Universidad de las Ciencias Informáticas

Facultad de Ciencias y Tecnologías Computacionales

Módulo de visualización de cámaras del Sistema de Video Vigilancia SURIA 3.0 para dispositivos móviles.

Trabajo de diploma para optar por el título de Ingeniero en Ciencias Informáticas

Autores:

José Orlando Pérez Villegas

Diana Zuria Rodriguez Cuervo Arango

Tutor:

Ing. Miguel Morciego Varona

Ing. Guillermo Luzua Farias

La Habana, Junio de 2017

Dedicatoria

Yo José Orlando, no solo dedico esta Tesis de Diploma sino también el Título que alcancé con la realización de la misma a: mi Salvador por su gran Amor, mis suegros Ohtoniel y Dicmary, mi tío Lester, mis hermanos Jaime y Gaby, mi madre Maribel, mi abuela Hilda, Daymí por su cariño y cuidado, mi padre José Pérez.

Yo Diana Zuria dedico esta Tesis de Diploma y mi Título a: Dios por su inmenso Amor y Fidelidad en todo momento, a mi mamá Dicmary por su amor, su sacrificio y cuidado, a mi papá Ohtoniel por su gran sacrificio, por su amor y por cambiar, a mi hermanito querido Ohto por ser especial para mí, esta tesis es tuya, a mi segunda mamá Nivia, a mi tío Renier y Dianelis por encaminarme, a mis abuelos, a mis suegros por su apoyo y a mis cuñaditos Gaby y Jaime.

Agradecimientos

Gracias le doy al Señor por darme la oportunidad y el privilegio de poder estudiar en esta Universidad tan especial, por darme la fuerza y la inteligencia para culminar esta carrera y graduarme.

Gracias a mis profesores, compañeros de aula, compañeros de Proyecto y amigos por darme lo que me faltaba para lograr esto.

Gracias a mis tutores Miguel y Guillermo por su ayuda.

Gracias al pueblo de Dios por su gran apoyo, en especial a: Edisnel, Marcos, Yoandri, Alayo, Alejandro y Mariam, Yuniel y Lisandra, Ángel y Maridalia.

Gracias a mi familia por confiar y estar siempre cerca de mí. A Ohtoniel, Dicmary y Ohto por ser una parte de mí. A Renier y Dianelis por guiarnos en nuestros primeros pasos en la Universidad. A mi tío Lester por su apoyo. A mi madre por su lío con los patos (nota de 2 puntos) en la docencia. A Daymí por siempre estar atenta a todo de mí. A mi padre por ser mi mejor amigo y ser mi primer maestro.

Gracias a mi amada esposa por ser mi compañera estos 5 años de Universidad, por su paciencia, por su cuidado y por su amor.

José Orlando

Agradecimientos

Le doy gracias a Dios por poder darle gracias, porque me rescató de la vida que llevaba y me puso en su camino para servirle y amarle, gracias porque si no fuera por su Amor y su Paciencia no sé qué sería de mí. Gracias por su gran Fidelidad, pues aun cuando yo le he sido infiel, Él ha permanecido siendo Fiel, gracias porque por su gran Misericordia y Gracia soy quien soy. Gracias por permitirme que obtuviera esta carrera, pues aun cuando todos habíamos perdido la esperanza Él estaba obrando en silencio. Gracias Señor por ayudarme a mantenerme firme en tus caminos durante estos 5 años, por ayudarme a crecer espiritualmente, por batallar conmigo, por estar siempre cuando lo necesitaba. Gracias por permitir que me pueda graduar en esta doble universidad. Gracias Señor.

Le doy gracias a las dos personas más especiales en mi vida; mi Mamá Dicmary y mi Papá Ohtoniel. Mimi linda: gracias por tu inmenso sacrificio desde que nací hasta el día de hoy, gracias por tu amor incondicional, por apoyarme en todo momento y por ser quien eres. Papito lindo: gracias por sacrificarte tanto por mí, porque aun cuando estabas cansado decías que no me preocupara, porque aun cuando tus manos estaban dañadas tu seguías por tal de darnos todo, gracias por tu gran amor y tu cariño, por tus abrazos y besos, gracias por ser quien eres. Mamá, Papá gracias por existir.

Le doy gracias a mi hermanito Ohto, gracias por tu cariño, porque lo demuestras diferente a todos los demás, gracias por tus abrazos, gracias por quererme. Esta Tesis y este Título son tuyo.

Le doy gracias a toda mi familia, en especial a mi Tiota y segunda mamá Nivia por su cariño y amor, por sus regalos sin yo merecerlos y por quererme. Gracias a mi tío Renier y Diane, por encaminarme en esta universidad, por ayudarme en todo y por su cariño. Gracias a mi abuelo Tabare, por su amor y por sus trabajitos, gracias a mi abuela Elena. Gracias a mi tío Gustavo y Macuchi, a Yumi, Tavito y a todos mis primos. Gracias a mis suegros por su apoyo durante toda la carrera y por su cariño. Gracias a Gaby por ser mi cuñadita linda y por confiar en mí. Gracias a mi cuñadito Jaime, por ser mi primer hijo, por quererme y amarme.

Gracias a mis tutores Miguel y Guillermo, por su gran ayuda durante toda la tesis. A los profesores y compañeros del aula que me ayudaron a llegar hasta aquí.

Le doy gracias al pueblo de Dios de la UCI, por su ayuda incondicional y por sus oraciones, en especial: Maridalia y Ángel, Yuniel y Lisandra, Alejandro y Marian, Gissel y Ronny, Alayo, Yoandri, Vicet, Marcos, Mercedes y Haydee.

Gracias a mi precioso esposo, por su cuidado, por su amor, por levantarme cuando caigo, por su ayuda en estos 5 años, por ser mi compañero de tesis. Te Amo.

Diana Zuria

Declaración de Autoría

Declaramos ser los autores del trabajo de diploma titulado “Módulo de visualización y control de cámaras del Sistema de Video Vigilancia SURIA 3.0 para dispositivos móviles.” y reconocemos a la Universidad de las Ciencias Informáticas los derechos patrimoniales del mismo, con carácter exclusivo.

Para que así conste se firma el presente a los ____ días del mes de _____ del año 2017.

Autores:

_____	_____
José Orlando Pérez Villegas	Diana Zuria Rodriguez Cuervo Arango

Tutores:

_____	_____
Ing. Miguel Morciego Varona	Ing. Guillermo Luzua Farias

Resumen

El sistema de Video Vigilancia SURIA, desarrollado en el centro Geoinformática y Señales Digitales de la Universidad de las Ciencias Informáticas permite visualizar los flujos de las cámaras de un Circuito Cerrado de Televisión. Igualmente posibilita grabar los datos de videos provenientes de las cámaras que integran el circuito. Esto permite contar con evidencias de lo ocurrido en las áreas protegidas, además la recuperación y análisis de las grabaciones realizadas. Sin embargo, el trabajo de los usuarios de este sistema se ve limitado debido que solo se pueden monitorear las cámaras desde un local a través de ordenadores de escritorio que tengan instalado el sistema. En este contexto surge el presente trabajo, el cual se enmarca en la investigación y desarrollo de una aplicación que permita la visualización de cámaras para el sistema de Video Vigilancia SURIA 3.0 desde dispositivos móviles con Sistema Operativo Android. Con el desarrollo de la mencionada aplicación móvil se provee de una solución que complemente y facilite el trabajo de los usuarios del sistema de video vigilancia, ofreciéndole mayor movilidad sin que se vea limitado la visualización de las imágenes que transmiten las cámaras.

Palabras claves: Cámara, Video Vigilancia, Visualización.

Abstract

The SURIA Video Surveillance system, developed at the GEYSED Center from the University of Computer Sciences, allows the visual monitoring of the cameras of a security system simultaneously. Likewise, it makes possible to record the video streams from the cameras, allowing to have evidence of what happened and also to retrieve and analyze the recordings made. However, the work of users of this system is limited because they can only monitor the cameras from a room through desktop computers running the system. These are the circumstances that require the existance of this work, which focuses of the research and development of an application that allows the visualization of cameras for the SURIA 3.0 system on devices running Android Operating System. With the development of the aforementioned mobile application, a solution that complements and facilitates the work of the users of the video surveillance system is provided, offering greater mobility without being deprived of the visualization of the images transmitted by the cameras.

Keywords: Camera, Video Surveillance, Visualization.

Índice General

Índice de Figuras

Índice de tablas

Introducción

En la actualidad resulta cada vez más familiar el trabajo con la tecnología. Diariamente se hace uso de celulares, computadoras y demás dispositivos con los que interactuamos en el día a día. Las grandes compañías, los centros comerciales, las estaciones de trenes, túneles, bancos, y otros lugares públicos altamente concurridos por personas tienen instaladas cámaras; proporcionando una sensación de seguridad y de tranquilidad mucho mayor, tanto socialmente como para los propietarios de estos lugares. Según (CVRL Industries Inc., 2015) el origen de las cámaras de video vigilancia se remonta al año 1965, donde hubo informes de prensa en los Estados Unidos que sugerían el uso de cámaras de vigilancia policial en los lugares públicos. La práctica se extendió rápidamente con cámaras analógicas en los Sistemas de Circuito Cerrado de Televisión (CCTV), que eran monitorizados por agentes todo el día. Estos sistemas estaban ubicados en locales, donde tenían las condiciones necesarias para que los agentes estuvieran supervisando las cámaras de seguridad con grandes monitores donde se podía visualizar todo el flujo de video.

En 1996 se puso en marcha el cambio hacia las soluciones de video vigilancia basadas en el Protocolo de Internet (Por sus siglas en inglés IP), las cuales son video cámaras de vigilancia que tienen la particularidad de enviar señales de video y audio. Estas pueden estar conectadas directamente a un *Router* ADSL[1], o a un concentrador de una Red Local para que las imágenes se puedan visualizar. A la vez, las cámaras IP permiten el envío de alarmas por medio de correos electrónicos y la grabación de secuencias de imágenes en equipos informáticos situados dentro del CCTV, permitiendo de esta forma verificar posteriormente lo sucedido en el lugar monitorizado (Empresas Reqquality, 2016).

También se encuentran el Acceso Remoto como característica distintiva de las cámaras IP, el mismo permite que la observación y grabación de los eventos, no tengan que realizarse en el lugar como requieren los sistemas CCTV. La instalación es mucho más flexible ya que se basa en la infraestructura de la red local existente, o en la conexión directa a un *Router*, bien por cable o de forma inalámbrica. Otra de las funcionalidades es que las grabaciones de las cámaras IP se realizan en el disco duro de una PC[2] de la propia red, lo que mejora el acceso a los materiales. La ampliación del sistema, es otra ventaja de los CCTV sobre IP, ya que permiten su ampliación con nuevos puntos de monitoreo sin afectar la red creada (Seguridad Vía IP, 2017).

[1] ADSL: Línea de Abonado Digital Asimétrica
[2] PC: Personal Computer

Hoy en día existen variedad de sistemas para visualizar las cámaras IP. En la Universidad de las Ciencias Informáticas (UCI) específicamente en el Centro Geoinformática y Señales Digitales (GEYSED), existe el sistema de Video Vigilancia SURIA, el cual permite monitorear visualmente por el personal destinado las cámaras de un CCTV de manera simultánea. Por otra parte, posibilita grabar los flujos de videos de las cámaras, para contar con evidencias de las actividades que puedan suceder en los puntos de interés que se monitorean. Igualmente permite, la recuperación y análisis de las grabaciones realizadas, además, de operar en cámaras IP de cualquier fabricante conocido, lo cual facilita la integración de diferentes tipos de cámaras en un solo CCTV. Esto conlleva a un alto grado de escalabilidad y adaptabilidad para el sistema SURIA como solución de video vigilancia para áreas de interés.

A pesar de la amplia cantidad de facilidades que puede brindar el sistema de video vigilancia SURIA, en la actualidad la visualización de las cámaras solamente es posible desde un visor de escritorio del cual existen dos versiones, una para computadoras con Sistema Operativo Windows y otra para computadoras con Sistema Operativo Linux, estos ordenadores de escritorio se encuentran en un local específico. Lo que limita al personal de control, monitorizar las cámaras y a la ves desplazarse hacia los lugares cuando ocurren incidencias. Todo el análisis de la evidencia hay que realizarla luego que suceda el hecho, ya que en tiempo real el operador de monitorización no puede abandonar su puesto. Esto trae como consecuencia que la respuesta ante un hecho extraordinario pueda ser limitada.

