==true and camara_detectada==false){
        cerrarVisor();
        controller.showErrorScreen("la camara {cam_actual}:
            {controller.getDesc(cam_actual)} \n
            no se encuentra disponible");
```

```
            url_imagen = no_camara;
            controller.setCam(-1);
        }
    }
```

Código 25: Función *renovarUrl*

Todo este proceso de recarga de imágenes se inicia con la llamada de una de las vistas al controlador para que muestre la vista de monitorización, el ***Controller*** gestionará esta petición como muestra el código 26:

```
public function showVistaVisor() {
        if (vistaVisor == null) {
            vistaVisor = VistaVisor {
                controller: this
            }
        }
        vistaVisor.iniciarVisor();
        contents = vistaVisor;
        vistaActual=vistaVisor;
    }
```

Código 26: Función *showVistaVisor* en el *Controller*

El controlador, antes de mostrar por la pantalla la ***VistaVisor***, llama a la función *iniciarVisor*, esta función (descrita en el código 27) invoca al *Timeline* para que comience a ejecutarse e indica a la función *setTextoCamara* que, según el código 28, modifique el texto que debe ser mostrado para describir la cámara.

```
    protected function iniciarVisor(): Void {
        if(cam_actual != -1){
            camara_detectada=false;
            timeline.playFromStart();
            setTextoCamara()
        }
        else{cam.image=imagen;}
    }
```

Código 27: Función *iniciarVisor*

```
    protected function setTextoCamara(): Void {
    texto_camara="camara{cam_actual}:{controller.getDesc(cam_actual)}"
    }
```

Código 28: Función setTextoCamara

En cualquier momento el usuario puede decidir finalizar la monitorización; si esto ocurre, el *Timeline* será llamado a pararse para un rendimiento óptimo de la aplicación.

```
    protected function cerrarVisor(): Void {
        timeline.stop();
        cam.image=imagen;
    }
```

Código 29: Función *cerrarVisor*

4.4.7 VistaError - Manejo de errores

La propia naturaleza de la aplicación fuerza a la utilización de recursos externos, recursos cuya disponibilidad no tenemos la capacidad de controlar, es por esto se hace necesario un control sobre los errores que puedan producirse en la llamada a estos recursos remotos. En el código 25 hemos observado como al obtener un error en la visualización de la cámara se ejecutaba la siguiente línea:

```
controller.showErrorScreen("la camara {cam_actual}:
   {controller.getDesc(cam_actual)} \n
    no se encuentra disponible");
```

La función del ***Controller*** *showErrorScreen*, según el código 30, recibe un texto informativo sobre el tipo de error que se ha producido. Una vez guarde este texto en la variable indicada para ello, llamará a la VistaError para que informe definitivamente al usuario sobre el error (figura 4.5) y la aplicación pueda continuar con su funcionamiento normal.

```
public function showErrorScreen(str: String) {
    textoError=str;
    showVistaError();
}
```

Código 30: Función *showErrorScreen*

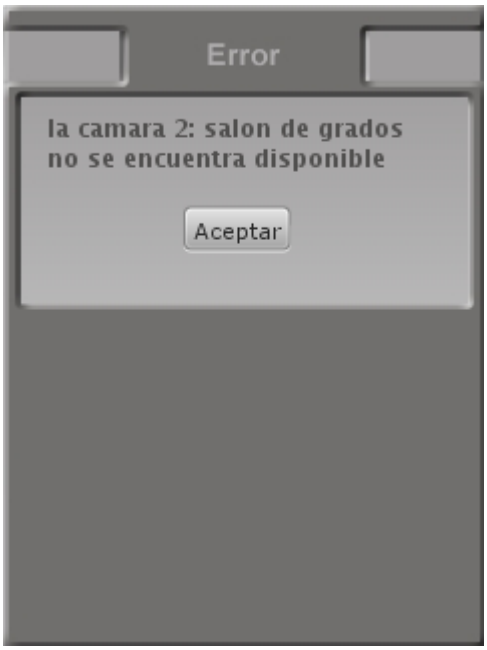


Figura 4.5: Error

Cuando el usuario haya leído el mensaje, pulsará el botón de aceptación del error, este botón invoca a la función *volverVistaAnterior* del Controller. Como muestra el código 31, esta función devolverá a la aplicación a su estado anterior al error.

```
    public function volverVistaAnterior() {
        if(vistaActual!=null){
            contents=vistaActual;
        }
        else{
            contents = vistaPrincipal;
        }
    }
```

Código 31: Función *volverVistaAnterior*

4.4.8 Creación de la GUI

La interfaz gráfica de usuario se basa en los objetos gráficos aportados por la API JavaFX 1.2. La debida colocación de estos objetos en el escenario gráfico permite que sean tapados por otra capa con la imagen que deseemos mostrar en la interfaz pero que a la vez sean los que controlen las interacciones del usuario gracias a su capacidad para el manejo de eventos. Una solución tan simple como efectiva, pues permite modificar completamente el aspecto de la aplicación con muy poco esfuerzo y sin modificar el código: basta con crear nuevas imágenes sustitutivas de las anteriores.

En la figura 4.6 apreciamos los objetos básicos que manejaran los eventos en todas las vistas para el cambio de una a otra o para el desplazamiento de toda la aplicación por la pantalla del escritorio, estos objetos están cubiertos por una imagen que le da el aspecto final.

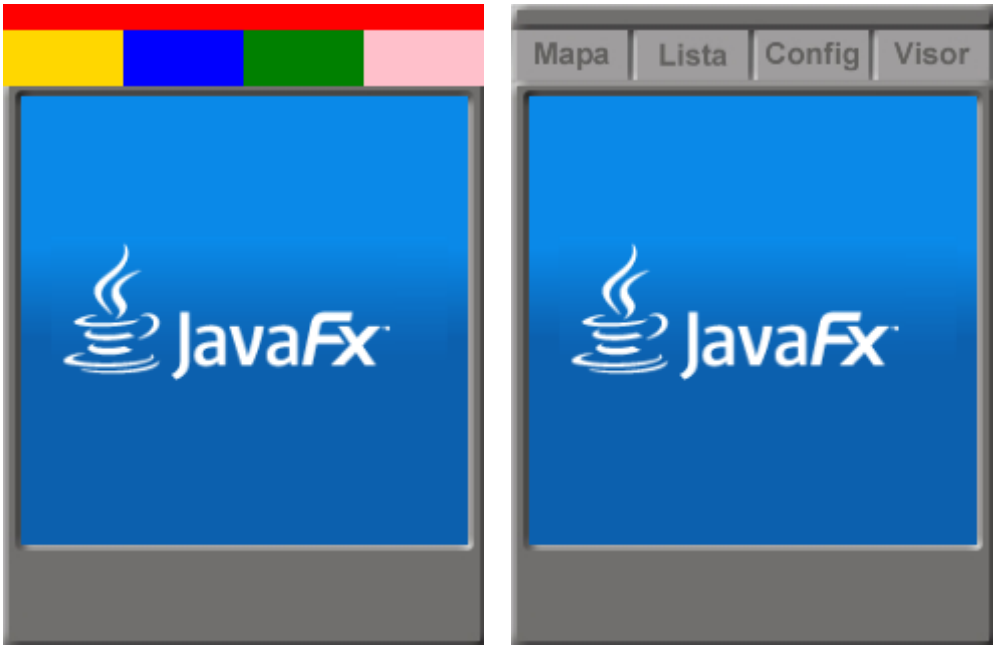


Figura 4.6: Pestañas de Interfaz Gráfica

En la imagen izquierda de la figura 4.6 se ha eliminado el código que mostraba la imagen que tapa a las pestañas y la barra, además se ha añadido color a cada pestaña como se observa en los códigos 32 y 33.

Los objetos que manejarán los eventos provocados por el usuario son:

- **Barra de desplazamiento:** en rojo, permitirá mover la aplicación sólo en entorno de escritorio (código 32).
- **Pestaña de Mapa:** en amarillo, al pulsar esta pestaña la aplicación pasará a mostrar la *VistaMapa* (código 33).
- **Pestaña de Listado:** en azul, al pulsar esta pestaña la aplicación pasará a mostrar la *VistaListado*.
- **Pestaña de Configuración:** en verde, al pulsar esta pestaña la aplicación pasará a mostrar la *VistaConfig*.
- **Pestaña de Visor:** en rosa, al pulsar esta pestaña la aplicación pasará a mostrar la *VistaVisor*.

```
var barra:Rectangle = Rectangle {
    cursor: {if(not controller.browser and not controller.mobile)
                {Cursor.MOVE}
             else{Cursor.DEFAULT}}
    width: 240
    height: 13
    fill: Color.RED
    onMouseDragged: function(me: MouseEvent) {
        if (not controller.browser and not controller.mobile) {
            controller.X = me.screenX - me.dragAnchorX;
            controller.Y = me.screenY - me.dragAnchorY;
        }
    }
}
```

Código 32: Barra de desplazamiento

```
var mapa:Rectangle = Rectangle {
    width: 240/4
    height: 28
    fill: Color.YELLOW
    onMouseClicked: function(me: MouseEvent) {
        controller.showVistaMapa();
    }
}
```

Código 33: Pestaña de Mapa

Para manejar de forma adecuada la disposición en pantalla de estas pestañas se ha recurrido a un contenedor horizontal como muestra el código 34:

```
var pestañas: HBox = HBox {
    cursor: Cursor.HAND
    content: [mapa, listado, configuracion, visor]
}
```

Código 34: *Hbox* contenedor de las pestañas

También se han utilizado otros recursos como etiquetas de texto, cuadros *TextBox*, listas *ListView* o botones. Como se aprecia en la figura 4.7, la clase ***VistaConfig*** aporta un ejemplo de la utilización de estos recursos. En la imagen izquierda aún no se han añadido los elementos de interfaz y se muestran sólo las pestañas y la imagen de fondo. En la derecha ya se han colocado estos elementos.

Figura 4.7: Uso de *Text*, *TextBox*, *Button* y *ListView* en interfaz gráfica

4.5 Software complementario

Para el completo funcionamiento de la aplicación, es necesario utilizar herramientas de software que nos aporten los recursos externos necesarios de los que la aplicación hace uso. Un servidor Apache y el programa Mplayer nos proveen de estos recursos que serán necesarios para la configuración de la aplicación y la monitorización de las cámaras.

4.5.1 Servidor Apache

El servidor Apache nos aporta la conectividad web necesaria para la descarga de los recursos externos. Un archivo de configuración como el del código 35 indicará a la aplicación la cantidad de mapas que se disponen en el servidor y dará una pequeña descripción de cada uno de ellos para que el usuario pueda seleccionar el adecuado.