Igualmente, el interés de los jefes de turno de guardia en fijar la dirección de las cámaras para monitorizar elementos puntuales dado situaciones particulares era comprometido en ocasiones. Esto condicionado a que el jefe de guardia debe realizar recorridos por las áreas y no siempre puede avisarle al operador de monitorización que existen determinadas cámaras que no pueden moverse. El operador que se queda en el local cambia la dirección de las cámaras para revisar mejor las áreas y esto afecta el objetivo inicial por lo cual el jefe de turno fijó la dirección de las cámaras.

De acuerdo con los elementos antes expuestos se define el siguiente **problema de investigación:** ¿Cómo lograr la portabilidad del módulo de visualización de cámaras del sistema de Video Vigilancia SURIA? A partir del problema propuesto se plantea como **objetivo general** desarrollar una aplicación móvil que permita la visualización de las cámaras del sistema de Video Vigilancia SURIA.

Se define como **objeto de estudio** las aplicaciones móviles para sistemas de video vigilancia basados en cámaras IP. Enmarcando la investigación en el **campo de acción**, aplicaciones móviles para visualizar las cámaras del sistema de Video Vigilancia SURIA 3.0.

Para cumplir el objetivo general de la investigación se trazaron las siguientes **preguntas de investigación**:
- ¿Cuáles son los fundamentos teóricos que rigen el desarrollo de aplicaciones compatibles con dispositivos móviles que permiten la visualización de cámaras?

- ¿Cómo ha evolucionado el desarrollo de aplicaciones compatibles con dispositivos móviles que permiten la visualización de cámaras?

- ¿Qué métodos y herramientas hay que tener en cuenta para el desarrollo de aplicaciones compatibles con dispositivos móviles que permiten la visualización de cámaras?

Para dar cumplimiento a las preguntas formuladas anteriormente se hace necesario desarrollar las siguientes **tareas de investigación*:***

- Caracterizar el proceso de visualización de cámaras de video vigilancia.
- Caracterizar las soluciones existentes que responden a la visualización de cámaras en un CCTV sobre IP.
- Caracterizar las principales herramientas, tecnologías y la metodología a utilizar para la construcción de la propuesta de solución.
- Realizar la confección de los artefactos de ingeniería según la metodología seleccionada.
- Implementar la solución del sistema.
- Validar el módulo de visualización de cámaras IP móvil implementado.

La investigación utiliza una serie de **métodos científicos** que guían el proceso investigativo, los cuales se exponen a continuación:

Teóricos:

- Histórico-Lógico: El método histórico estudia la trayectoria real de los fenómenos y acontecimientos en el transcurso de su historia. El método lógico investiga las leyes generales del funcionamiento y desarrollo de los fenómenos (Zayas, 2015). Este método se emplea para analizar la trayectoria y evolución de los conceptos asociados a la visualización de cámaras, lo cual permite formar una definición propia. La etapa que se estudia es desde el 2000 hasta la actualidad.

- Analítico-Sintético: El método analítico permite la descomposición de un todo complejo en sus partes y cualidades. La síntesis, por su parte, establece la unión entre las partes, previamente analizadas y posibilita descubrir relaciones y características generales entre los elementos de la realidad (Zayas, 2015). Este método se emplea para la valoración de las soluciones existentes que responden al campo de acción. Además, se utiliza en la selección de las herramientas y tecnologías que se emplearán en el desarrollo de la solución propuesta.

La presente investigación está estructurada en tres capítulos:

- Capítulo 1: En este capítulo se exponen los fundamentos teóricos relacionados con los conceptos fundamentales de la visualización de cámaras, así como el análisis de las soluciones existentes en la actualidad que cuentan con módulos similares. Además, se describe y define la metodología de desarrollo que va a ser empleada, y las herramientas a utilizar para el diseño e implementación del componente.

- Capítulo 2: En este capítulo se definen los artefactos del diseño y de la implementación de la solución. Se define la estructura del diseño de la aplicación desarrollada, para ello se representa el modelo de dominio, el diagrama de clases del diseño, los requisitos funcionales y no funcionales, se describen los casos de uso del sistema y la arquitectura. Además, se exponen los estándares de codificación utilizados en la implementación de la solución.

- Capítulo 3: En este capítulo se exponen el diagrama de componentes, el diagrama de despliegue, la implementación y los resultados de las pruebas realizadas a la aplicación para verificar su correcto funcionamiento de acuerdo a los requisitos establecidos.

CAPÍTULO I: FUNDAMENTACIÓN TEÓRICA

1.1 Introducción

El presente capítulo expone los conceptos fundamentales que se abordan durante la investigación, realizándose una descripción del objeto de estudio propuesto. De igual manera se exponen las herramientas y tecnologías que se utilizan durante la construcción de la propuesta de solución; se documenta la metodología de software seleccionada y se describen las principales características de cada herramienta utilizada. Además, se enfatiza en los elementos de interés que sustentan la investigación.

1.2 Visualización

En la presente investigación se acotan un conjunto de conceptos referentes a la visualización definidos según (RAE, 2014)

Visualización: Acción y efecto de visualizar.

Visualizar: Hacer visible una imagen en un monitor.

1.3 Dispositivos móviles

Los dispositivos móviles se definen como un grupo de dispositivos electrónicos que poseen las siguientes características: movilidad, tamaño reducido, comunicación inalámbrica e interacción con las personas. Algunos tipos de dispositivos móviles son los teléfonos inteligentes, tabletas, lectores de libros electrónicos y Asistentes Personales Digitales (Por sus siglas en inglés PDA) (Pozo, 2011).

Es necesario comprender que la posibilidad misma del acceso a Internet desde un dispositivo móvil hizo evidente la necesidad de adaptar los estándares del Consorcio de la Web (Por sus siglas en inglés W3C) a las características del dispositivo de bolsillo: menor tamaño y resolución de pantalla, menor capacidad de procesamiento de datos, configuración vertical, problemas específicos de usabilidad en cuanto a entrada de datos y presentación de contenidos. Como consecuencia de estas necesidades surge un nuevo modelo de desarrollo centrado en el usuario, enfocado principalmente a la experiencia del mismo, donde el contenido no es el objetivo esencial (Aguado Terrón, y otros, 2009).

Mediante este nuevo paradigma de desarrollo se comenzaron a tener en cuenta principios de usabilidad que antes no eran tan importantes. Otros conceptos fuertemente ligados a este paradigma de los cuales se van a estar dando una serie de características son: la portabilidad, la adaptabilidad, la usabilidad y el patrón de diseño.

Usabilidad

La usabilidad se define como: *"…la medida en que un producto puede ser utilizado por usuarios especificados para alcanzar los objetivos especificados con eficacia, eficiencia y satisfacción en un contexto específico de uso…"* (UsabilityNet, 2006).

La usabilidad es parte de la Interacción Humano-Computadora (Por sus siglas en inglés HCI), rama dentro de las ciencias de la computación que estudia todos los aspectos relacionados con la interacción entre las personas y las máquinas. Incluye el análisis de los aspectos sociales, físicos, emocionales, ergonómicos, entre otros. Además, se basa en aspectos evaluativos relacionados con las interfaces desde donde el usuario pueda desenvolverse fácilmente, le sea sencillo encontrar lo que busca dentro del sitio, cometa la menor cantidad de errores y sea capaz de realizar un manejo eficiente de los contenidos. Todos estos aspectos se deben cumplir para lograr la satisfacción por parte del usuario al utilizar un sistema (Rubio, 2015).

Portabilidad

Para la informática la portabilidad es: Una característica no funcional de la arquitectura de un software que trata de adaptar un sistema a una variedad de plataformas de hardware diferentes, interfaces de usuario, sistemas operativos, lenguajes de programación o compiladores. Para ser portable un sistema necesita estar organizado de tal manera que su dependencia del hardware, otro software y ambientes sea responsabilidad de componentes especiales tales como bibliotecas de software capaces de abstraer al sistema como un todo de las diferentes variables y plataformas donde puede ejecutarse (Buschmann, y otros, 2001).

Los autores de la presente investigación consideran que para la actual investigación el concepto de portabilidad contendrá todos los elementos relacionados anteriormente y se le acota que es la propiedad de hacer que un objeto sea movible a cualquier lugar.

Adaptabilidad

En términos de software, es la capacidad que tiene el mismo para adaptarse a diferentes entornos especificados (hardware o sistemas operativos) sin que implique reacciones negativas ante el cambio. Incluye la escalabilidad de capacidad interna, es decir, la capacidad de reacción en diferentes campos de batalla, ante diferentes volúmenes de transacciones, formatos de reporte, etc. (Largo García, y otros, 2005).

Patrón de diseño

En la conferencia *Google I/O* del 2014 donde fue presentado el concepto de *Material Design*, se definió que este es una guía completa para el diseño, animación e interacción de las interfaces visuales para todas las plataformas y dispositivos. Está basado en objetos materiales, piezas colocadas en un lugar y con un movimiento determinado, donde la profundidad, las superficies, los bordes, las sombras y los colores juegan un papel principal. Está guiado por las leyes de la física, donde los movimientos son lógicos, los objetos se superponen, pero no pueden atravesarse el uno al otro, los menús poseen una tipografía clara, casillas bien ordenadas, colores e imágenes llamativos para no perder el foco y dar un sentido de orden y jerarquía (Google, 2017).

El concepto de *Material Design* es una estrategia de Google para ganar en uniformidad en las aplicaciones, donde al usuario le sea posible una navegación sencilla e intuitiva. Además de respetar las reglas de usabilidad definidas para la aplicación. Este estándar está enfocado a la experiencia de usuario, por lo que muchas aplicaciones y plataformas han modificado sus interfaces teniendo en cuenta estos principios definidos.

1.4 Descripción del Objeto de estudio

La forma de acceder a las plataformas web desde dispositivos móviles puede variar en dos formas: aplicaciones nativas o aplicaciones web, donde cada una de estas tiene sus ventajas y sus desventajas.

Aplicación Web Móvil

Una aplicación web móvil es una aplicación web con formato para dispositivos móviles y se accede a esta a través de un navegador. Como una aplicación web tradicional, la aplicación web móvil se construye con tres tecnologías: *HTML*[3]*, CSS*[4] *y JavaScript* (Lionbridge, 2016).

[3] HTML: HyperText Markup Language
[4] CSS: Cascading Style Sheets

Ventajas

- La principal ventaja de las aplicaciones web móviles con respecto a las aplicaciones móviles nativas es la compatibilidad entre plataformas, lo que les permite llegar a un público más amplio por el menor esfuerzo. Son relativamente baratas, fáciles y rápidas de construir, aunque en algunos dispositivos específicos por lo general requieren personalización. Los navegadores web para móviles son bastantes estándares, por lo que es mucho más fácil crear una aplicación web móvil universal que una aplicación nativa.
- Las aplicaciones web son más fáciles de mantener que las aplicaciones nativas.
- Acceso simple: los usuarios no tienen que descargar una aplicación, sino simplemente acceder a una URL[5] a través de su navegador móvil.

Desventajas:

- Los navegadores móviles tienen capacidades limitadas en comparación con los navegadores de escritorio tradicionales. Muchas veces la aplicación puede parecer torpe en comparación con la de un navegador web de escritorio.
- Las aplicaciones web tienen un acceso limitado al hardware del dispositivo.
- Las aplicaciones web requieren de una conexión lo que puede ocasionar problemas de rendimiento si el sitio es lento o no está disponible.

Aplicación Nativa

Una aplicación móvil nativa es una aplicación construida para los dispositivos móviles con un sistema operativo específico. Las aplicaciones nativas tienen una gran ventaja sobre las aplicaciones web: la capacidad de aprovechar el hardware y el software específico del dispositivo (Lionbridge, 2016).

Ventajas

- Una experiencia de usuario más convincente: Las aplicaciones nativas pueden aprovechar las capacidades del dispositivo móvil, incluyendo el hardware como el GPS, la cámara y gráficos, y el software como el correo electrónico, calendario, contactos, galería de fotos / video, administradores de archivos.

[5] URL: Localizador de Recursos Uniforme

- Capacidad de ejecutarse *off-line*: Dado que la aplicación está instalada en el dispositivo, no se requiere conexión a Internet. Los usuarios pueden obtener el máximo rendimiento en todo momento, con todos los gráficos, imágenes, secuencias de comandos y datos. En caso de pérdida de conexión, la transferencia de datos se puede reanudar cuando se restablece la misma.
- En caso de querer conseguir ganancias económicas con las aplicaciones nativas es mucho más fácil, pues se obtienen mediante la compra de la misma en una tienda de aplicaciones, mientras que las aplicaciones web garantizan esto mediante una suscripción.

Desventajas

- Para las aplicaciones nativas puede resultar difícil el acceso desde distintas plataformas, ya que presentan requisitos predefinidos para su uso.
- Si el sistema sufre modificaciones, es más costoso realizar cambios a la aplicación nativa para adaptarla a las nuevas funcionalidades.

Las aplicaciones web móvil presentan tentativas ventajas para el desarrollo, pero viendo las necesidades que requiere la aplicación a desarrollar, como por ejemplo un uso más eficiente y completo de los recursos del hardware del dispositivo, se selecciona como mejor opción una aplicación nativa.

1.5 Soluciones existentes

MOBOTIX Video Security

Mobotix Remote Control es una aplicación para Android específica para controlar las cámaras *Mobotix*. Este puede controlar las alarmas, el sistema de mensajería, encender luces, abrir la puerta, activar y desactivar la grabación. Igualmente permite hacer que la cámara llame por teléfono al número que le indique, generar un evento desde el teléfono y además podrás comprobar los estados de las cámaras y visualizar las mismas en directo. Esta versión tiene un Servidor IP incorporado que permite recibir mensajes de texto gratis provenientes de cualquier cámara *Mobotix* o recibir una imagen cuando la cámara detecta movimiento y guardar la imagen en la tarjeta SD[6]. El software implementa un cliente *DynDns*[7] para poder actualizar la dirección IP cuando se está conectado mediante *3G*. Esto permite poder recibir los mensajes de texto o imágenes cada vez que la cámara detecte movimiento, estos están disponibles en el idioma

[6] SD: Secure Digital
[7] DynDns: Compañía de Internet de los Estados Unidos de América dedicada a soluciones de DNS en direcciones IP dinámicas

alemán, español, inglés e italiano. Esta estructura centralizada tradicional es inadecuada para sistemas de video de alta resolución, ya que requiere no sólo un ancho de banda de red alto, sino que también necesita un enorme poder de procesamiento de PC para soportar varias cámaras (Google Play, 2016).

Elementos de Mobotix identificados de interés para la investigación:

- Visualizar el flujo de una cámara.
- Mostrar el estado de las cámaras.
- Controlar alarmas asociadas a sucesos.
- Informar la ocurrencia de eventos a través de notificaciones.
- Iniciar y detener la grabación en una cámara.

Elementos de Mobotix que generan limitantes para la investigación:

- Solo visualiza cámaras fabricadas por *Mobotix*, lo cual limita las potencialidades del Sistema de Video Vigilancia SURIA 3.0 que persigue el objetivo de visualizar cámaras sin importar fabricantes o características de las mismas.
- No se integra con el sistema de Video Vigilancia SURIA 3.0 pues es de código cerrado.

iViewer de VIVOTEK

iViewer permite a los usuarios monitorizar video en directo procedente de cientos de cámaras gestionadas por el software de gestión centralizada *VAST*, o del software estándar *ST7501*, todo ello en cualquier momento, y desde cualquier lugar donde se encuentren. Este acceso siempre disponible, no sólo es una gran comodidad para los administradores, sino que además mejora la seguridad, dotando a estos administradores de flexibilidad para controlar focos de tensión, mientras están fuera de la instalación, permitiendo detectar actividades sospechosas en tiempo real. Con respecto a la reproducción, soporta cámara de *"visión de ojo de pez"*, la configuración de las vistas desde cámara panorámica *"ojo de pez"* y la visualización del vídeo nativo (VIVOTEK inc., 2016).

Elementos de iViewer identificados de interés para la investigación:

- Permite guardar la configuración de la vista preferida a través de la opción *"Guardar Diseño"*.
- Soporta el cambio de cámaras en tiempo real, seleccionando una nueva fuente de vídeo para reemplazar la fuente actual.
- Permite la captura de imágenes de vídeo a través del comando "Instantáneas".

Elementos de iViewer que generan limitantes para la investigación:

- Visualiza el flujo de cámaras que gestiona los sistemas *VAST* y *ST7501*, por tanto, no se puede integrar con el sistema de Video Vigilancia SURIA 3.0.
- Solamente soporta cámaras de red *VIVOTEK* (series 7000, 8000 y 9000), *VIVOTEK NVR* (serie ND y NR) y *VIVOTEK VAST*.
- No se integra con el sistema de Video Vigilancia SURIA 3.0 pues es de código cerrado.

AXIS Camera Station 5

Aplicación de visualización para el sistema de vigilancia y seguridad *AXIS* es una solución rica en características para una vigilancia eficaz de instalaciones de tamaño pequeño y mediano, ideal para 4 a 50 cámaras. Permite la visualización en vivo con perfiles de transmisión de vídeo seleccionables y el acceso móvil a múltiples sistemas (Axis Communications, 2016).

Elementos de AXIS identificados de interés para la investigación:

- Instantáneas de vistas en vivo.
- La aplicación es ajustable a la necesidad del usuario.

Elementos de AXIS que generan limitantes para la investigación:

- No se integra con el sistema de Video Vigilancia SURIA 3.0 pues es de código cerrado.

Conclusión de las soluciones existentes:

Luego de una profunda investigación de las soluciones existentes se llega a las siguientes conclusiones:

- Ninguno se integra con los servicios que publica el Servidor del proyecto Video Vigilancia SURIA3.0.
- Solamente funcionan bajo cámaras fabricadas por sus compañías.
- No son de código abierto por ende no se pueden cambiar los servicios que consumen ni la forma de comunicarse.

Por lo tanto, se propone el desarrollo de la aplicación *SuriaVisor*, teniéndose en cuenta los elementos identificados de interés y corregir las negativas de las aplicaciones antes investigadas.

1.6 Caracterización de las tecnologías, herramientas, y metodología

Desarrollar un software implica que sea necesario definir las tecnologías, herramientas y metodología de desarrollo que se van a emplear para lograr el objetivo planteado. A continuación, se exponen las herramientas, tecnologías, técnicas y metodología de desarrollo a utilizar, las cuales ya han sido seleccionada por el arquitecto del proyecto de Video Vigilancia SURIA 3.0.

1.6.1 Tecnologías

A continuación, se describen las tecnologías seleccionadas que inciden en la visualización de cámaras desde dispositivos móviles con sistema operativo Android, mostrando una caracterización y análisis de las mismas.

Sistema Operativo Android

Google adquirió la compañía Android Inc. y desarrolló la plataforma Android. Posteriormente en acuerdo con el consorcio *Open Handset Alliance* decide promocionar los estándares de código abierto para dispositivos móviles. Como característica distintiva del sistema operativo Android se tiene que su núcleo está basado en el *Kernel* de Linux (Read the Docs, 2012).

Entre las características notables se tiene que la plataforma es adaptable a pantallas más grandes, VGA[8], biblioteca de gráficos *2D* y *3D* basada en las especificaciones de *OpenGL*; utiliza *SQLite* para el almacenamiento de datos; emplea SMS[9] y MMS[10] como vías de mensajería; soporta múltiples tecnologías de conectividad: datos móviles (*GSM/EDGE, IDEN, CDMA, EVDO, UMTS, LTE*) *Bluetooth, Wi-Fi y WiMAX*; maneja varios formatos multimedia: *AMR, MP3, MIDI, WAV, JPEG, PNG, GIF, BMP*, entre otros; incluye además soporte para hardware adicional: cámaras de fotos, de video, pantallas táctiles, GPS, acelerómetros, giroscopios, magnetómetros, termómetros, sensores de proximidad y de presión (Ferras, 2012).

[8] VGA: Adaptador Gráfico de Video
[9] SMS: Servicio de Mensajes Simples
[10] MMS: Servicio de Mensajería Multimedia

Componentes básicos de una aplicación Android

Hay tres tipos de componentes básicos en una aplicación Android. Cada uno de ellos tiene un propósito y un ciclo de vida distinto que define cómo se crea y se destruye el componente (Pulido, 2013).

Activity: Una actividad permite crear componentes que interaccionan con el usuario. Cada actividad tiene asociada una vista. La vista constituye el conjunto de elementos gráficos que se presentan al usuario para que interaccione con el sistema. Por ejemplo, cajas de edición, etiquetas o botones, entre otros. Para crear actividades personalizadas es necesario definir una clase que herede de la clase *Activity*.

Intent: Una intención o *intent* permite el envío de mensajes entre componentes. El paso de mensajes a través de *Intent* puede considerarse una facilidad para asociar (*binding*) componentes de forma tardía y en tiempo de ejecución en la misma aplicación o entre diferentes aplicaciones.

Service: Permite llevar a cabo operaciones en segundo plano o *background* y, por tanto, no proporcionan al usuario una interfaz de usuario.

Simple Object Access Protocol (SOAP)

El término *"servicios web SOAP"* designa una tecnología que permite que las aplicaciones se comuniquen en una forma que no depende de la plataforma ni del lenguaje de programación. Un servicio web *SOAP* es una interfaz de software que describe un conjunto de operaciones a las cuales se puede acceder por la red a través de mensajería *XML* estandarizada. Usa protocolos basados en el lenguaje *XML* con el objetivo de describir una operación para ejecutar sobre los datos o para intercambiar con otro servicio web. Los servicios web *SOAP*, pueden describir cualquier tipo de datos de manera independiente de la plataforma para el intercambio entre sistemas, lo que permite la creación de aplicaciones de bajo acoplamiento. Además, pueden funcionar a un nivel más abstracto que permita reevaluar, modificar o manejar tipos de datos dinámicamente mediante solicitud. Por tanto, en términos técnicos, este tipo de servicios puede manejar datos con mucha más facilidad y permite una comunicación más libre entre los sistemas (IBM, 2016). El *Framework* para servicios web SOAP se divide en tres áreas: protocolos de comunicación, descripción de servicios y descubrimiento de servicios (Curbera, et al., 2002).

1.6.2 Metodología de desarrollo de software

Teniendo en cuenta las necesidades para la solución del problema investigativo, la línea de desarrollo del proyecto SURIA y la sugerencia por parte del cliente, se utiliza la metodología de desarrollo Agile Unified Process (AUP) en su variante UCI. Esta describe de una manera fácil de entender la forma de desarrollar aplicaciones de software de negocio usando técnicas ágiles y conceptos que aún se mantienen válidos en RUP[11].

1.6.3 Lenguaje de modelado

El Lenguaje Unificado de Modelado (por sus siglas en inglés UML), permite visualizar, especificar, construir y documentar los artefactos de un sistema que involucra el desarrollo de un software. Un modelo UML describe lo que supuestamente hará un sistema, no cómo implementarlo. Permite la modelación de sistemas con tecnología orientada a objetos. Los diagramas son entes importantes de UML, cuya finalidad es representar diversas perspectivas de un sistema, a las cuales se les conoce como modelo. El modelo gráfico de UML tiene un vocabulario en el que se identifican: elementos, relaciones y diagramas (RUMBAUGH, et al., 2000). Por tanto, se hace uso de este lenguaje de modelado porque la metodología de desarrollo seleccionada recomienda usarla, además por la amplia documentación que existe.

Herramienta CASE Visual Paradigm 8.0

Visual Paradigm es una herramienta CASE[12] que brinda un entorno para la realización de diagramas con UML y BPMN[13]. Soporta todo el ciclo de desarrollo: análisis y diseño orientados a objetos, construcción, pruebas y despliegue. Facilita mejor entendimiento para el desarrollo entre los miembros del equipo ya que establece un lenguaje estándar, común para todos (Visual Paradigm, 2016). Se decide utilizar la herramienta por la facilidad que brinda para el trabajo colaborativo en equipos de desarrollo, también por su robustez y el uso del lenguaje UML.