```
numero_mapas = 3
mapa0 = upct
mapa1 = casa
mapa2 = empresa
```

Código 35: Archivo de configuración *config.txt*

Una vez se haya seleccionado el mapa, la aplicación podrá descargarse el plano correspondiente junto con las coordenadas de las cámaras gracias a un archivo de mapa (uno por cada mapa) que también reside en el servidor y que deberá tener una configuración similar a la mostrada por el código 36.

```
    mapa = http://www.tel.uva.es/images/edificio_baja.jpg
    numero_camaras = 15

    camara0desc = secretaria
    camara0X = 243
    camara0Y =295

    camara1desc = reprografia
    camara1X = 251
    camara1Y = 236

    camara2desc = salon de grados
    camara2X = 272
    camara2Y = 180
    …

    camara13desc = escaleras centrales
    camara13X = 278
    camara13Y = 7

    camara14desc = escaleras oeste
    camara14X = 40
    camara14Y = 10
```

Código 36: Archivo de mapa *mapa0.txt*

Por otra parte, el servidor Apache también se puede utilizar para poner la aplicación a disposición de usuario en su modo de ejecución *applet*, para ello, un archivo *html* (similar al del código 37) invocará la ejecución del *applet* mediante un archivo *jnlp* en la misma ruta en el servidor. Este *jnlp* servirá de link para la aplicación y deberá llevar en su campo *codebase* la ruta relativa o absoluta del *jar* al que hace referencia.

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title>video-vigilancia JavaFX</title>
</head>
<body BACKGROUND="background.png" >
<div align="center">
<h1>video-vigilancia JavaFX</h1>
<script src="http://dl.javafx.com/1.2/dtfx.js"></script>
<script>
    javafx(
        {
              archive: "Videovigilancia_UPCT.jar",
              draggable: true,
              width: 240,
              height: 320,
              code: "videovigilancia.Main",
              name: "Videovigilancia_UPCT"
        }
    );
</script>
</div>
</body>
</html>
```

Código 37: Archivo html *index.html*

4.5.2 Mplayer

Mientras que cualquier cámara, ya sea IP o web, exporta un flujo de video para su visualización, la aplicación necesita imágenes para efectuar la monitorización. Mplayer es el programa encargado de realizar esta conversión de formatos gracias a la siguiente línea a introducir en el terminal Linux del servidor:

```
mplayer tv:// -tv driver=v4l2:device=/dev/video0 -fps 4
              -nosound -vo jpeg
```

Para una mayor comprensión del comando podemos analizar cada uno de sus componentes por separado:

- **mplayer:** invoca al programa Mplayer previamente instalado en el terminal Linux.
- **tv://:** indica al programa que debe ejecutarse en modo de monitorización tv.
- **-tv driver=v4l2:device=/dev/video0:** el recurso de vídeo al que el Mplayer debe acceder para la monitorización. Como vemos, el dispositivo del que se hace uso se encuentra en */dev/videoX*. El kernel Linux detecta las cámaras conectadas y les asigna en la ruta */dev* un nombre *videoX* en que la *X* se incrementará secuencialmente tras la conexión, de forma que es necesario un control sobre cada cámara, su conexión y su distribución física real.
- **-fps 4:** La velocidad en frames por segundo a la que se muestreará la imagen. Para una correcta visualización en el cliente, esta velocidad debe coincidir con la inversa del tiempo que pasa entre cada keyFrame del Timeline de la aplicación. Como puede comprobarse en el código 24, en este caso ¼ = 0,25 segundos.
- **-nosound:** no recogerá el sonido del micrófono de la cámara.
- **-vo jpeg:** exportará imágenes en formato *jpeg* desde el dispositivo indicado por *-tv* y a la velocidad *–fps*.

Para el soporte de múltiples cámaras, será necesario lanzar un proceso Mplayer por cámara, de forma que la ruta de lanzamiento de cada proceso se sitúe en la carpeta adecuada dentro del directorio de archivos del servidor Apache. Por ejemplo, para la cámara secretaría (cámara 0) del mapa upct (mapa 0) la ruta adecuada será (para el servidor Apache Lampp):

```
/opt/lampp/htdocs/mapa0/camara0
```

4.6 Funcionamiento de la aplicación

En esta sección se explica el funcionamiento general de la aplicación, esto es, la forma en que un usuario interactuaría con el programa para obtener los resultados deseados. De forma general un usuario podría seguir los siguientes pasos para llegar a monitorizar una cámara:

1. Configurar la aplicación para conectarse con un servidor, indicando una dirección IP.
2. Obtener del servidor la lista de mapas que se encuentren disponibles en el mismo.
3. Seleccionar uno de los mapas y descargar toda la información sobre el mapa: plano, número de cámaras disponibles, una descripción de cada cámara y las coordenadas de la cámara sobre el plano.
4. Seleccionar una cámara para su monitorización, ya sea desde el listado de cámaras o desde el mapa haciendo uso de las herramientas de desplazamiento o zoom para su selección.
5. Visualizar la cámara escogida.
6. Volver a iniciar el proceso volviendo a cualquiera de los puntos anteriores según sus necesidades.

Todos los datos de configuración como la selección del mapa o la IP del servidor quedarían almacenados de forma local y no necesitarían volver a ser insertados a pesar del cierre de la aplicación

Al arrancar la aplicación el usuario se encontraría con la pantalla de bienvenida de la figura 4.8:

Figura 4.8: Pantalla de bienvenida

Desde ahí, si nunca ha configurado la aplicación, deberá hacerlo acudiendo a la pestaña de configuración. Donde insertará una dirección IP como muestra la imagen izquierda de la figura 4.9.

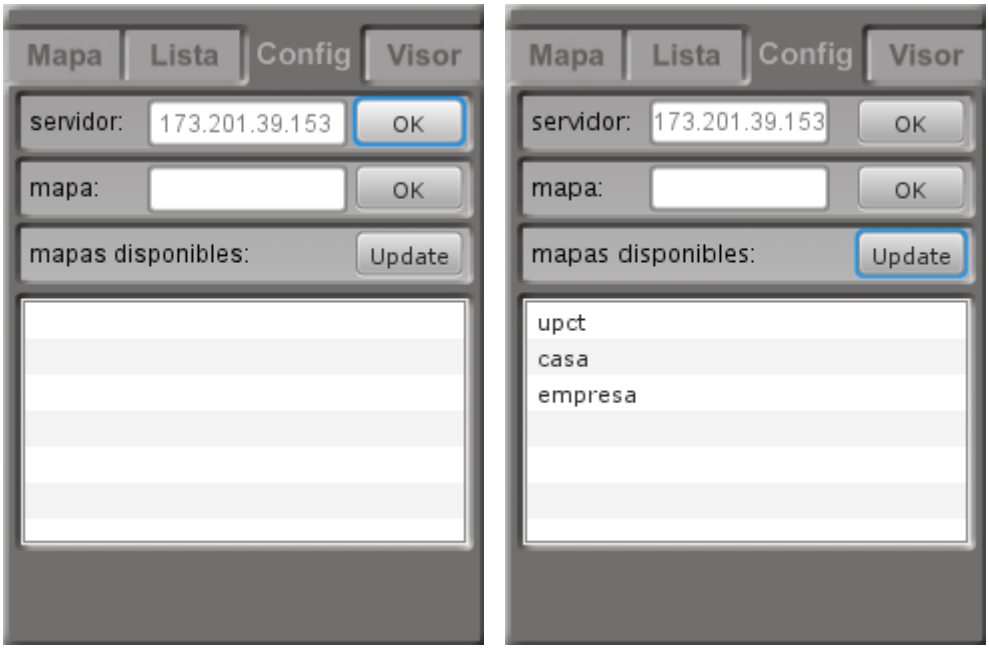


Figura 4.9: Configurar IP y mapa

Tras pulsar el botón de actualización se mostrará una lista de mapas disponibles en el servidor tal y como muestra la imagen derecha de la figura 4.9. Una vez el usuario seleccione el mapa adecuado, podrá pasar a la selección de la cámara, por ejemplo, a través del mapa. La *VistaMapa* mostrará una imagen de carga mientras se descarga la imagen del plano y gestiona la posición de las coordenadas de las cámaras (figura 4.10).

Figura 4.10: Selección y carga del mapa

Cuando el mapa se encuentre cargado, el usuario podrá desplazarse por él mediante el desplazamiento directo o la utilización de las flechas auxiliares hasta encontrar la cámara deseada en el mapa. También puede hacer uso de las herramientas de zoom para la búsqueda de la cámara adecuada. En la figura 4.11 se muestra como el mapa ha sido desplazado y posteriormente reducido en escala.