[11] RUP: Proceso Racional Unificado
[12] CASE: Ingeniería de Software Asistida por Computadora
[13] BPMN: Modelo y Notación de Procesos de Negocio

1.6.4 Lenguaje de programación

Un lenguaje de programación puede definirse como un idioma artificial diseñado para que sea fácilmente entendible por un humano e interpretable por una máquina. Consta de una serie de reglas y de un conjunto de órdenes o instrucciones. Cada una de estas instrucciones realiza una tarea determinada (Jimenez Marin, et al., 2016).

Java

Java es un lenguaje de programación orientado a objetos ideado por la *Sun Microsystems* en 1995, supone un significativo avance en el desarrollo de software, es un lenguaje simple, distribuido, interpretado, robusto, de arquitectura neutral, portable y dinámico (SCHILDT, 2001).

Se selecciona este lenguaje por ser una de las grandes fortalezas que tiene la plataforma Android, ya que Java es muy rico en bibliotecas de software. A pesar de que existen otros lenguajes de programación para desarrollar aplicaciones Android Google lo propone como lenguaje estándar para el desarrollo de sus aplicaciones nativas. Además, existe una gran cantidad de bibliografía para apoyarse durante la fase de implementación.

1.6.5 Herramientas de desarrollo

IDE Android Studio 2.3

Android Studio es un entorno de desarrollo integrado (por sus siglas en inglés IDE) para la plataforma Android, basado en la suite de *JetBrains*. Este IDE posee características que facilitan el desarrollo de aplicaciones, algunas de ellas son: es capaz de realizar completamiento de código avanzado, refactorizar y analizar código; es posible crear aplicaciones para cualquier dispositivo con sistema operativo Android ya sean teléfonos inteligentes, tabletas, *Android Wear* o *Android TV*, también permite probarlas en emuladores virtuales con cualquier configuración de los tipos de dispositivos antes mencionados (Android Studio, 2016).

Gradle 2.14.1

Android Studio usa *Gradle*, un paquete de herramientas de compilación avanzadas, para automatizar y administrar el proceso de compilación, y al mismo tiempo permite definir configuraciones de compilación personalizadas y flexibles. El complemento de Android para *Gradle* funciona con el paquete de

herramientas de compilación para proporcionar procesos y ajustes configurables específicos para la compilación y prueba de aplicaciones de Android (Google, 2017).

Java Development Kit (JDK) 8u77

Es un software que provee herramientas de desarrollo para la creación de programas en Java. El JDK consta de una serie de aplicaciones y componentes, para realizar cada una de las tareas de las que es capaz de encargarse (Oracle Corporation, 2016).

Software Development Kit (SDK) 25.3.1

El SDK es necesario para desarrollar aplicaciones y ejecutar un emulador del sistema Android de la versión que sea. Todas las aplicaciones Android se desarrollan en lenguaje Java con este kit. Cada vez que aparece una nueva versión de Android, Google libera el código fuente y publica el SDK con la nueva versión de Android para que los desarrolladores puedan adaptarse a los cambios y funcionalidades del nuevo sistema (Desarrollo de aplicaciones sobre Android, 2013).

LibVLC 3.0

LibVLC es la API[14] del reproductor multimedia VLC[15], este se utiliza para integrar VLC en otras aplicaciones o *frameworks* (VideoLAN, 2017). No requiere de *codec* del sistema para su funcionamiento ya que es totalmente independiente en ese aspecto.

Gson 2.2.4

Gson es una biblioteca Java que se puede utilizar para convertir objetos Java en su representación *JSON*. También se puede utilizar para convertir una cadena *JSON* en un objeto Java equivalente, puede trabajar con objetos Java arbitrarios incluyendo objetos preexistentes que no tiene código fuente. Este es un proyecto de código abierto (Google Sites, 2016).

java-websocket 1.3.1

WebSocket es un protocolo de aplicación que proporciona comunicaciones *full-duplex* entre dos pares a través del protocolo TCP[16]. En el modelo de solicitud-respuesta tradicional utilizado en *HTTP*, el cliente

[14] API: Application Programming Interface
[15] VLC: Reproductor multimedia de VideoLAN.
[16] TCP: Transmission Control Protocol

solicita recursos y el servidor proporciona respuestas, el intercambio siempre es iniciado por el cliente. El servidor no puede enviar ningún dato sin que el cliente lo solicite primero. Este modelo funcionó bien para la *World Wide Web* cuando los clientes hicieron peticiones ocasionales de documentos que cambiaron con poca frecuencia, pero las limitaciones de este enfoque son cada vez más relevantes, ya que el contenido cambia rápidamente y los usuarios esperan una experiencia más interactiva en la Web. El protocolo *WebSocket* soluciona estas limitaciones proporcionando un canal de comunicación dúplex completo entre el cliente y el servidor. Combinado con otras tecnologías de cliente, como JavaScript y HTML5, *WebSocket* permite a las aplicaciones web ofrecer una experiencia de usuario más rica (ORACLE, 2014).

FFmpegMediaMetadataRetriever 1.0.14

FFmpegMediaMetadataRetriever es una reimplementación de la clase *MediaMetadataRetriever* de Android. Esta proporciona una interfaz unificada para recuperar los metadatos de la entrada de un archivo de medias y utiliza *FFmpeg* como *backend*. Permitiendo así la toma de una a varias instantáneas del flujo de video de una cámara.

1.7 Conclusiones parciales

Los elementos que se muestran en este capítulo permiten la culminación de varias tareas trazadas en el inicio del trabajo investigativo, específicamente las tareas 1, 2 y 3, lo cual se manifiesta en:

- Con la revisión de los conceptos asociados a la investigación y la adaptación de estos con la problemática de la investigación se pudo fomentar una fuerte base de conocimientos para comprender el problema.
- El estudio de las soluciones existentes demuestra que no pueden ser utilizadas para dar solución al problema planteado, ya que sus funcionalidades responden al negocio para el que están diseñadas, además no son compatibles con los servicios que brinda SURIA. A pesar de esto se tomaron como referencia para el diseño y la presentación de la información.
- La caracterización de las principales herramientas, tecnologías, lenguajes y metodologías escogidas para el desarrollo de la aplicación, fueron determinadas por las características de estas, lo cual posibilitará la obtención de una solución bien documentada, ofreciendo facilidad para el desarrollo de futuras versiones de la aplicación, además de seguir los estándares utilizados internacionalmente.

CAPÍTULO II: LEVANTAMIENTO DE REQUISITOS Y PROPUESTA DE SOLUCIÓN

2.1 Introducción

En el presente capítulo se describe la solución propuesta y el modelo de dominio, se muestran los conceptos principales y su relación. Se enuncian los requisitos para determinar el dominio de la aplicación, el desempeño, así como las restricciones que el mismo debe poseer, la especificación de los requisitos para determinar lo que el sistema debe realizar, además de las descripciones de los casos de uso para mayor comprensión de su funcionamiento. Se describe la arquitectura del sistema y se argumentan los patrones de diseño a utilizar.

2.2 Modelo de dominio

Un modelo de dominio es una definición de abstracciones del dominio como políticas, procedimientos, objetos, relaciones y eventos. Sirve de base de conocimiento sobre alguna área del problema (Sommerville, 2011).

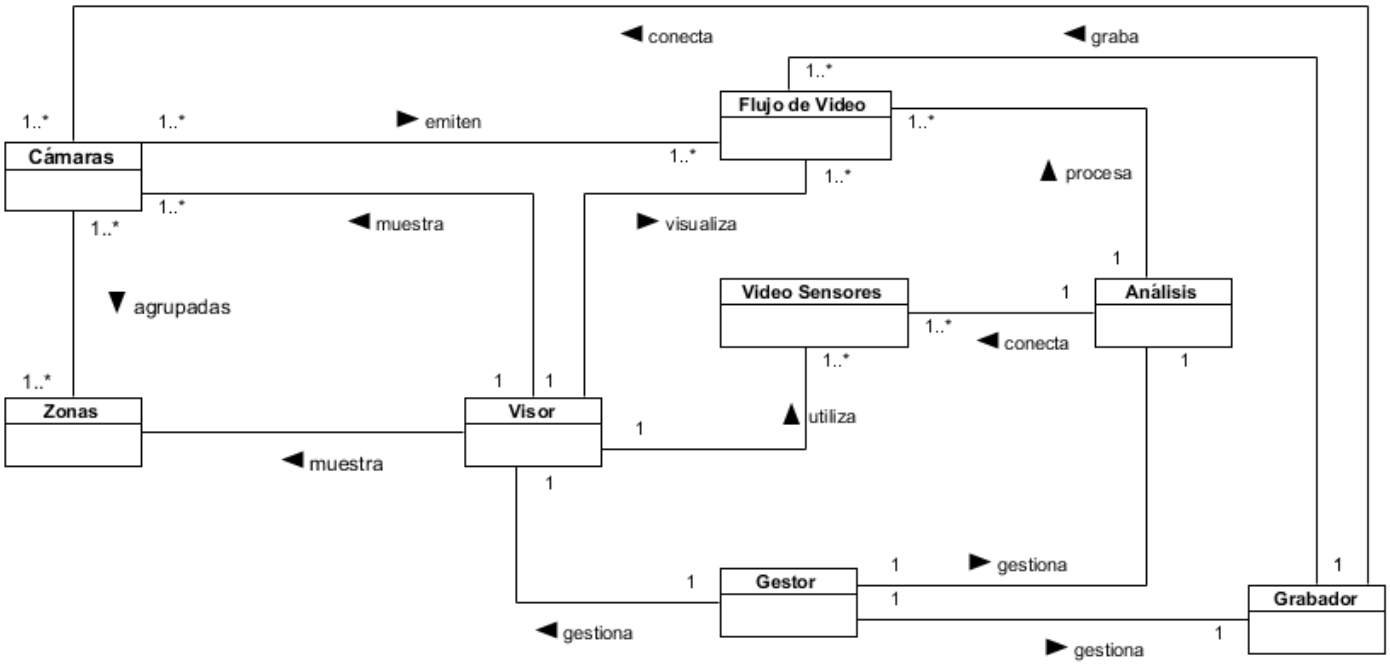


Fig. 1: Diagrama de clases del modelo de dominio

2.3 Descripción de los conceptos del modelo de dominio

Visor: Utiliza los video sensores que están implementados y contenidos en el módulo de análisis, muestra las zonas y cámaras en forma de árbol y visualiza el flujo que emiten estas cámaras.

Video Sensores: Algoritmo que se encarga de detectar algún tipo de situación después de hacer un análisis o mientras que se hace un análisis en el flujo de una cámara. Estos están contenidos en el módulo de análisis.

Zonas: Contiene las cámaras agrupadas por zona.

Cámaras: Contiene la información de las cámaras, como *ip, user, password, model, spefic info, protocol* entre otros. Estas emiten flujo de video.

Flujo de Video: Flujo de video emitido por las cámaras, procesado por el módulo de análisis, visualizado por el visor y grabado por el grabador.

Análisis: Contiene video sensores, procesa el flujo de videos emitido por las cámaras y es gestionado por el gestor.

Gestor: Es el intermediario entre el visor y todas las funcionalidades de la aplicación, es el que se encarga de mandar al módulo de análisis a activar los sensores, al módulo de grabación a grabar y gestiona el visor.

Grabador: Inicia los flujos de videos de las cámaras y los graba.

2.4 Requisitos del Software

La especificación de requisitos es el proceso de desarrollo de software donde se analizan las necesidades exactas de los usuarios de un sistema y traducir estas a precisas funciones y acciones que subsecuentemente serán usadas en el desarrollo del sistema. Un correcto levantamiento de requisitos permite posteriormente la construcción de un software de calidad (Pressman, 2010). Generalmente estos requisitos pueden clasificarse en funcionales y no funcionales.

2.4.1 Requisitos funcionales

Para saber qué debe hacer el sistema y su funcionamiento es necesario conocer los requisitos funcionales (RF), de la manera en que este debe reaccionar a entradas particulares y como debe comportarse en

situaciones particulares, no es más que una capacidad o condición que el sistema debe cumplir (Pressman, 2010). Se utilizó el método entrevista no estructurada a través de la tormenta de ideas para la recopilación de los requisitos funcionales. A continuación, se muestran los requisitos funcionales y una breve descripción de los mismos:

RF 1 Autenticar usuario en el visor: El sistema debe permitir que solo usuarios autorizados tengan acceso a la aplicación. Estos usuarios solo tendrán acceso a las funcionalidades propias de su rol.