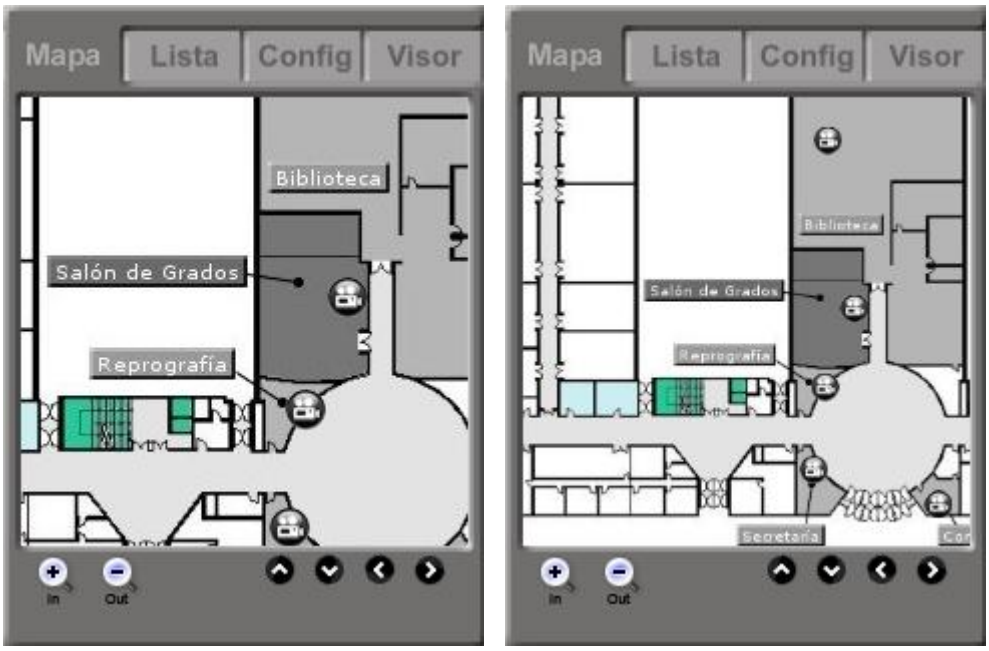


Figura 4.11: Desplazamiento y Zoom

Al pulsar uno de los iconos de cámara automáticamente se mostrará el visualizador monitorizando la cámara seleccionada- También es posible hacer esta selección a través del listado descriptivo de cámaras, tal y como muestra la figura 4.12.

Figura 4.12: Listado y Visor

5 Conclusiones

La familia tecnológica JavaFX promete un futuro en el que los más diversos escenarios y dispositivos se encuentren unidos por una plataforma de desarrollo común. La actual versión JavaFX 1.3 ya da soporte para la creación sobre dispositivos Windows Mobile y sobre las principales plataformas de escritorio y navegadores web. La creación de escenarios gráficos animados resulta sencilla gracias al lenguaje declarativo y a las múltiples herramientas que nos brinda la API. La gran conectividad web y multimedia permite conexiones asíncronas con el servidor donde la carga de todo tipo de medios resulta casi trivial. Los entornos de desarrollo integrados con soporte JavaFX ya aportan múltiples herramientas que facilitan el trabajo sobre esta plataforma.

Con todo esto, y después del completo estudio que se ha realizado y presentado, la conclusión general sobre la tecnología es clara: JavaFX debe mejorar. Por muchos aspectos el presente de JavaFX no invita a subirse a su carro de desarrollo. Ya desde un principio, Sun llega tarde a la lucha en el mundo RIA, campo sobradamente dominado por Adobe Flash, Microsoft Silverlight y Google GWT, y lo hace además con un producto en dudoso estado de madurez, con mucho por trabajar. La sensación general que JavaFX nos deja tras el uso de una aplicación es mediocre. Los tiempos de espera son altos para ciertas operaciones, el rendimiento general es bajo comparado con aplicaciones similares Flash o Silverlight, aparecen síntomas que auguran cierta inestabilidad y falta de coherencia en las aplicaciones y el escenario gráfico no resulta tan espectacular como con otros frameworks. Estos aspectos lastran enormemente las posibilidades de JavaFX como framework para el desarrollo de proyectos de software y RIAs.

Una gran ventaja de la tecnología sobre sus competidoras es su capacidad multi-dispositivo. Es cierto que existe soporte para JavaFX en dispositivos móviles Windows Mobile, pero, en el estado actual de la técnica, la falta de soporte para pilares básicos y con gran futuro como Symbian, Android o Iphone ponen en entredicho y lastran la credibilidad sobre la capacidad móvil real de la tecnología. Es más, el rendimiento de JavaFX sobre el dispositivo Windows Mobile utilizado para el estudio de este proyecto ha sido verdaderamente decepcionante, rindiendo a un nivel muy por debajo de lo que puede considerarse aceptable para una aplicación comercial, presentando además diversas incompatibilidades e incoherencias en el funcionamiento real de la misma.

Muchas mejoras están aún por llegar: soporte para ejecución en TV y los diferentes SO operativos móviles, además del propio SO JavaFX Mobile, incluyendo la optimización de las funcionalidades JavaFX ya implementadas. Estas mejoras esperanzan el proyecto JavaFX, sin embargo, de cara a los desarrolladores, las distintas actualizaciones JavaFX han traído numerosos cambios en la forma de trabajar con la tecnología y las incompatibilidades entre las versiones siembran dudas a la hora de optar por desarrollar con una tecnología que todavía debe cambiar y mejorarse.

Por otra parte, JavaFX ha sido capaz de reinventarse a sí mismo en poco tiempo dando muestras de sus previsibles futuras capacidades, ha mejorado aspectos fundamentales y cada vez incorpora un mayor soporte para el desarrollo. Tiene grandes y exclusivas funcionalidades como el multi-dispositivo, la unión desarrollador-diseñador y la capacidad de enlace y presenta una envidiable base de dispositivos Java que soportarán la tecnología. Todo con una sintaxis declarativa en el lenguaje que facilita la labor de un desarrollador que no esté acostumbrado a trabajar con información gráfica.

Sin duda, JavaFX parte de un estado complejo pero prometedor, en el que una apuesta por la tecnología puede ser arriesgada, pero también incalculablemente ganadora. En nuestro objetivo de video-vigilancia, por ejemplo, JavaFX resulta muy adecuado desde la perspectiva de que nos ha permitido crear una aplicación de monitorización online que además estará disponible para dispositivos móviles, creando numerosas posibilidades de utilización. Gracias a JavaFX, la aplicación podrá reunir lo mejor de cada escenario: se creará un punto de vigilancia móvil en el que estar conectado no supondrá encontrarse atado a un emplazamiento y cualquier dispositivo conectado a internet se convertirá en un punto de acceso a la monitorización gracias a las capacidades online que permiten la no instalación de software. Las funcionalidades a implementar en la aplicación son ilimitadas, pues la conectividad web nos da acceso grandes posibilidades en este campo.

La aplicación desarrollada deja muchas puertas abiertas a futuros trabajos de desarrollo que partan de su base. Una futura ampliación del sistema de video-vigilancia podría pasar por una mejora del lado servidor del mismo. Debería existir una mayor comunicación entre cliente y servidor para gestionar conjuntamente otras opciones más avanzadas en el sistema. Resultaría muy interesante contar con un sistema en que cada cliente deba registrarse en el servidor de forma que sea necesaria autenticación para poder acceder a los recursos del sistema. El control de usuarios también podría hacer uso de gestión de privilegios para que solo ciertos usuarios puedan acceder o modificar ciertos recursos. Esta capacidad de modificación permitiría, por ejemplo, a un superusuario acceder al servidor para modificar un mapa si fuese necesario. Un sencillo editor de mapas en la aplicación resultaría óptimo para este objetivo.

Esta mejora del servidor y de la comunicación en todo el sistema podría incluir funcionalidades de monitorización más avanzadas que la simple visualización, llegando a soportar control remoto de las cámaras IP, de manera que el usuario pueda modificar el área de visionado de una cámara rotando su ángulo hasta 360º remotamente. El servidor podría soportar detección de movimiento para una notificación en el cliente de que cierta cámara debería ser observada, pues supone una situación de riesgo. Esta notificación de eventos podría ser parametrizable para variar la sensibilidad de respuesta ante la detección del movimiento, desechando cambios en la imagen que puedan resultar inútiles para el usuario.

No resultaría menos interesante el mantenimiento de un historial de los eventos registrados, con grabación en memoria de porciones de video en los momentos clave de la monitorización. Siempre puede resultar interesante disponer de una copia en disco de aquellos momentos susceptibles de ser revisualizados. Para esto el servidor también podría contar con opciones extra como inserción de fecha en la imagen o pre-identificación del desencadenante de la actividad, siendo capaz de detectar si se trata de una persona o no. Sin embargo, podría no resultar eficiente, por ejemplo, la grabación local de vídeo si la aplicación dispone de pocos recursos de memoria, como es el caso de los dispositivos móviles. JavaFX permite un control total sobre las herramientas que podrían estar disponibles en cada entorno de ejecución, de modo que la aplicación podría ampliarse teniendo en cuenta las posibilidades reales de cada escenario. En entornos de escritorio JavaFX permite el uso de clases Java para su complemento, multiplicando las opciones en base a la disponibilidad de los recursos. Por otra parte, el escenario gráfico debería ser modificado para adaptarse completamente al medio, ya que no existe ninguna necesidad de restringir el tamaño de la aplicación al 320x240 de una pantalla móvil, este tamaño debe justificarse al escenario utilizado, permitiendo, por ejemplo, pantalla completa en modo escritorio o dimensiones variables en modo *applet*.

Un aspecto importante a mejorar dependería de las futuras actualizaciones de JavaFX, y es que un futuro soporte de *live video streaming* permitiría una gran mejora en la monitorización, siempre y cuando esta funcionalidad se incorpore para el perfil común de desarrollo, de forma que esté disponible para dispositivos móviles. En caso contrario, si como ahora, no existiese soporte para *streaming* en directo, esto no tiene por qué suponer un problema, tal y como demuestra este proyecto, siempre es posible una mejora partiendo del sistema de monitorización existente basado en el continuo reenvío de imágenes para crear sensación *streaming*. La aplicación cliente podría aprovechar esta peculiaridad para, por ejemplo, indicarle al servidor la velocidad en fps y el tamaño de imagen que podría soportar, según el usuario estime que la visualización resulta correcta y libre de errores.

En definitiva, JavaFX y el mundo de la video-vigilancia suponen una puerta abierta a la imaginación y al desarrollo. Las posibilidades en seguridad que un sistema de video-vigilancia supone son infinitas y no excesivamente complejas de implementar gracias al framework JavaFX, que, por sus características resulta idóneo para este tipo de sistemas. Recordemos que la nueva tecnología de Sun, aunque aún en estado de desarrollo, aporta numerosas posibilidades gracias a su suporte para todos los escenarios, tanto escritorio como navegador y móvil, a los que pronto se unirá TV. La potente capacidad gráfica que la sintaxis declarativa JavaFX convierte en algo sencillo de implementar puede convertirse en un gran arma para desarrolladores con poca experiencia gráfica, y no menos, en una excelente herramienta para diseñadores que quieran trabajar en conjunto con los desarrolladores para convertir las aplicaciones en toda una explosión de capacidad gráfica.