RF 2 Cerrar sesión en el visor: El sistema debe permitir cerrar la sesión de un usuario autenticado y dirigirlo hacia la vista del *login*.

RF 3 Salir del visor: El sistema debe permitir al usuario salir del sistema.

RF 4 Notificar desconexión del visor con el servidor: En caso de que el Visor se desconecte del servidor, el sistema debe ser capaz de notificar la desconexión a los usuarios autenticados.

RF 5 Mostrar árbol de cámaras y zonas adicionadas en el sistema: El sistema debe mostrar las cámaras y zonas a las cuales tenga acceso el usuario autenticado. Los elementos del árbol deben aparecer agrupados en orden jerárquico de zona y cámara.

RF 6 Establecer conexión con el servidor: El sistema debe permitir insertar los datos de configuración del servidor.

RF 7 Visualizar flujo de video de una cámara en el visor: El sistema debe ser capaz de mostrar el flujo de video de una cámara. El usuario deberá indicar que cámara será visualizada.

RF 8 Adicionar vista en el visor: El sistema debe permitirle al usuario adicionar una vista personalizada. Para ello deberá seleccionar el modelo de vista (la cantidad de áreas de visualización que tendrá la nueva vista) y la cámara que se reproducirá por cada área de visualización.

RF 9 Eliminar vista en el visor: El sistema debe permitirle al usuario eliminar una vista existente.

RF 10 Editar vista en el visor: El sistema debe permitirle al usuario editar una vista existente.

RF 11 Mostrar video en pantalla completa en el visor: El sistema debe permitirle al usuario, establecer el modo pantalla completa en una de las áreas de visualización que contenga un flujo de video de una cámara.

RF 12 Mostrar video en pantalla normal en el visor: El sistema debe permitirle al usuario volver a visualizar en el modo pantalla normal. La pantalla previamente debe estar en pantalla completa.

RF 13 Desacoplar área de visualización en el visor: El sistema debe permitir al usuario desacoplar a un área de visualización perteneciente a una vista. Dicha área de visualización debe poseer un flujo de video de una cámara.

RF 14 Acoplar área de visualización en el visor: El sistema debe permitir al usuario acoplar un área de visualización que haya sido desacoplada previamente.

RF 15 Cerrar área de visualización en el visor: El sistema debe permitir al usuario cerrar el flujo de video que no desee seguir visualizando. El área debe estar siendo visualizada en el momento.

RF 16 Ver modo mudo de un visualizador en el visor: El sistema debe permitir al usuario establecer el modo mudo en área de visualización de una vista.

RF 17 Salir del modo mudo de un visualizador en el visor: El sistema debe permitir al usuario salir del modo mudo en un área de reproducción donde previamente se haya establecido el modo mudo. El sistema por defecto carga el área de visualización en modo mudo.

RF 18 Visualizar sensores disponibles en el visor: El sistema debe ser capaz de mostrar los sensores disponibles para cada flujo de video en dependencia de la cámara.

RF 19 Activar video sensor en el visor: El sistema debe permitir al usuario activar un video sensor de los que se encuentren disponibles en dependencia de la cámara.

RF 20 Desactivar video sensor en el visor: El sistema debe permitir al usuario desactivar un video sensor que haya sido activado previamente.

RF 21 Iniciar grabaciones manuales en el visor: El sistema debe permitir al usuario iniciar la grabación de un flujo de video obtenido desde una cámara, que se encuentre visualizado en un área de reproducción de una vista.

RF 22 Detener grabaciones manuales en el visor: El sistema debe permitir al usuario detener una grabación que se haya iniciado previamente.

RF 23 Buscar zona y cámara en el visor: El sistema debe permitir al usuario buscar zona y cámara en el Visor.

RF 24 Actualizar automáticamente el árbol de zonas y cámaras en el visor: El sistema debe ser capaz de actualizar automáticamente el árbol de zonas y cámaras. El árbol se actualiza en caso de existir algún cambio en el servidor.

RF 25 Generar alarmas visuales por eventos asociados en el visor: El sistema debe permitir generar alarmas visuales asociada a eventos que puedan ocurrir. Los eventos pueden ser: (Detección de movimiento, detección de fuego, detección de humo, etc.).

RF 26 Silenciar alarma en el visor: El sistema debe permitir al usuario silenciar una alarma.

RF 27 Salir del modo silenciar alarmas en el visor: El sistema debe permitir al usuario salir del modo silenciar alarma en una alarma que haya sido silenciada previamente.

RF 28 Limpiar alarmas en el visor: El sistema debe permitir eliminar las alarmas existentes para el flujo de video seleccionado.

RF 29 Mostrar cambios de estado de la cámara en el visor: El sistema debe ser capaz de mostrar el estado en el que se encuentren las cámaras (activo, inactivo, bloqueada y grabando).

RF 30 Guardar instantánea en el visor: El sistema debe permitir al usuario hacer una instantánea del video que se está reproduciendo en un área de visualización. Esta instantánea se guardará por defecto en el servidor.

RF 31 Guardar ráfaga de instantánea en el visor: El sistema debe permitir al usuario hacer una ráfaga de instantáneas del video que se está reproduciendo en un área de visualización. Estas instantáneas se guardarán por defecto en el servidor.

RF 32 Incorporar marcador a un video en tiempo real en el visor: El sistema deber permitir al usuario incorporar un marcador al video que se está reproduciendo en un área de visualización, en tiempo real. El video debe estar grabando en el instante que se desea incorporar marcador.

RF 33 Configurar localmente las instantáneas en el visor: El sistema debe permitirle al usuario, configurar localmente las instantáneas, especificando la ruta de almacenamiento que desee.

RF 34 Mostrar notificaciones en el visor: El sistema debe permitir mostrar al usuario las notificaciones asociadas a diferentes acciones realizadas. Las acciones pueden estar asociadas a los eventos, alarmas, reglas, entre otras.

RF 35 Salir de la interfaz principal del visor: El sistema debe permitir al usuario autenticado salir de la interfaz principal.

RF 36 Mostrar listado de las vistas en el visor: El sistema debe permitir mostrar listado de las vistas existentes en el visor.

2.4.2 Requisitos no funcionales

Los requisitos no funcionales se refieren a funciones específicas que proporciona el sistema. Son restricciones de los servicios o funciones ofrecidas por el sistema (fiabilidad, tiempo de respuestas, capacidad de almacenamiento). Generalmente se aplican al sistema en su totalidad. En muchos casos los requerimientos no funcionales son fundamentales en el éxito del producto. Están vinculados a requisitos funcionales, es decir, una vez que se conozca lo que el sistema debe hacer se puede determinar cómo ha de comportarse, qué cualidades debe tener o cuán rápido o grande debe ser (Pressman, 2010). A continuación, se muestran los requisitos no funcionales y la descripción de los mismos:

Requisitos de usabilidad:

- Las interfaces gráficas implementadas por el sistema deben concebirse con un ambiente sencillo y de navegación fácil para el usuario, debido a que la aplicación podrá ser usada por usuarios con conocimientos básicos de dispositivos móviles.

Requisitos de software:

- Se debe disponer en el dispositivo móvil una versión 4.0 o superior del sistema operativo Android.

Requisitos mínimos de hardware:

- El dispositivo debe soportar tecnología *Wi-Fi 802.11* en sus variantes ac/b/g/n.

- Se requiere un microprocesador con una velocidad de 1000 MHz[17] con tecnología *Dual Core* o superior.
- Se requiere de 512 Mb de memoria RAM[18].
- Dispositivo con una resolución igual o superior a 480 x 800 píxeles, para una correcta visualización de la aplicación.

Requisitos de Seguridad:

- La información que se gestiona en la aplicación estará protegida del acceso no autorizado. Ya que cualquier usuario no tendrá acceso a la aplicación pues necesariamente el usuario para acceder a la información tiene que autenticarse.
- El protocolo que se usa para la transferencia de datos es WSS (Web Socket Seguro) con SSL[19].

Requisitos de restricciones de diseño:

- El diseño de la aplicación estará sujeto a las especificaciones de la dirección de producción de la Universidad de las Ciencias Informáticas, específicamente a la marca XILEMA.

2.5 Modelo del Sistema

Según (RUMBAUGH, et al., 2000) *"… El Modelo de Casos de Uso del Sistema colabora con el cliente, los usuarios y los desarrolladores en establecer un lenguaje común para la futura implementación de un sistema software..."* Cada tipo de usuario del sistema a desarrollar se representa mediante un actor. Todos los actores y casos de uso conforman el Modelo de Casos de Uso del Sistema (CUS).

2.5.1 Casos de uso

Todo software ofrece a su entorno un conjunto de servicios, un caso de uso es la utilización de estos servicios por alguien o algo fuera del sistema. Los casos de uso definen un escenario que identifica una línea de utilización para el sistema, facilitando una descripción de cómo se utilizará el mismo (Pressman, 2010). A continuación, se listan los casos de uso del sistema:

[17] MHz: Megahercio
[18] RAM: Memoria de Acceso Aleatorio
[19] SSL: Secure Sockets Layer

CU 1: Autenticar usuario en el visor.

CU 2: Cerrar sesión en el visor.

CU 3: Salir del visor.

CU 4: Notificar desconexión del visor.

CU 5: Mostrar árbol de cámaras y zonas adicionadas en el sistema según los permisos del usuario autenticado en el visor.

CU 6: Visualizar flujo de video de una cámara en el visor.

CU 7: Gestionar vista en el visor.

CU 8: Realizar operaciones con el visualizador.

CU 9: Realizar peticiones de grabación manual en el visor.

CU 10: Buscar zona y cámara en el visor.

CU 11: Actualizar automáticamente el árbol de zonas y cámaras en el visor.

CU 12: Realizar operaciones con las alarmas en el visor.

CU 13: Mostrar notificaciones en el visor.

CU 14: Configurar localmente las instantáneas y las vistas en el visor.

CU 15: Establecer datos de configuración.

CU 16: Realizar operaciones con los video sensores.

CU 17: Realizar operaciones a un video en tiempo real en el visor.

CU 18: Salir de la interfaz principal del visor.

Diagrama de Casos de Uso del Sistema

A continuación, en la Figura 2 se muestra el diagrama de casos de uso del sistema para una mejor comprensión de las funcionalidades, evidenciándose la pertenencia de cada caso de uso con el actor que lo ejecuta.

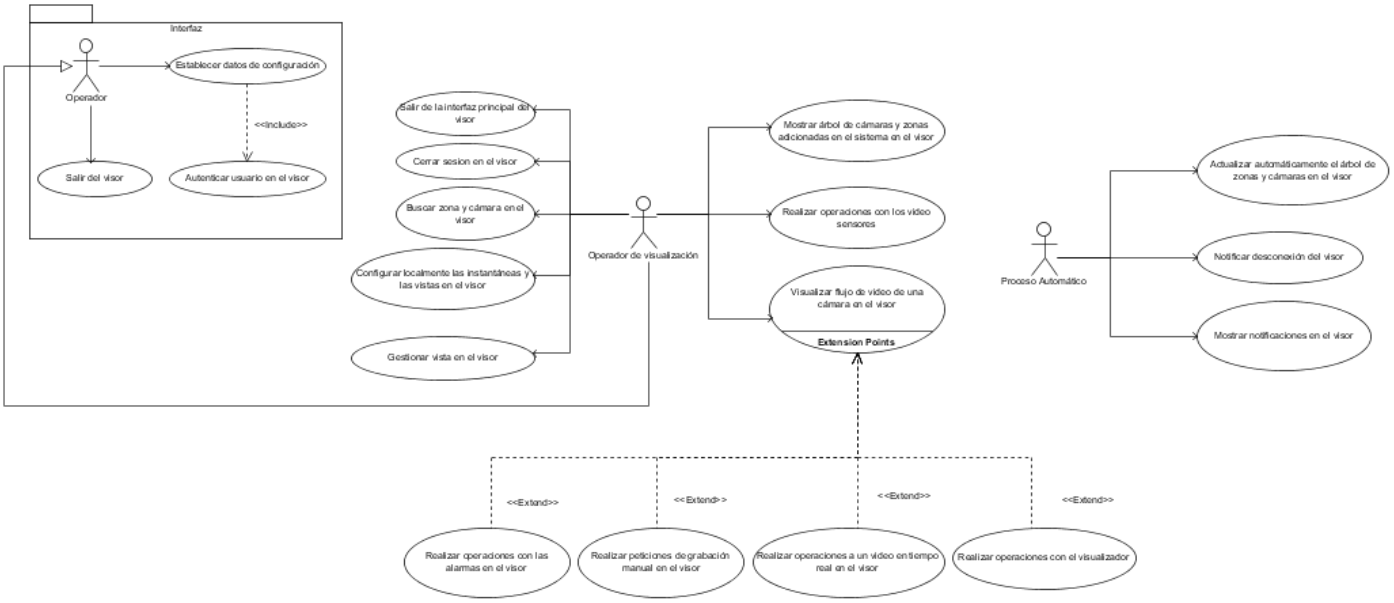


Fig. 2: Diagrama de Caso de Uso del Sistema

2.5.2 Descripción de los actores del sistema

Los actores del sistema se definen como los agentes externos al sistema que interactúan con este. En el diagrama de caso de uso del sistema se especifica con que servicios específicos interactúa cada actor.