Todo esto unido a las capacidades del lenguaje JavaFX Script como la dependencia *binding*, la animación gráfica *Timeline* y la gran conectividad web y multimedia hacen de JavaFX una aventura con un futuro tan incierto como prometedor, pero que, sin duda, solamente acaba de comenzar.

Anexo A – Código Completo

Clase Main

```
package videovigilancia;

import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.scene.Scene;
import javafx.stage.Stage;
import javafx.stage.StageStyle;
import videovigilancia.Vista;
import videovigilancia.Controller;

var mobile: Boolean = "{__PROFILE__}" == "mobile";
var browser: Boolean = "{__PROFILE__}" == "browser";

Vista.screenWidth = 240;
Vista.screenHeight = 320;


var fondo:ImageView =  ImageView{
    image: Image{
           url: "{__DIR__}images/visor.jpg";
     }
}

var controller = new Controller;

var stage: Stage = Stage {
    x:bind controller.X;
    y:bind controller.Y;
    title: "Videovigilancia UPCT"
    style: StageStyle.UNDECORATED

    scene: Scene {
        width: Vista.screenWidth
        height: Vista.screenHeight
        content: bind [fondo, controller.contents]
    }

}

controller.showDefaultScreen();
```

Clase Controller

```
package videovigilancia;

import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.scene.Scene;
import javafx.stage.Stage;
import javafx.stage.StageStyle;
import videovigilancia.Vista;
import videovigilancia.Controller;

var mobile: Boolean = "{__PROFILE__}" == "mobile";
var browser: Boolean = "{__PROFILE__}" == "browser";

Vista.screenWidth = 240;
Vista.screenHeight = 320;


var fondo:ImageView =  ImageView{
    image: Image{
           url: "{__DIR__}images/visor.jpg";
     }
}

var controller = new Controller;

var stage: Stage = Stage {
    x:bind controller.X;
    y:bind controller.Y;
    title: "Videovigilancia UPCT"
    style: StageStyle.UNDECORATED

    scene: Scene {
        width: Vista.screenWidth
        height: Vista.screenHeight
        content: bind [fondo, controller.contents]
    }

}

controller.showDefaultScreen();
```

Clase Vista

```
package videovigilancia;

import javafx.scene.*;

public var screenWidth: Number;
public var screenHeight: Number;

public abstract class Vista extends CustomNode {

    protected var controller: Controller;

    protected var view: Node;

    protected abstract function createView(): Void;

    public override function create(): Node {
        createView();
        return view
    }

}
```

Clase VistaConfig

```
package videovigilancia;

import javafx.scene.Cursor;
import javafx.util.Properties;
import javafx.geometry.HPos;
import javafx.geometry.VPos;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.scene.input.MouseEvent;
import javafx.scene.layout.LayoutInfo;
import javafx.scene.layout.VBox;
import javafx.scene.text.Font;
import videovigilancia.Vista;
import javafx.scene.control.ListView;
import java.io.InputStream;
import java.io.OutputStream;
import java.io.IOException;
import javafx.io.Resource;
import javafx.io.Storage;
import javafx.scene.control.TextBox;
import javafx.scene.layout.HBox;
import javafx.scene.paint.Color;
import javafx.scene.shape.Rectangle;
import javafx.scene.text.Text;
import javafx.io.http.HttpRequest;
import javafx.scene.layout.Panel;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import java.lang.Exception;

public class VistaConfig extends Vista{
    var version_config;
    def maximo_mapas = 4;
    var numero_mapas = 4;
    var mapas = ["","","","",""];
    var buttonWidth = (screenWidth - 10) / 4.0;

    var ip_actual;
    var location_lista;
    var storage_ip: Storage;
    var res_ip: Resource;
    var storage_lista: Storage;
    var res_lista: Resource;
    var storage_mapa: Storage;
    var res_mapa: Resource;

    var count_chars: Number;
    var count_lines: Number;

    def max_chars = 320;
    def max_lines = 14;

    var width: Number = 250;
    var height: Number = 320;

    var usrtext: String = " ";
    var stageDragInitialX:Number;
    var stageDragInitialY:Number;
```

```
    public var argumento_ip: TextBox = TextBox {
        translateY:if (controller.mobile == true) {49.5}else{47.5}
        translateX:72
        onMouseClicked: function(e: MouseEvent):Void{
            argumento_ip.text = argumento_ip.promptText;
        }
    }

    public var argumento_mapa: TextBox = TextBox {
        translateY:if (controller.mobile == true) {82.5}else{80.5}
        translateX:72
    }

    public var ip: Text = Text {
        font:Font{name:"Dialog"}
        translateY:62.5
        translateX:13
        content:"servidor:      "
    }

    public var mapa_config: Text = Text {
        translateY:95.5
        translateX:13
        font:Font{name:"Dialog"}
        content:"mapa:          "
    }

    public var lista_mapas: Text = Text {
        translateY:128.5
        translateX:13
        font:Font{name:"Dialog"}
        content:"mapas disponibles: "
    }

    var ok_ip: Button = Button {
        cursor: Cursor.HAND
        text: "OK"
        translateY:if (controller.mobile == true) {49.5}else{47.5}
        translateX:176
        action: function() {
            storeIp(false);
        }
        layoutInfo: LayoutInfo {
            width: buttonWidth - 4
        }
    }

    var ok_mapa: Button = Button {
        cursor: Cursor.HAND
        translateY:if (controller.mobile == true) {82.5}else{80.5}
        translateX:176
        text: "OK"
        action: function() {
            try{

if(Integer.parseInt(argumento_mapa.text)<=numero_mapas){

controller.setMapa(Integer.parseInt(argumento_mapa.text));
                    storeMapa(false);
```

```
                }

else{argumento_mapa.text="";argumento_mapa.promptText="no
disponible";}
            }
            catch(e : Exception){
                argumento_mapa.text="";argumento_mapa.promptText="no
disponible";
            }
        }
        layoutInfo: LayoutInfo {
            width: buttonWidth - 4
        }
    }

    var actualizar: Button = Button {
        cursor: Cursor.HAND
        translateY:if (controller.mobile == true) {115.5} else{113.5}
        translateX:176
        text: "Update"
        action: function() {
            iniciarPeticionLista();
        }
        layoutInfo: LayoutInfo {
            width: buttonWidth - 4
        }
    }

    var imagenPestañas:ImageView =  ImageView{
        image: Image{
            url: "{__DIR__}images/configfondo2.jpg";
        }
    }

    var barra:Rectangle = Rectangle {
        cursor: {if (not controller.browser and not controller.mobile
) {Cursor.MOVE} else{Cursor.DEFAULT}}
        width: 240
        height: 13
        onMouseDragged: function(me: MouseEvent) {
            if (not controller.browser and not controller.mobile ) {
                controller.X = me.screenX - me.dragAnchorX;
                controller.Y = me.screenY - me.dragAnchorY;
            }
        }
    }

    var mapa:Rectangle = Rectangle {
        width: 240/4
        height: 28
        onMouseClicked: function(me: MouseEvent) {
        controller.showVistaMapa();
        }
    }

    var listado:Rectangle = Rectangle {
        width: 240/4
        height: 28
        onMouseClicked: function(me: MouseEvent) {
            controller.showVistaListado();
```

```
        }
    }

    var configuracion:Rectangle = Rectangle {
        width: 240/4
        height: 28
        onMouseClicked: function(me: MouseEvent) {
            controller.showVistaConfig();
        }
    }

    var visor:Rectangle = Rectangle {
        width: 240/4
        height: 28
        onMouseClicked: function(me: MouseEvent) {
            controller.showVistaVisor();
        }
    }

    var pestañas: HBox = HBox {
        cursor: Cursor.HAND
        content: [mapa, listado, configuracion, visor]
    }

    var headingLabel: Label = Label {
        text: "Menu Principal"
        textFill: Color.WHITE
        layoutInfo: LayoutInfo {
            hpos: HPos.CENTER,
            vpos: VPos.CENTER
            height: 30
        }
        font: Font {
            name: "Bitstream Vera Sans Bold"
            size: 14
        }
    }

    var list: ListView = ListView {
        cursor: Cursor.HAND
        translateX: 9
        translateY: 147
        layoutInfo: LayoutInfo {
            width: 223
            height: 123
        }
        items: bind for(x in[0..numero_mapas]){mapas[x];}
        onMouseClicked: function (me: MouseEvent) {
            controller.setMapa(list.selectedIndex);
            storeMapa(false);
        }
    }

    var vBox:VBox = VBox {
        nodeHPos:HPos.LEFT;
        content: [barra,pestañas]
    }

    public function storeIp(clearold : Boolean): Void {
        if (checkStoreIp() == true){loadIp();}
```

```
        var outp: OutputStream = res_ip.openOutputStream(clearold);
        argumento_ip.commit();
        var str: String = argumento_ip.text;
        var properties = Properties{};
        if(str.trim().length() == 0 ) {
            outp.close();
            return;
        }
        properties.put("ip", str);
        properties.store(outp);
        outp.close();
        argumento_ip.text = "";
        loadIp();
    }

    public function checkStoreIp() : Boolean {
        var ret: Boolean = false;
        try {
            storage_ip = Storage {
                source: "Ip"
            }

            res_ip = storage_ip.resource;
            ret = false;

        }
        catch (e : IOException ) {
            res_ip = storage_ip.resource;
            ret = true;
        }
        ret = true;
    }

    public function checkStoreLista() : Boolean {
        var ret: Boolean = false;
        try {
            storage_lista = Storage {
                source: "Lista"
            }
            res_lista = storage_lista.resource;
            ret = false;
        }
        catch (e : IOException ) {
            res_lista = storage_lista.resource;
            ret = true;
        }
        ret = true;
    }

    public function loadIp() : Void {
        var inp: InputStream = res_ip.openInputStream();
        var properties = Properties{};
        var ip;
        properties.load(inp);
        if(properties.get("ip") != null){
            ip = properties.get("ip");
        }
        inp.close();
        controller.setIp(ip);
        ip_actual = ip;
```

```
        location_lista="http://{ip}/config/config.txt";
        argumento_ip.promptText = controller.getIp();
    }

    public function loadLista() : Void {
        var inp: InputStream = res_lista.openInputStream();
        var properties = Properties{};
        properties.load(inp);
        if(properties.get("numero_mapas") != null){
            numero_mapas                                                    =
Integer.parseInt(properties.get("numero_mapas"));
            numero_mapas--;
        }
        if(numero_mapas < maximo_mapas){
            for(x in [0..maximo_mapas]){
                mapas[x]="";
            }

            for(x in [0..numero_mapas]){
                mapas[x]=properties.get("mapa{x}");
            }
        }
        inp.close();
    }

    public function storeMapa(clearold : Boolean): Void {
        if (checkStoreMapa() == true){loadIp();}
        var outp: OutputStream = res_mapa.openOutputStream(clearold);
        var str: String = "{controller.getMapa()}";
        var properties = Properties{};
        if(str.trim().length() == 0 ) {
            outp.close();
            return;
        }
        properties.put("mapa", str);
        properties.store(outp);
        outp.close();
        loadMapa();
    }

    public function checkStoreMapa() : Boolean {
        var ret: Boolean = false;
        try {
            storage_mapa = Storage {
                source: "Mapa_Actual"
            }
            res_mapa = storage_mapa.resource;
            ret = false;
        }
        catch (e : IOException ) {
            res_mapa = storage_mapa.resource;
            ret = true;
        }
        ret = true;
    }

    public function loadMapa() : Void {
        var inp: InputStream = res_mapa.openInputStream();
        var properties = Properties{};
        properties.load(inp);
```

```
        if(properties.get("mapa") != null){

controller.setMapa(Integer.parseInt(properties.get("mapa")));
        }
        argumento_mapa.promptText=properties.get("mapa");
        argumento_mapa.text="";
        inp.close();
    }

    function iniciarPeticionLista():Void{
        var getRequest_config: HttpRequest = HttpRequest {
            location: bind location_lista
            method: HttpRequest.GET

            onStarted: function() {
                println("onStarted   -   started   performing   method:
{getRequest_config.method}                on                location:
{getRequest_config.location}");
            }

            onException: function(ex :Exception){
                if(ex.getMessage()=="Red inalcanzable"){
                    controller.showErrorScreen("error  al  obtener  el
listado:\nservidor no encontrado");
                }
                else{
                    controller.showErrorScreen("error  al  obtener  el
listado: \nrecurso no disponible en el servidor");
                }
                var properties = Properties{};
                checkStoreLista();
                var           outp:           OutputStream           =
res_lista.openOutputStream(false);
                properties.put("numero_mapas","0");
                properties.store(outp);
                outp.close();
                loadLista();
            }

            onInput: function(is: InputStream) {
                var properties = Properties{};
                checkStoreLista();
                var           outp:           OutputStream           =
res_lista.openOutputStream(false);
                properties.load(is);
                properties.store(outp);
                outp.close();
                is.close();
                loadLista();
            }
        }
        getRequest_config.start();
    }


    protected override function createView(): Void {
        if (checkStoreIp() == true){loadIp();}
        if (checkStoreLista() == true){loadLista();}
        if (checkStoreMapa() == true){loadMapa();}
        view = Panel {
```

```
            content: [vBox,imagenPestañas,
                ip,argumento_ip,ok_ip,mapa_config,
                argumento_mapa,ok_mapa,
                lista_mapas,actualizar,list
            ]
        }
    }

}
```

Clase VistaError

```
package videovigilancia;

import javafx.scene.Cursor;
import javafx.geometry.HPos;
import javafx.geometry.VPos;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.scene.input.KeyEvent;
import javafx.scene.layout.HBox;
import javafx.scene.layout.LayoutInfo;
import javafx.scene.layout.VBox;
import javafx.scene.paint.Color;
import javafx.scene.text.Font;
import videovigilancia.Vista;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.scene.text.Text;
import javafx.scene.text.TextAlignment;
import javafx.scene.shape.Rectangle;
import javafx.scene.input.MouseEvent;
import javafx.scene.layout.Panel;
import javafx.scene.text.FontWeight;

public class VistaError extends Vista{

    var buttonWidth = (screenWidth - 10) / 4.0;

    public var textoError: Text = Text {
        font: Font.font("Dialog", FontWeight.BOLD, 12);
        translateX:20
        wrappingWidth: 240-2*20
        textAlignment: TextAlignment.JUSTIFY
        content: bind controller.getTextoError();
        fill: Color.rgb(80,79,78)
    }

    var aceptar: Button = Button {
        cursor: Cursor.HAND
        translateX:90
        translateY:15
        text: "Aceptar"
        action: function() {
            controller.volverVistaAnterior();
        }

        layoutInfo: LayoutInfo {
            width: buttonWidth - 4
        }
        onKeyPressed: function (ke: KeyEvent) {

        }
    }

    var imagenPestañas:ImageView =  ImageView{
        image: Image{
            url: "{__DIR__}images/error2.jpg";
     }
    }
```

```
    var barra:Rectangle = Rectangle {
        cursor: {if (not controller.browser and not controller.mobile
) {Cursor.MOVE} else{Cursor.DEFAULT}}
        width: 240
        height: 13
        onMouseDragged: function(me: MouseEvent) {
            if (not controller.browser and not controller.mobile) {
                controller.X = me.screenX - me.dragAnchorX;
                controller.Y = me.screenY - me.dragAnchorY;
            }
        }
    }

    var mapa:Rectangle = Rectangle {
        width: 240/4
        height: 28
        onMouseClicked: function(me: MouseEvent) {
            controller.showVistaMapa();
        }
    }

    var listado:Rectangle = Rectangle {
        width: 240/4
        height: 28
        onMouseClicked: function(me: MouseEvent) {
            controller.showVistaListado();
        }
    }

    var configuracion:Rectangle = Rectangle {
        width: 240/4
        height: 28
        onMouseClicked: function(me: MouseEvent) {
            controller.showVistaConfig();
        }
    }

    var visor:Rectangle = Rectangle {
        width: 240/4
        height: 28
        onMouseClicked: function(me: MouseEvent) {
            controller.showVistaVisor();
        }
    }

    var pestañas: HBox = HBox {
        content: [mapa, listado, configuracion, visor]
    }

    var headingLabel: Label = Label {
        translateY: 15
        text: "Menu Principal"
        textFill: Color.WHITE
        layoutInfo: LayoutInfo {
            hpos: HPos.CENTER,
            vpos: VPos.CENTER
            height: 30
        }
        font: Font {
```

```
            name: "Bitstream Vera Sans Bold"
            size: 14
        }
    }

    var vBox:VBox = VBox {
        translateY:55
        content: [textoError, aceptar ]

    }

    protected override function createView(): Void {
        view = Panel {
            content: [barra,imagenPestañas,vBox ]
        }
    }

}
```

Clase VistaListado

```
package videovigilancia;

import javafx.scene.Cursor;
import javafx.geometry.HPos;
import javafx.geometry.VPos;
import javafx.scene.control.Label;
import javafx.scene.input.MouseEvent;
import javafx.scene.layout.HBox;
import javafx.scene.layout.LayoutInfo;
import javafx.scene.layout.VBox;
import javafx.scene.paint.Color;
import javafx.scene.shape.Rectangle;
import javafx.scene.text.Font;
import videovigilancia.Vista;
import javafx.scene.control.ListView;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.scene.layout.Panel;

public class VistaListado extends Vista{
    var buttonWidth = (screenWidth - 10) / 4.0;
    var numero_camaras = bind controller.getNumCam();
    var imagenPestañas:ImageView =  ImageView{
        image: Image{
            url: "{__DIR__}images/lista.jpg";
     }
    }

    var barra:Rectangle = Rectangle {
        cursor: {if (not controller.browser and not controller.mobile
) {Cursor.MOVE} else{Cursor.DEFAULT}}
        width: 240
        height: 13
        onMouseDragged: function(me: MouseEvent) {
            if (not controller.browser and not controller.mobile) {
                controller.X = me.screenX - me.dragAnchorX;
                controller.Y = me.screenY - me.dragAnchorY;
            }
        }
    }

    var mapa:Rectangle = Rectangle {
        width: 240/4
        height: 28
        onMouseClicked: function(me: MouseEvent) {
            controller.showVistaMapa();
        }
    }

    var listado:Rectangle = Rectangle {
        width: 240/4
        height: 28
        onMouseClicked: function(me: MouseEvent) {
            controller.showVistaListado();
        }
    }

    var configuracion:Rectangle = Rectangle {
```

```
        width: 240/4
        height: 28
        onMouseClicked: function(me: MouseEvent) {
            controller.showVistaConfig();
        }
    }

    var visor:Rectangle = Rectangle {
        width: 240/4
        height: 28
        onMouseClicked: function(me: MouseEvent) {
            controller.showVistaVisor();
        }
    }

    var pestañas: HBox = HBox {
        cursor: Cursor.HAND
        content: [mapa, listado, configuracion, visor]
    }

    var headingLabel: Label = Label {
        text: "Listado Camaras"
        textFill: Color.WHITE
        layoutInfo: LayoutInfo {
            hpos: HPos.CENTER,
            vpos: VPos.CENTER
            height: 30
        }
        font: Font {
            name: "Bitstream Vera Sans Bold"
            size: 14
        }
    }
    var list: ListView = ListView {
        cursor: Cursor.HAND
        translateY:46
        translateX: 9
        layoutInfo: LayoutInfo {
            width: 223
            height: 224
        }
        items: bind for(x in[0..numero_camaras]){
            "Camara {x}: {controller.descCam[x]}";
        }
        onMouseClicked: function (me: MouseEvent) {
            controller.setCam(list.selectedIndex);
            controller.showVistaVisor()
        }
    }
    var vBox:VBox=VBox {
        nodeHPos:HPos.LEFT;
        content: [barra,pestañas]
    }

    protected override function createView(): Void {
        view = Panel {
            content: [vBox,imagenPestañas,list ]
        }
    }
}
```