Tabla 1: Descripción de los actores del sistema

Actor	Objetivos
Operador	Usuario con permisos para realizar todas las operaciones relacionadas con la visualización de las cámaras.
Proceso automático	Representa los procesos que se están ejecutando constantemente para chequear el estado de conexión con el Servidor de Administración.
Operador de visualización	Usuario con permisos para realizar todas las operaciones relacionadas con la visualización de las cámaras.

2.5.3 Descripción de los casos de uso del sistema

La descripción detallada de los casos de uso posibilita la posterior implementación y diseño de las pruebas de interfaz. A continuación, en la Tabla 2 se muestra la descripción del caso de uso Autenticar usuario en el visor, para ver las descripciones de los otros casos de uso ver Anexo no.1.

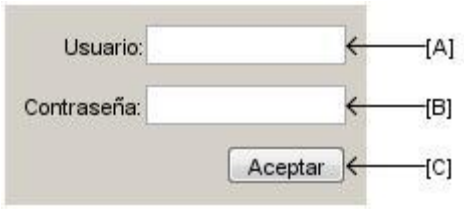
Tabla 2: Descripción del caso de uso Autenticar usuario en el visor

Objetivo	Registrar al usuario en el sistema, proporcionándole acceso a las funcionalidades a las que tenga permiso.
Actores	Operador: (Inicia) Autenticación de usuarios en el sistema.
Resumen	El CU inicia cuando se solicita acceder al sistema, termina cuando se habilita o deniega el acceso al usuario con los permisos asignados.
Complejidad	Baja
Prioridad	Media
Precondiciones	Debe existir conexión con el Servidor de Administración.
Postcondiciones	Usuario registrado en el sistema con acceso a las funcionalidades que tiene permiso.

Flujo de eventos		
Flujo básico Autenticar usuarios en el módulo Visor.		
	Actor	**Sistema**
1.	Introduce los siguientes datos: • Nombre de usuario (A) (Cadena de caracteres) • Contraseña (B) (Cadena de caracteres) *Nota: Las contraseñas no deben ser visibles en la interfaz, y deben cumplir con las restricciones de seguridad establecidas para este campo.*	

2.	Selecciona la opción Aceptar (C).	
3.		Envía los datos al Servidor de Administración para que los compruebe con la Base de Dato.
4.		Recibe notificación por parte del Servidor de Administrador.
5.		Si los datos son correctos realiza un certificado de seguridad, donde especifican los permisos del usuario autenticado.
6.		Muestra al usuario la interfaz del módulo con acceso a todas las funcionalidades a las que tiene permiso. Termina el CU.
Flujos alternos		
5a. Evento: Los datos del usuario son incorrectos.		
	Actor	**Sistema**
5.		El sistema muestra un mensaje de error informando al usuario que los datos insertados no son válidos. Termina el CU.

Relaciones	**CU incluidos**	No existe.
	CU extendidos	No existe.
Requisitos funcionales	RF 1	
Requisitos no funcionales	No procede	
Asuntos pendientes	No procede	
Prototipo elemental de interfaz gráfica de usuario		



2.6 Arquitectura del software

La arquitectura de un sistema consiste en la descripción de los subsistemas y componentes de un sistema informático y las relaciones que se establecen entre ellos (Pressman, 2010). La correcta selección y aplicación de una arquitectura en el diseño ingenieril de un proyecto es fundamental, pues permite organizar el desarrollo, promover la reutilización de código, facilitar el mantenimiento y modificaciones posteriores.

2.6.1 Estilo arquitectónico

Carlos Billy Reynoso en su libro Introducción a la Arquitectura de Software plantea*: "Un estilo es un concepto descriptivo que define una forma de articulación u organización arquitectónica."* Los estilos arquitectónicos indican los tipos de componentes y conectores utilizados, además provee una serie de patrones óptimos para el desarrollo.

Para el desarrollo de la solución propuesta se escoge el estilo arquitectónico ***Llamada y Retorno***, pues permite la realización de estructuras de programación escalables y fáciles de modificar, lo cual contribuye a la flexibilidad del sistema.

2.6.2 Patrón arquitectónico

Los patrones arquitectónicos proveen una descripción de los componentes, elementos y conectores; así como la interacción entre ellos. Además, exponen un conjunto de restricciones sobre cómo pueden ser usados. Un patrón arquitectónico organiza estructuralmente un sistema de software, definiendo subsistemas a partir de sus responsabilidades e interrelaciones (REYNOSO, 2004).

Para el desarrollo de la solución propuesta se escoge el patrón arquitectónico *N-Capas*, en específico *2-Capas*. Este patrón organiza jerárquicamente en capas, donde cada capa provee de información a la capa superior y se sirve de la inferior, permitiendo agrupar en dos capas los componentes de la aplicación (Ferras, 2012), las cuales se detallan a continuación:

- Capa de presentación: es la única que interactúa directamente con el usuario, permitiendo el intercambio de información con ellos. En esta capa se agrupan todas las vistas. Ejemplo de estas son: *LoginActivity y IPlayerActivity*.

- Capa de lógica: recibe las peticiones de la capa de presentación y le envía la respuesta tras el procesado de las mismas. Agrupa los componentes lógicos del flujo de trabajo. Ejemplo de estas son: *VisorController y CustomView.*

La principal ventaja de esta arquitectura, es que el desarrollo se puede llevar a cabo en varios niveles y, en caso de que sobrevenga algún cambio, sólo se ataca al nivel requerido sin tener que revisar entre código mezclado. Además, a cada nivel se le confía una misión simple, lo que permite el diseño de arquitecturas escalables.

2.6.3 Patrones de diseño

Los patrones de diseño elevan la calidad de los sistemas, además de brindar una solución probada y documentada a problemas de desarrollo de *software* que están sujetos a contextos similares y de representar una excelente manera de trabajar en base a la calidad (Fowler, 2007). Existen clasificaciones para estos patrones entre los que se encuentran los Patrones Generales de Software para Asignar Responsabilidades y los Patrones del Grupo de los Cuatro. A continuación, se presentan los patrones de diseño que son utilizados para el desarrollo de la solución.

Patrones Generales de Software para Asignar Responsabilidades

Los patrones generales de software para asignar responsabilidades (por sus siglas en inglés GRASP) permiten asignar responsabilidades a objetos de forma tal que estos conozcan el rol que les corresponde (Larman, 2003). Los patrones que se utilizan son:

Experto: Asigna una responsabilidad a la clase que cuenta con la información necesaria para cumplirla. Constituye un principio básico que frecuentemente se utiliza en el diseño orientado a objetos. Las ventajas que presenta este patrón son: la conservación del encapsulamiento, debido a que los objetos para hacer lo que se les pide se valen de su propia información, además de posibilitar que el sistema sea robusto y de fácil mantenimiento. Este patrón se evidencia en las clases: *Sensor y Zone.*

Creador: Asigna la responsabilidad a una clase de crear objetos, práctica muy frecuente en los sistemas orientados a objetos. Entre sus beneficios se encuentra el brindar soporte a un bajo acoplamiento lo que trae como consecuencia una menor dependencia respecto al mantenimiento y la reutilización en mayor medida. Este patrón se evidencia en las clases: *LoginActivity y RecyclerAlarmAdapter.*

Controlador: Asigna la responsabilidad a una clase de administrar un mensaje de los eventos del sistema. *"Un evento del sistema es un evento de alto nivel generado por un actor externo; es un evento de entrada externa. Se asocia a operaciones del sistema: las que emite en respuesta a los eventos del sistema"* (Larman, 2003). Este patrón se evidencia en las clases: *VisorController y MainActivity.*

Alta cohesión: Asigna las responsabilidades en función de mantener la complejidad dentro de los límites posibles. La cohesión es una medida que refleja cuán enfocadas y relacionadas se encuentran las responsabilidades en una clase. Las clases que presentan una alta cohesión se caracterizan por poseer responsabilidades estrechamente relacionadas que no hagan un enorme trabajo; sin embrago, una clase con baja cohesión realiza muchas cosas no afines a un trabajo excesivo. Entre los beneficios que presenta el patrón está la mejora de la facilidad y claridad para comprender el diseño. Este patrón se evidencia en la clase: *Server*.

Bajo acoplamiento: Asigna la responsabilidad a las clases garantizando que las mismas se encuentren lo más independiente posibles, lo que trae consigo una dependencia escasa y un aumento en la reutilización. Este patrón se evidencia en las clases: *Event y TreeElement*.

Patrones Grupo de los Cuatro (GOF)

Los patrones del Grupo de los Cuatro (por sus siglas en inglés GOF) se dividen en tres categorías: patrones de creación, patrones estructurales y patrones de comportamiento (Larman, 2003). A continuación, se describen los patrones que se evidencian en la solución:

Instancia única (del inglés **Singleton**): es un patrón de creación y garantiza que una clase posea una única instancia, a la vez que provee un punto de acceso global a ella. Permite por su diseño restringir la creación de objetos pertenecientes a una clase o el valor de un tipo a un único objeto. En la solución se empleó para la construcción de la clase *VisorController*.

Adaptador (del inglés **Adapter**): convierte la interfaz de una clase en otra distinta que es la que esperan los clientes. Permite que cooperen clases que de otra manera no podrían por tener interfaces incompatibles. Este patrón posibilitó el diseño e implementación de varias clases de la capa Presentación, específicamente las clases que se encuentran el paquete *Adapter*.

Observer: Este patrón define una relación de uno a muchos, entre un grupo de objetos. Cuando un objeto cambia su estado, todos sus dependientes son notificados y actualizados automáticamente. Este patrón se evidencia en las clases: *Server*.

2.6.4 Modelo de diseño

El modelo de diseño es un modelo de objetos que describe la realización física de los casos de uso centrándose en cómo los requisitos funcionales y no funcionales, junto con otras restricciones relacionadas con el entorno de implementación, tienen impacto en el sistema (Larman, 2003), sirve de abstracción para la implementación y como entrada fundamental de la misma.