Clase VistaMapa

```
package videovigilancia;

import javafx.geometry.HPos;
import javafx.geometry.VPos;
import javafx.scene.control.Label;
import javafx.scene.input.MouseEvent;
import javafx.scene.layout.HBox;
import javafx.scene.layout.LayoutInfo;
import javafx.scene.layout.VBox;
import javafx.scene.paint.Color;
import javafx.scene.shape.Rectangle;
import javafx.scene.text.Font;
import videovigilancia.Vista;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.scene.layout.Container;
import javafx.scene.layout.Panel;
import javafx.io.http.HttpRequest;
import javafx.util.Properties;
import java.io.InputStream;
import java.io.OutputStream;
import java.io.IOException;
import java.lang.Exception;
import javafx.io.Resource;
import javafx.io.Storage;
import javafx.scene.Cursor;

public class VistaMapa extends Vista{

    var mapa_actual = controller.getMapa();
    var ip = controller.getIp();
    var location_lista;
    var numero_camaras;
    def maximo_camaras = 15;
    var location_request = "";
    var storage_mapa: Storage;
    var res_mapa: Resource;

    var startX;
    var startY;
    var xFondoMax = bind fondo.image.width - 224 +xFondoMin;
    var yFondoMax = bind fondo.image.height - 224+yFondoMin;
    var xFondoMin:Number = bind fondo.image.width *(1+escala_fondo)/2
- fondo.image.width;
    var yFondoMin:Number = bind fondo.image.height *(1+escala_fondo)/2
- fondo.image.height;
    var xFondo:Number = 0;
    var yFondo:Number = 0;
    var escala fondo:Number = 1.0;
    var xCam = [1..maximo_camaras];
    var yCam = [1..maximo_camaras];
    var                          visibleCam                          =
[false,false,false,false,false,false,false,false,false,false,false,fal
se,false,false,false];

    var icoCam = "{__DIR__}images/cam.png";
    var mapadir = "{__DIR__}images/nodisponible.jpg";
```

```
    var imagen_fondo:Image = bind Image{
            url: mapadir
    }

    var fondo:ImageView =  ImageView{
        cursor: Cursor.MOVE
        image: bind imagen_fondo
    }

    var zoomIn:ImageView = ImageView {
        cursor: Cursor.HAND
        x: 8
        y: 272
        image: Image {
            url: "{__DIR__}images/MapZoomIn.png"
        }
        onMouseClicked: function(e: MouseEvent):Void{
            escala_fondo= escala_fondo + 0.05;
            if (escala_fondo > 1 ) {escala_fondo =1;}
        }
        visible: true
        opacity:1
    }

    var zoomOut:ImageView = ImageView {
        cursor: Cursor.HAND
        x: 40
        y: 272
        image: Image {
            url: "{__DIR__}images/MapZoomOut.png"
        }
        onMouseClicked: function(e: MouseEvent):Void{
            escala_fondo= escala_fondo - 0.05;
            if (escala_fondo < 224 / imagen_fondo.height) {
                escala_fondo = 224 / imagen_fondo.height;
            }
            if (escala_fondo < 224 / imagen_fondo.width) {
                escala_fondo = 224 / imagen_fondo.height;
            }
            if (xFondo>xFondoMin){xFondo=xFondoMin;}
            if (yFondo>yFondoMin){yFondo=yFondoMin;}
            if (xFondo<-xFondoMax){xFondo=-xFondoMax}
            if (yFondo<-yFondoMax){yFondo=-yFondoMax}
        }
        visible: true
        opacity:1
    }

    var flechaDerecha:ImageView = ImageView {
        cursor: Cursor.HAND
        x: 205
        y: 274
        image: Image {
            url: "{__DIR__}images/arrow.png"
        }
        onMouseClicked: function(e: MouseEvent):Void{
            xFondo = xFondo-50;
            if (xFondo<-xFondoMax){
                xFondo=-xFondoMax;
            }
```

```
        }
        visible: true
        opacity:1
    }

    var flechaIzquierda:ImageView = ImageView {
        cursor: Cursor.HAND
        x: 180
        y: 274
        rotate: 180
        image: Image {
            url: "{__DIR__}images/arrow.png"
        }
        onMouseClicked: function(e: MouseEvent):Void{
         xFondo = xFondo+50;
            if (xFondo>xFondoMin){
                xFondo=xFondoMin;
            }
        }
        visible: true
        opacity:1
    }

    var flechaAbajo:ImageView = ImageView {
        cursor: Cursor.HAND
        x: 155
        y: 274
        rotate: 90
        image: Image {
            url: "{__DIR__}images/arrow.png"
        }
     onMouseClicked: function(e: MouseEvent):Void{
         yFondo = yFondo-50;
            if (yFondo<-yFondoMax){
                yFondo=-yFondoMax
            }
        }
        visible: true
        opacity:1
    }

    var flechaArriba:ImageView = ImageView {
        cursor: Cursor.HAND
        x: 130
        y: 274
        rotate: 270
        image: Image {
            url: "{__DIR__}images/arrow.png"
        }
        onMouseClicked: function(e: MouseEvent):Void{
            yFondo = yFondo+50;
            if (yFondo>yFondoMin){
                yFondo=yFondoMin;
            }
        }
        visible: true
        opacity:1
    }

    var flechas : Container= Container {
```

```
        content: [
            flechaArriba,flechaAbajo,flechaDerecha,flechaIzquierda
        ]
    }

    var Cam0:ImageView = ImageView {
        cursor: Cursor.HAND
        x: bind xCam[0]
        y: bind yCam[0]
        image: Image {url: icoCam}
        onMouseClicked: function(e: MouseEvent):Void{
            if (not flechaAbajo.hover
            and not flechaArriba.hover
            and not flechaIzquierda.hover
            and not flechaDerecha.hover){
                controller.setCam(0);
                controller.showVistaVisor();
            }
        }
        visible:bind visibleCam[0];
    }

    var Cam1:ImageView = ImageView {
        cursor: Cursor.HAND
        x: bind xCam[1]
        y: bind yCam[1]
        image: Image {url: icoCam}
            onMouseClicked: function(e: MouseEvent):Void{
                if (not flechaAbajo.hover
                and not flechaArriba.hover
                and not flechaIzquierda.hover
                and not flechaDerecha.hover){
                    controller.setCam(1);
                    controller.showVistaVisor();
                }
            }
        visible:bind visibleCam[1];
    }


    var Cam2:ImageView = ImageView {
        cursor: Cursor.HAND
        x: bind xCam[2]
        y: bind yCam[2]
        image: Image {url: icoCam}
        onMouseClicked: function(e: MouseEvent):Void{
            if (not flechaAbajo.hover
            and not flechaArriba.hover
            and not flechaIzquierda.hover
            and not flechaDerecha.hover){
                controller.setCam(2);
                controller.showVistaVisor();
            }
        }
        visible:bind visibleCam[2];
    }

    var Cam3:ImageView = ImageView {
        cursor: Cursor.HAND
        x: bind xCam[3]
```

```
        y: bind yCam[3]
        image: Image {url: icoCam}
     onMouseClicked: function(e: MouseEvent):Void{
            if (not flechaAbajo.hover
            and not flechaArriba.hover
            and not flechaIzquierda.hover
            and not flechaDerecha.hover){
                controller.setCam(3);
                controller.showVistaVisor();
            }
        }
        visible:bind visibleCam[3];
    }

    var Cam4:ImageView = ImageView {
        cursor: Cursor.HAND
        x: bind xCam[4]
        y: bind yCam[4]
        image: Image {url: icoCam}
     onMouseClicked: function(e: MouseEvent):Void{
            if (not flechaAbajo.hover
            and not flechaArriba.hover
            and not flechaIzquierda.hover
            and not flechaDerecha.hover){
                controller.setCam(4);
                controller.showVistaVisor();
            }
        }
        visible:bind visibleCam[4];
    }

    var Cam5:ImageView = ImageView {
        cursor: Cursor.HAND
        x: bind xCam[5]
        y: bind yCam[5]
        image: Image {url: icoCam}
        onMouseClicked: function(e: MouseEvent):Void{
            if (not flechaAbajo.hover
            and not flechaArriba.hover
            and not flechaIzquierda.hover
            and not flechaDerecha.hover){
                controller.setCam(5);
                controller.showVistaVisor();
            }
        }
        visible:bind visibleCam[5];
    }

    var Cam6:ImageView = ImageView {
        cursor: Cursor.HAND
        x: bind xCam[6]
        y: bind yCam[6]
        image: Image {url: icoCam}
     onMouseClicked: function(e: MouseEvent):Void{
            if (not flechaAbajo.hover
            and not flechaArriba.hover
            and not flechaIzquierda.hover
            and not flechaDerecha.hover){
                controller.setCam(6);
                controller.showVistaVisor();
```

```
            }
        }
        visible:bind visibleCam[6];
    }

    var Cam7:ImageView = ImageView {
        cursor: Cursor.HAND
        x: bind xCam[7]
        y: bind yCam[7]
        image: Image {url: icoCam}
        onMouseClicked: function(e: MouseEvent):Void{
            if (not flechaAbajo.hover
            and not flechaArriba.hover
            and not flechaIzquierda.hover
            and not flechaDerecha.hover){
                controller.setCam(7);
                controller.showVistaVisor();
            }
        }
        visible:bind visibleCam[7];
    }

    var Cam8:ImageView = ImageView {
        cursor: Cursor.HAND
        x: bind xCam[8]
        y: bind yCam[8]
        image: Image {url: icoCam}
        onMouseClicked: function(e: MouseEvent):Void{
            if (not flechaAbajo.hover
            and not flechaArriba.hover
            and not flechaIzquierda.hover
            and not flechaDerecha.hover){
                controller.setCam(8);
                controller.showVistaVisor();
            }
        }
        visible:bind visibleCam[8];
    }

    var Cam9:ImageView = ImageView {
        cursor: Cursor.HAND
        x: bind xCam[9]
        y: bind yCam[9]
        image: Image {url: icoCam}
     onMouseClicked: function(e: MouseEvent):Void{
            if (not flechaAbajo.hover
            and not flechaArriba.hover
            and not flechaIzquierda.hover
            and not flechaDerecha.hover){
                controller.setCam(9);
                controller.showVistaVisor();
            }
        }
        visible:bind visibleCam[9];
    }

    var Cam10:ImageView = ImageView {
        cursor: Cursor.HAND
        x: bind xCam[10]
        y: bind yCam[10]
```

```
        image: Image {url: icoCam}
     onMouseClicked: function(e: MouseEvent):Void{
            if (not flechaAbajo.hover
            and not flechaArriba.hover
            and not flechaIzquierda.hover
            and not flechaDerecha.hover){
                controller.setCam(10);
                controller.showVistaVisor();
            }
        }
        visible:bind visibleCam[10];
    }

    var Cam11:ImageView = ImageView {
        cursor: Cursor.HAND
        x: bind xCam[11]
        y: bind yCam[11]
        image: Image {url: icoCam}
     onMouseClicked: function(e: MouseEvent):Void{
            if (not flechaAbajo.hover
            and not flechaArriba.hover
            and not flechaIzquierda.hover
            and not flechaDerecha.hover){
                controller.setCam(11);
                controller.showVistaVisor();
            }
        }
        visible:bind visibleCam[11];
    }

    var Cam12:ImageView = ImageView {
        cursor: Cursor.HAND
        x: bind xCam[12]
        y: bind yCam[12]
        image: Image {url: icoCam}
     onMouseClicked: function(e: MouseEvent):Void{
            if (not flechaAbajo.hover
            and not flechaArriba.hover
            and not flechaIzquierda.hover
            and not flechaDerecha.hover){
                controller.setCam(12);
                controller.showVistaVisor();
            }
        }
        visible:bind visibleCam[12];
    }

    var Cam13:ImageView = ImageView {
        cursor: Cursor.HAND
        x: bind xCam[13]
        y: bind yCam[13]
        image: Image {url: icoCam}
     onMouseClicked: function(e: MouseEvent):Void{
            if (not flechaAbajo.hover
            and not flechaArriba.hover
            and not flechaIzquierda.hover
            and not flechaDerecha.hover){
                controller.setCam(13);
                controller.showVistaVisor();
            }
```

```
        }
        visible:bind visibleCam[13];
    }

    var Cam14:ImageView = ImageView {
        cursor: Cursor.HAND
        x: bind xCam[14]
        y: bind yCam[14]
        image: Image {url: icoCam}
        onMouseClicked: function(e: MouseEvent):Void{
            if (not flechaAbajo.hover
            and not flechaArriba.hover
            and not flechaIzquierda.hover
            and not flechaDerecha.hover){
                controller.setCam(14);
                controller.showVistaVisor();
            }
        }
        visible:bind visibleCam[14];
    }


    var imagenPestañas:ImageView =  ImageView{
        image: Image{
            url: "{__DIR__}images/mapa.png";
     }
    }

    var barra:Rectangle = Rectangle {
        cursor: {if (not controller.browser and not controller.mobile
) {Cursor.MOVE} else{Cursor.DEFAULT}}
        width: 240
        height: 13
        onMouseDragged: function(me: MouseEvent) {
            if (not controller.browser and not controller.mobile) {
                controller.X = me.screenX - me.dragAnchorX;
                controller.Y = me.screenY - me.dragAnchorY;
            }
        }
    }

    var mapa:Rectangle = Rectangle {
        width: 240/4
        height: 28
        onMouseClicked: function(me: MouseEvent) {
            controller.showVistaMapa();
        }
    }

    var listado:Rectangle = Rectangle {
        width: 240/4
        height: 28
        onMouseClicked: function(me: MouseEvent) {
            controller.showVistaListado();
        }
    }

    var sur:Rectangle = Rectangle {
        cursor: Cursor.DEFAULT
        translateY:270
```

```
        width: 240
        height: 50
    }

    var configuracion:Rectangle = Rectangle {
        width: 240/4
        height: 28
        onMouseClicked: function(me: MouseEvent) {
            controller.showVistaConfig();
        }
    }

    var visor:Rectangle = Rectangle {
        width: 240/4
        height: 28
        onMouseClicked: function(me: MouseEvent) {
            controller.showVistaVisor();
        }
    }

    var pestañas: HBox = HBox {
        cursor: Cursor.HAND
        content: [mapa, listado, configuracion, visor]
    }

    var headingLabel: Label = Label {
        text: "Mapa"
        textFill: Color.WHITE
        layoutInfo: LayoutInfo {
            hpos: HPos.CENTER,
            vpos: VPos.CENTER
            height: 30
        }
        font: Font {
            name: "Bitstream Vera Sans Bold"
            size: 14
        }
    }

    var group : Container= Container {
        scaleX : bind escala_fondo
        scaleY : bind escala_fondo
        translateX:bind xFondo + 9;
        translateY:bind yFondo + 46;

        onMousePressed: function(me: MouseEvent) {
            startX = xFondo;
            startY = yFondo;
        }

        onMouseDragged: function(me: MouseEvent) {
            if (not flechaAbajo.hover
            and not flechaArriba.hover
            and not flechaIzquierda.hover
            and not flechaDerecha.hover
            and not barra.pressed
            and not imagenPestañas.hover){
                if( xFondo<=xFondoMin
                and xFondo>=-xFondoMax
                and yFondo<=yFondoMin
```

```
                and yFondo>=-yFondoMax){
                    xFondo = startX + me.dragX;
                    yFondo = startY + me.dragY;
                    if (xFondo>xFondoMin){xFondo=xFondoMin;}
                    if (yFondo>yFondoMin){yFondo=yFondoMin;}
                    if (xFondo<-xFondoMax){xFondo=-xFondoMax}
                    if (yFondo<-yFondoMax){yFondo=-yFondoMax}
                }
            }
        }
        content: [
            fondo,Cam0,Cam1,Cam2,Cam3,Cam4,Cam5,Cam6,
            Cam7,Cam8,Cam9,Cam10,Cam11,Cam12,Cam13,Cam14
        ]
    }

    var vBox = VBox {
        content: [barra,pestañas]
    }

    protected function crearMapa(){
        if(ip != controller.getIp()){
            escala_fondo=1;
            xFondo=0;
            yFondo=0;
            ip = controller.getIp();

location_request="http://{ip}/config/mapa{mapa_actual}.txt";
            iniciarPeticion();
        }
        if(mapa_actual!=controller.getMapa()){
            escala_fondo=1;
            xFondo=0;
            yFondo=0;
            mapa_actual=controller.getMapa();

location_request="http://{ip}/config/mapa{mapa_actual}.txt";
            iniciarPeticion();
        }

    }

    public function checkStoreMapa() : Boolean {
        var ret: Boolean = false;
        try {
            storage_mapa = Storage {
            source: "Mapa"
            }
            res_mapa = storage_mapa.resource;
            ret = false;
        }
        catch (e : IOException ) {
            res_mapa = storage_mapa.resource;
            ret = true;
        }
        ret = true;
    }

    public function loadMapa() : Void {
        var inp: InputStream = res_mapa.openInputStream();
```

```
        var properties = Properties{};
        properties.load(inp);
        if(properties.get("numero_camaras") != null){
            numero_camaras                                              =
Integer.parseInt(properties.get("numero_camaras"));
            numero_camaras--;
            controller.setNumCam(numero_camaras);
        }
        else{controller.showErrorScreen("error al obtener el numero de
camaras");}
        if(properties.get("mapa") != null){
            mapadir = properties.get("mapa");
        }
        else{controller.showErrorScreen("error al obtener el mapa");}
        fondo.cursor = Cursor.MOVE;
        if(numero_camaras < maximo_camaras){
            for(x in [0..maximo_camaras]){
                xCam[x]=0;
                yCam[x]=0;
                visibleCam[x]=false;
                controller.setDesc("",x);
            }
            for(x in [0..numero_camaras]){
                if(properties.get("camara{x}X") != null){

xCam[x]=Integer.parseInt(properties.get("camara{x}X"));
                    if(properties.get("camara{x}Y") != null){

yCam[x]=Integer.parseInt(properties.get("camara{x}Y"));
                        visibleCam[x]=true;

controller.setDesc(properties.get("camara{x}desc"),x);
                    }
                    else{
                        visibleCam[x]=false;
                        controller.showErrorScreen("error  al  obtener
la camara{x}: \n la camara no sera visible");
                    }
                }
                else{
                    visibleCam[x]=false;
                    controller.showErrorScreen("error  al  obtener  la
camara{x}: \n la camara no sera visible");
                }
            }
        }

        inp.close();

    }
    function iniciarPeticion(): Void{

        var getRequest_config: HttpRequest = HttpRequest {
            location: bind location_request
            method: HttpRequest.GET

            onStarted: function() {
                fondo.cursor = Cursor.WAIT;
                mapadir = "{__DIR__}images/cargando.jpg";
                for(x in [0..maximo_camaras]){
```

```
                    xCam[x]=0;
                    yCam[x]=0;
                    visibleCam[x]=false;
                    controller.setDesc("",x);
                }
                println("onStarted - started performing method:
{getRequest_config.method} on location:
{getRequest_config.location}");
            }

            onException: function(ex :Exception){
                fondo.cursor = Cursor.MOVE;
                if(ex.getMessage()=="Red inalcanzable"){
                    controller.showErrorScreen("error al obtener el
mapa:\nservidor no encontrado");
                }
                else{
                    controller.showErrorScreen("error al obtener el
mapa: \nrecurso no disponible en el servidor");
                }

                var properties = Properties{};
                checkStoreMapa();
                var outp: OutputStream =
res_mapa.openOutputStream(false);

properties.put("mapa","{__DIR__}images/nodisponible.jpg");
                properties.put("numero_camaras","0");
                properties.store(outp);
                outp.close();
                loadMapa();
                controller.setNumCam(-1);
            }
            onInput: function(is: InputStream) {
                var properties = Properties{};
                checkStoreMapa();
                var outp: OutputStream =
res_mapa.openOutputStream(true);
                properties.load(is);
                properties.store(outp);
                outp.close();
                is.close();
                loadMapa();
            }
        }
        getRequest_config.start();
    }
    protected override function createView(): Void {
        mapa_actual=controller.getMapa();
        location_request="http://{ip}/config/mapa{mapa_actual}.txt";
        if(checkStoreMapa()==true){loadMapa();}
        else if(ip!=null){iniciarPeticion();}
        view = Panel {
            content: [
                group,vBox,sur,imagenPestañas,
                flechas,zoomIn,zoomOut
            ]
        }
    }
}
```

Clase VistaPrincipal

```
package videovigilancia;

import javafx.scene.Cursor;
import javafx.geometry.HPos;
import javafx.geometry.VPos;
import javafx.scene.control.Label;
import javafx.scene.layout.HBox;
import javafx.scene.layout.LayoutInfo;
import javafx.scene.layout.VBox;
import javafx.scene.paint.Color;
import javafx.scene.text.Font;
import videovigilancia.Vista;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.scene.shape.Rectangle;
import javafx.scene.input.MouseEvent;
import javafx.scene.layout.Panel;

public class VistaPrincipal extends Vista{

    var imagen:ImageView = ImageView{
        translateY:5
        translateX:9
        fitWidth: 223;
        fitHeight: 224;
     image: Image{
           url: "{__DIR__}images/jfx.png";
     }
    }

    var imagenPestañas:ImageView =  ImageView{
        image: Image{
            url: "{__DIR__}images/principal.jpg";
     }
    }

    var barra:Rectangle = Rectangle {
        cursor: {if (not controller.browser and not controller.mobile
) {Cursor.MOVE} else{Cursor.DEFAULT}}
        width: 240
        height: 13
        onMouseDragged: function(me: MouseEvent) {
            if (not controller.browser and not controller.mobile) {
                controller.X = me.screenX - me.dragAnchorX;
                controller.Y = me.screenY - me.dragAnchorY;
            }
        }
    }

    var mapa:Rectangle = Rectangle {
        width: 240/4
        height: 28
        onMouseClicked: function(me: MouseEvent) {
            controller.showVistaMapa();
        }
    }

    var listado:Rectangle = Rectangle {
```

```
        width: 240/4
        height: 28
        onMouseClicked: function(me: MouseEvent) {
            controller.showVistaListado();
        }
    }

    var configuracion:Rectangle = Rectangle {
        width: 240/4
        height: 28
        onMouseClicked: function(me: MouseEvent) {
            controller.showVistaConfig();
        }
    }

    var visor:Rectangle = Rectangle {
        width: 240/4
        height: 28
        onMouseClicked: function(me: MouseEvent) {
            controller.showVistaVisor();
        }
    }

    var pestañas: HBox = HBox {
        cursor: Cursor.HAND
        content: [mapa, listado, configuracion, visor]
    }

    var headingLabel: Label = Label {
        translateY: 15
        text: "Menu Principal"
        textFill: Color.WHITE
        layoutInfo: LayoutInfo {
            hpos: HPos.CENTER,
            vpos: VPos.CENTER
            height: 30
        }
        font: Font {
            name: "Bitstream Vera Sans Bold"
            size: 14
        }
    }

    var vBox = VBox {
        content: [barra,pestañas,imagen]
    }

    protected override function createView(): Void {
        view = Panel {
            content: [vBox,imagenPestañas ]
        }
    }

}
```

Clase VistaVisor

```
package videovigilancia;

import javafx.scene.Cursor;
import javafx.animation.Timeline;
import javafx.animation.KeyFrame;
import javafx.geometry.HPos;
import javafx.scene.control.Label;
import javafx.scene.layout.HBox;
import javafx.scene.layout.VBox;
import javafx.scene.paint.Color;
import javafx.scene.shape.Rectangle;
import javafx.scene.text.Font;
import videovigilancia.Vista;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.scene.layout.Panel;
import javafx.scene.input.MouseEvent;

public class VistaVisor extends Vista{
    var ipserv = bind controller.getIp();
    var cam_actual: Integer= bind controller.getCam();
    var mapa_actual: Integer= bind controller.getMapa();
    var camara_detectada: Boolean = false;
    var no_camara = "{__DIR__}images/nodisponible2.jpg";
    var texto_camara: String;
    var url_imagen = no_camara;
    var imagen: Image = Image{
        url: url_imagen;
    }
    var cam: ImageView = ImageView {
        cache:false
        translateY:72;
        translateX:9;
        fitWidth:223;
        preserveRatio:true;
        image: imagen
    }

    var imagenPestañas:ImageView =  ImageView{
        image: Image{
            url: "{__DIR__}images/visor.png";
        }
    }

    var barra:Rectangle = Rectangle {
        cursor: {if (not controller.browser and not controller.mobile
) {Cursor.MOVE} else{Cursor.DEFAULT}}
        width: 240
        height: 13
        onMouseDragged: function(me: MouseEvent) {
            if (not controller.browser and not controller.mobile) {
                controller.X = me.screenX - me.dragAnchorX;
                controller.Y = me.screenY - me.dragAnchorY;
            }
        }
    }

    var mapa:Rectangle = Rectangle {
```

```
        width: 240/4
        height: 28
        onMouseClicked: function(me: MouseEvent) {
            cerrarVisor();
            controller.showVistaMapa();
        }
    }

    var listado:Rectangle = Rectangle {
        width: 240/4
        height: 28
        onMouseClicked: function(me: MouseEvent) {
            cerrarVisor();
            controller.showVistaListado();
        }
    }

    var configuracion:Rectangle = Rectangle {
        width: 240/4
        height: 28
        onMouseClicked: function(me: MouseEvent) {
            cerrarVisor();
            controller.showVistaConfig();
        }
    }

    var visor:Rectangle = Rectangle {
        width: 240/4
        height: 28
        onMouseClicked: function(me: MouseEvent) {
        }
    }

    var pestañas: HBox = HBox {
        cursor: Cursor.HAND
        content: [mapa, listado, configuracion, visor]
    }

    var cam desc: Label = Label {
        translateY: 247
        translateX: 12
        width:140
        textWrap:true
        text: bind texto_camara
        textFill: Color.YELLOW
        visible:bind camara_detectada
        font: Font {
            name: "Bitstream Vera Sans Bold"
            size: 12
        }
    }

    var vbox = VBox {
        nodeHPos:HPos.LEFT;
        content: [barra,pestañas]
    }

    var timeline: Timeline = Timeline{
        repeatCount: Timeline.INDEFINITE
        keyFrames:[
```

```
            KeyFrame{
                time   :   if   (controller.mobile   ==   true){0.75s}
else{0.25s}
                action : renovarUrl
            }
        ]
    }

    protected function setTextoCamara(): Void {
        texto_camara="camara                                {cam_actual}:
{controller.getDesc(cam_actual)}"
    }

    protected function iniciarVisor(): Void {
        if(cam_actual != -1){
            camara_detectada=false;
            timeline.playFromStart();
            setTextoCamara()
        }
        else{cam.image=imagen;}
    }

    protected function cerrarVisor(): Void {
        timeline.stop();
        cam.image=imagen;
    }

    function renovarUrl() : Void{
        var imagen2: Image = Image{
            url:
"http://{ipserv}/mapa{mapa_actual}/camara{cam_actual}/cam.jpg";
        }
        cam.image=imagen2;
        if(imagen2.error==false){camara_detectada=true}
        if(imagen2.error==true and camara_detectada==false){
            cerrarVisor();
            controller.showErrorScreen("la     camara     {cam_actual}:
{controller.getDesc(cam_actual)} \nno se encuentra disponible");
            url imagen = no camara;
            controller.setCam(-1);
        }
    }

    protected override function createView(): Void {
            view = Panel {
                content: [vbox,cam_desc,imagenPestañas,cam ]
            }
    }

}
```

Bibliografía

[Alberola, 2009] Gustavo Alejandro Alberola y Matías Álvarez Durán: ***"JavaFX, ahora el límite lo pone tu imaginación…"***. Seminario, 2009.

[Anderson, 2009] Gail Anderson y Paul Anderson: ***"Essential JavaFX"***. Editoral Prentice Hall, 2009.

[API JFX 1.2] http://java.sun.com/javafx/1.2/docs/api/

[Burbeck, 1992] http://st-www.cs.illinois.edu/users/smarch/st-docs/mvc.html

[Eguíluz, 2008] Javier Eguíluz Pérez: ***"Introducción a AJAX"*** 2008

[JFX 1.2 Release Notes, 2009] http://www.javafx.com/downloads/mobile/ReleaseNotes.pdf

[Montero, 2009] Beatriz Montero Casado: ***"Rich Internet Applications (RIA): comparativa de frameworks en Software Libre"*.** Artículo, 2009.

[Morris, 2010] Simon Morris: "JavaFX in action". Editorial Manning, 2010.

[Rapoza, 2008] http://etech.eweek.com/content/application_development/ria_war_is_brewing.html

[Stamos, 2008] Alex Stamos, David Thiel y Justine Osborne ***"Living in the RIA world".*** 2008

[Weaver, 2007] James L. Weaver: ***"JavaFX Script"***. Editorial Apress, 2007

[Web Adobe Air] http://www.adobe.com/es/products/air/

[Web Adobe Flash] http://www.adobe.com/es/products/flash/?promoid=BPBIQ

[Web Adobe Flex] http://www.adobe.com/es/products/flex/?promoid=BPBJI

[Web ConnectVU] http://www.cctvmobile.com/index.php

[Web GWT] http://code.google.com/intl/es/webtoolkit/overview.html

[Web JavaFX: API] http://java.sun.com/javafx/1.3/docs/api/

[Web JavaFX: Media] http://javafx.com/samples/MediaBox/index.html

[Web JavaFX: Script] http://java.sun.com/javafx/1/tutorials/core/

[Web Live2Phone] http://www.live2phone.com/

[Win Mob Install Guide, 2009] http://www.javafx.com/downloads/mobile/InstallGuide.pdf

[Web Netvision] http://www.verextech.com/index.php?option=com_content&task=view&id=41

[Web Oficial JavaFX] www.javafx.com

[Web Oficial Mplayer] http://www.mplayerhq.hu/design7/news.html

[Web Oficial On2] http://www.on2.com/index.php?387

[Web Oficial Xampp] http://www.apachefriends.org/en/xampp-linux.html

[Web QNAP VSMobile] http://www.qnapsecurity.com/pro_detail_feature.asp?p_id=141

[Web ROM WM6] http://www.pdatungsteno.com/2009/02/16/como-desbloquear-un-telefono-windows-mobile-e-instalar-una-rom-avanzada/

[Web Silverlight] http://msdn.microsoft.com/es-mx/library/cc838158%28VS.95%29.aspx

[Web UGOlog] http://www.ugolog.com/

[Web uWatchIt] http://uwatchit.net/

[Web Vancouver 2010] http://www.vancouver2010.com/olympic-medals/geo-view/

[Web Vitamin D] http://www.vitamindinc.com/index.php

[Web WSM] http://www.athtek.com/webcam-surveillance-monitor.html