2.6.5 Diagrama de clases del diseño

Un diagrama de clases sirve para visualizar las relaciones entre las clases que involucran el sistema, los cuales responden al cumplimiento de los requisitos funcionales. Los diagramas de clases del diseño son subproductos del modelo de diseño proporcionando una perspectiva estática que representa el diseño estructural del sistema, mostrando un conjunto de clases y sus atributos (Larman, 2003). A continuación, en la Figura 3 se muestra el diagrama de clases del diseño del sistema:

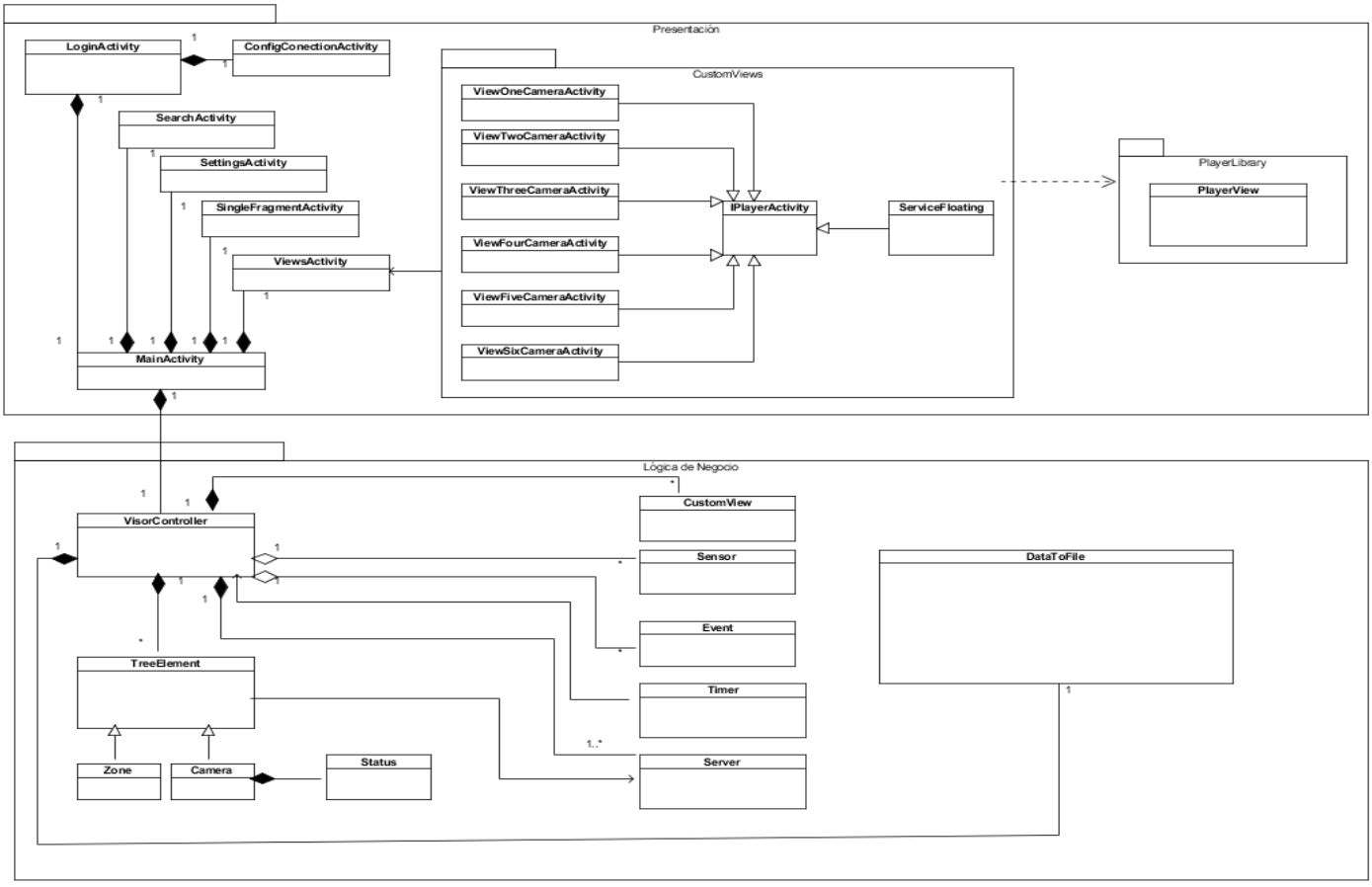


Fig. 3: Diagrama de Clases del Diseño

A continuación, se muestran las clases con sus atributos y métodos correspondientes al diagrama de clases del diseño del sistema:

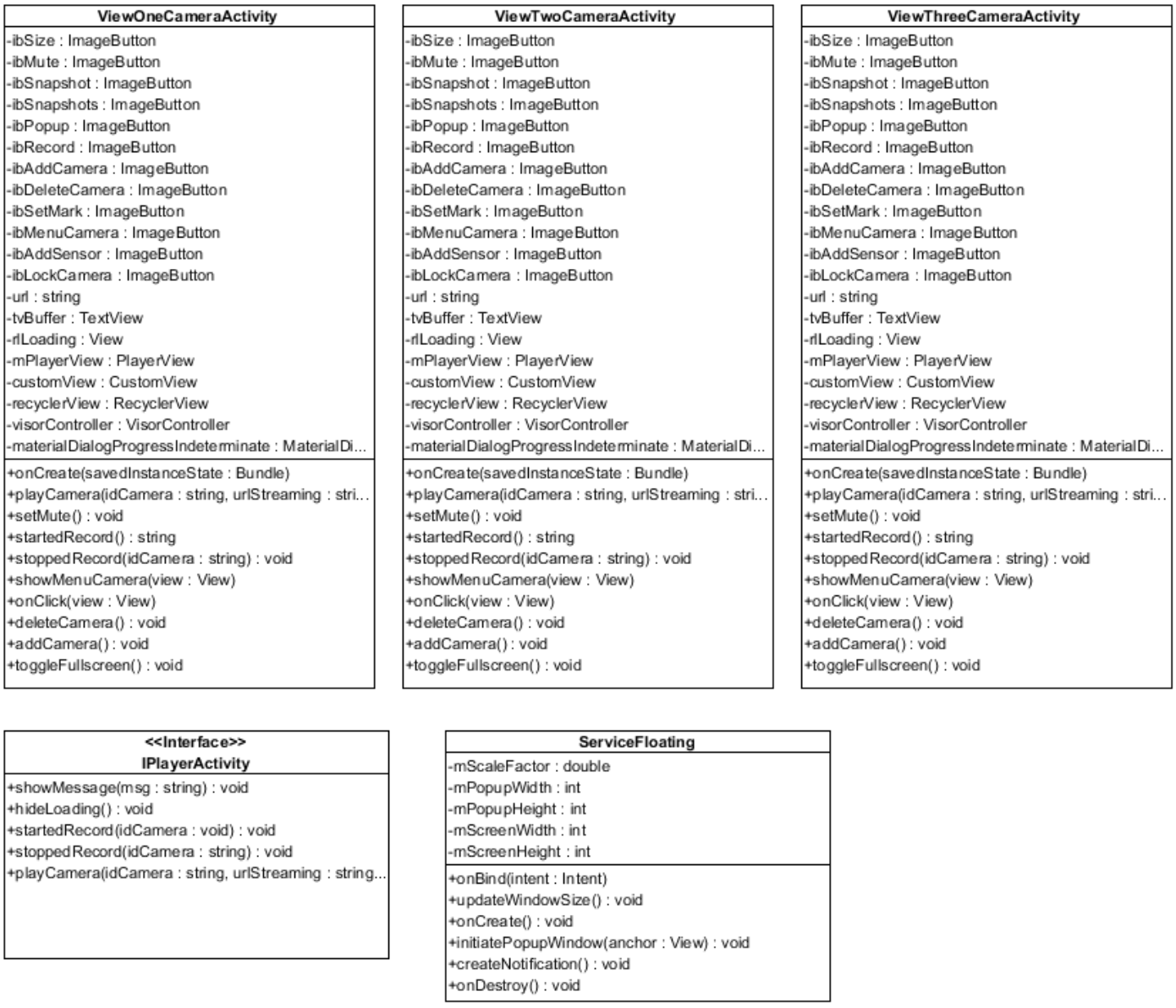


Fig. 4: Clases del Diagrama de Clases del Diseño del sistema: Capa Presentación

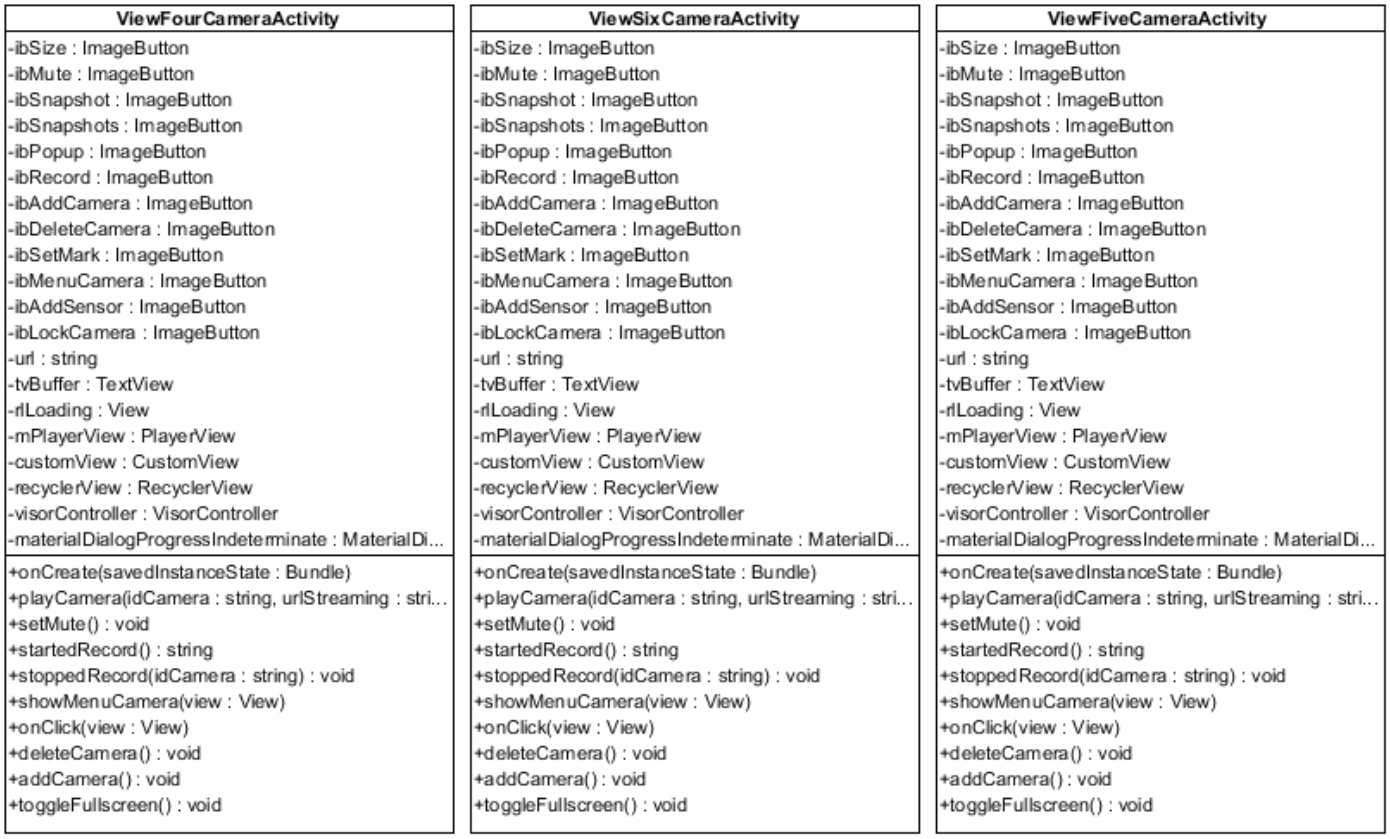


Fig. 5: Clases del Diagrama de Clases del Diseño del sistema: Capa Presentación

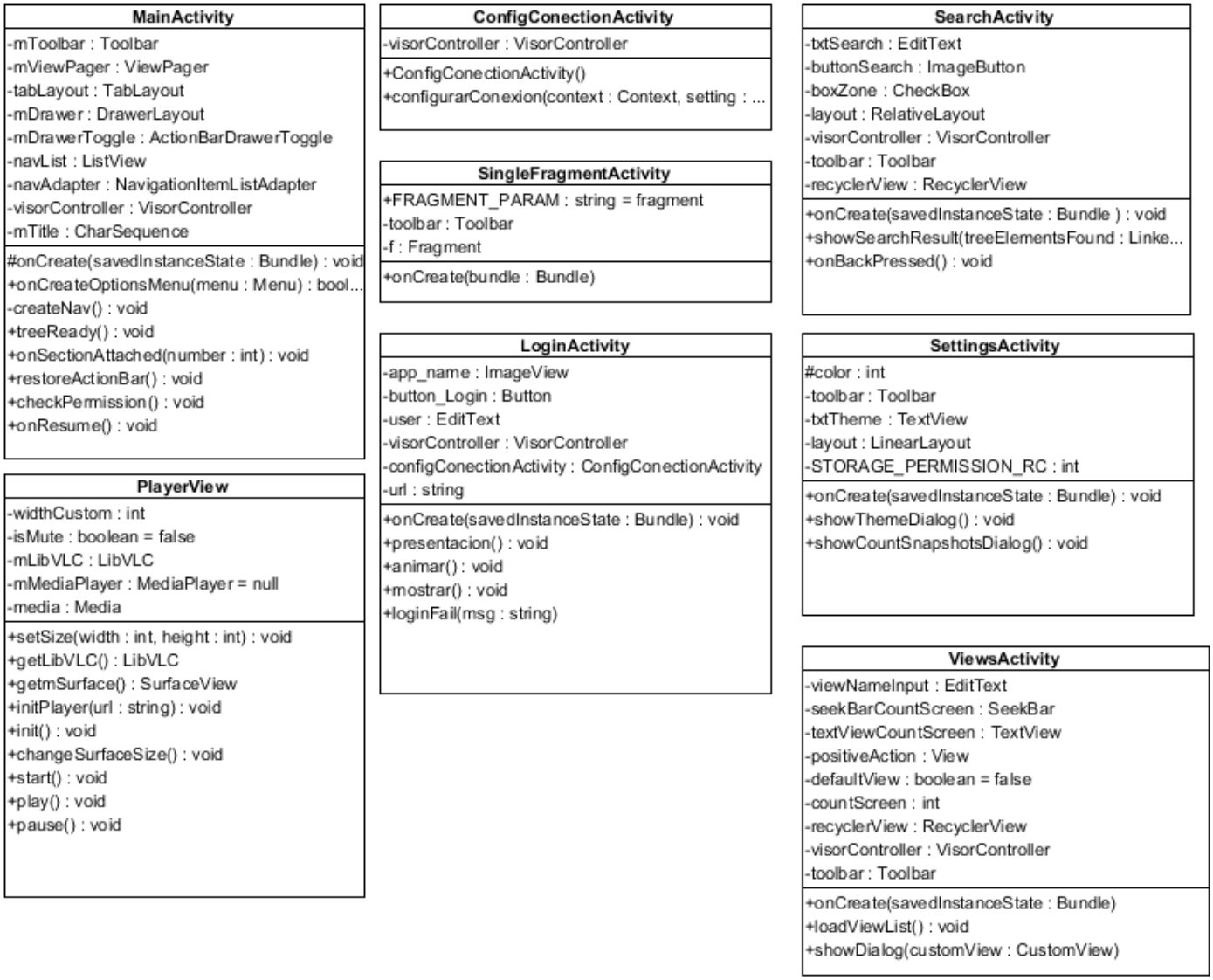


Fig. 6: Clases del Diagrama de Clases del Diseño del sistema: Capa Presentación

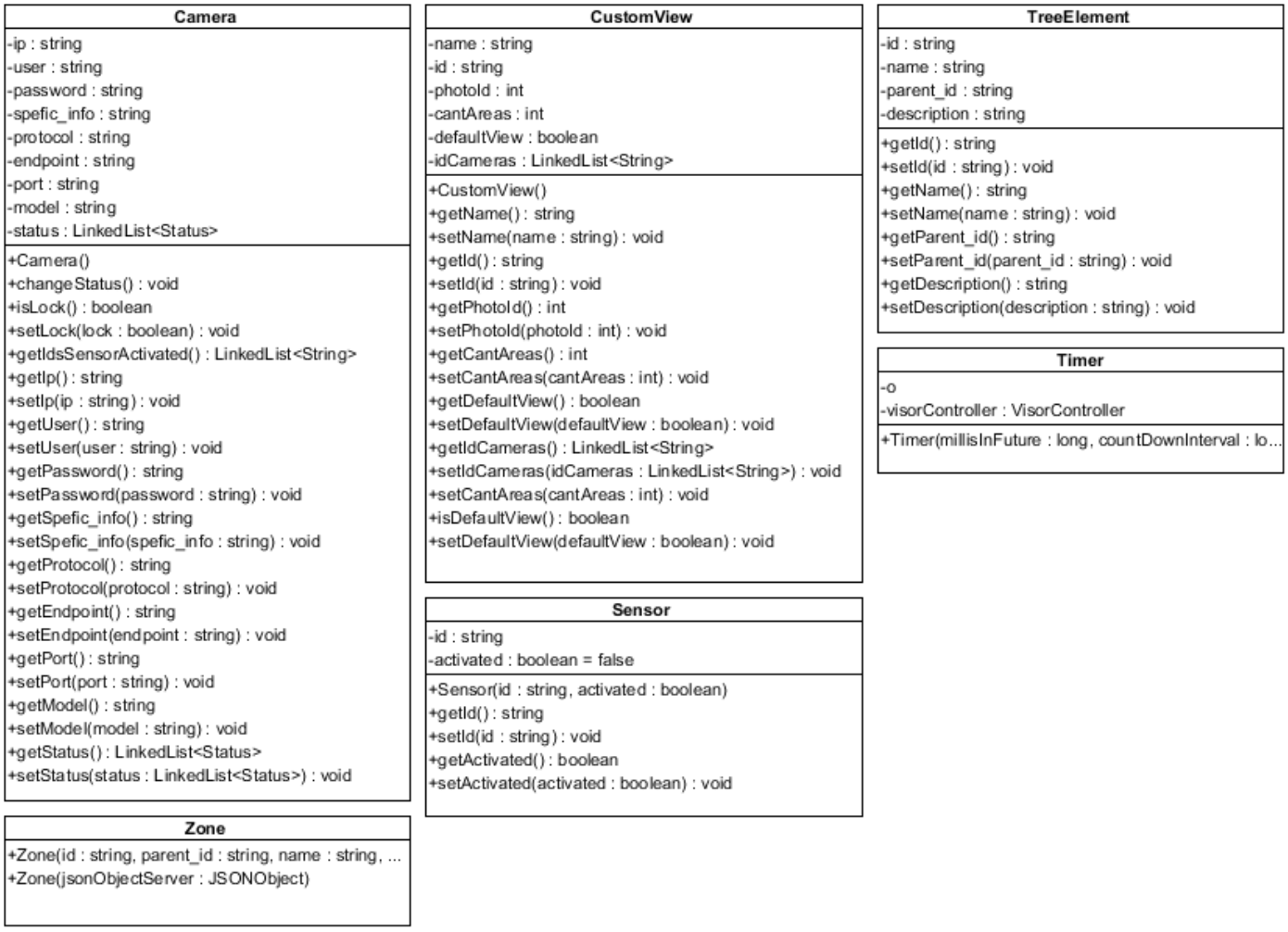


Fig. 7: Clases del Diagrama de Clases del Diseño del sistema: Capa Lógica de Negocio

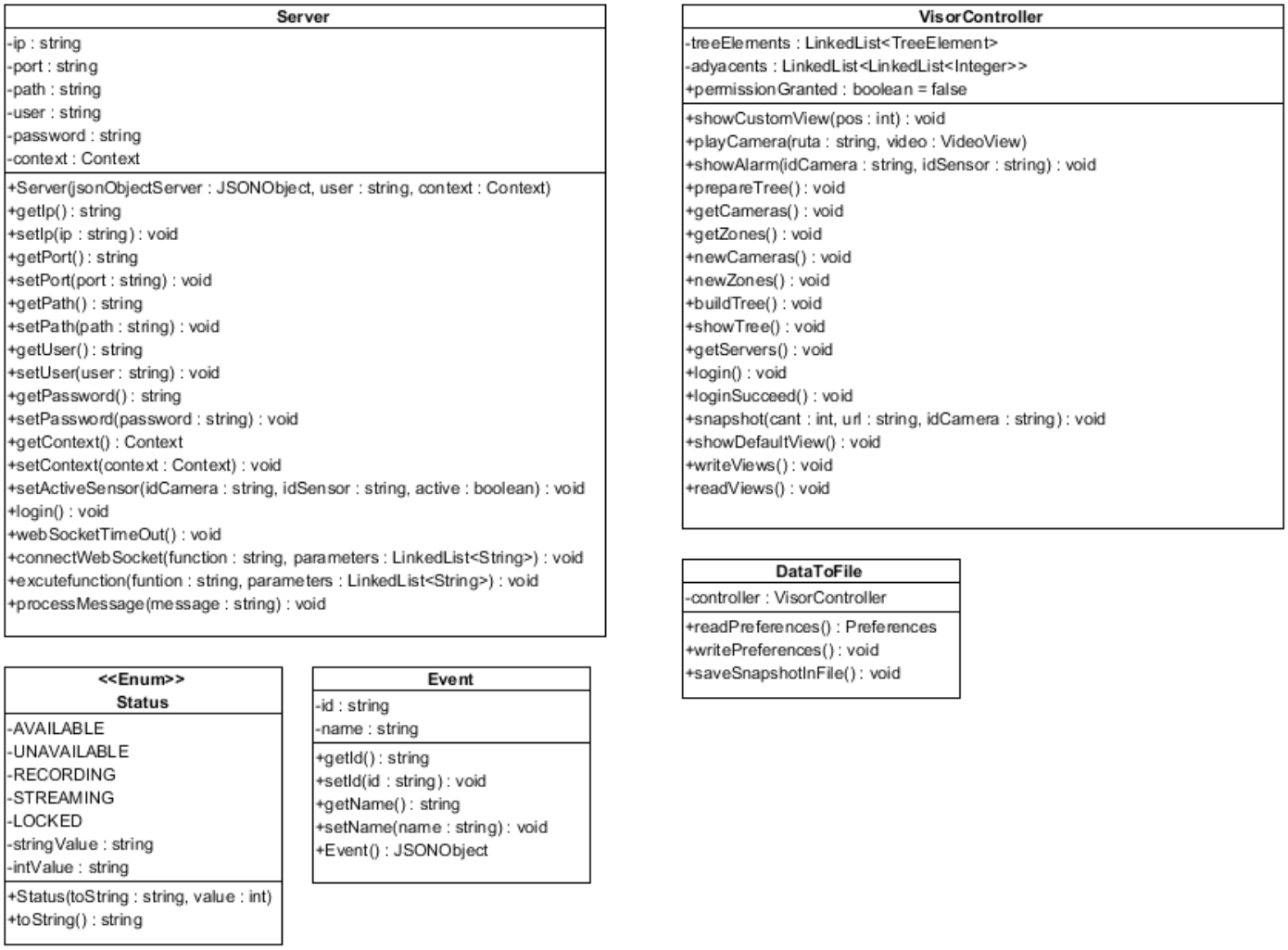


Fig. 8: Clases del Diagrama de Clases del Diseño del sistema: Capa Lógica de Negocio

2.6.6 Estándar de codificación

El estándar en el código de programación es muy importante en cualquier producto, principalmente si este involucra muchos desarrolladores. Esto asegura que el código sea de alta calidad, que contenga una cantidad baja de errores y sea fácil de mantener (Microsoft, 2017).

Principios generales

- Los nombres de cada uno de los elementos del programa deben ser significativos; su nombre debe explicar en lo posible el uso del elemento.
- No manejar en los programas más de una instrucción por línea.
- Declarar las variables en líneas separadas.
- Añadir comentarios descriptivos junto a cada declaración de variables, si es necesario.
- La mayoría de los elementos se deben nombrar usando sustantivos.
- Los atributos deben comenzar con letra minúscula y los métodos deben comenzar con letra minúscula.
- Especificar importaciones: especificar la ruta completa de la clase a utilizar. Esto posibilita conocer cuál es la clase utilizada en cada paquete importado, el código es más legible y entendible.
- Escribir métodos cortos: siempre que sea posible, escribir métodos pequeños y concretos. Sin embargo, no hay límites estrictos para la cantidad de líneas de código.

Nomenclatura de los elementos

Para una mejor organización y comprensión del código es recomendable definir reglas de nombramiento para cada tipo de elementos. Cumplir con esta nomenclatura permite crear software más escalables y fáciles de mantener. A continuación, se define la nomenclatura de los elementos a seguir en la solución:

Tabla 3: Nomenclatura de los elementos para el estilo de codificación

Elemento	Reglas de nombramiento
Clases, interfaces	Nombre sustantivo singular, con la primera letra en mayúscula y las demás en minúsculas. Si el nombre de la clase está comprimido en más de una palabra la primera letra debe de ser mayúscula. Letras mayúsculas seguidas no son permitidas. No hay separación entre palabras. Ejemplo: “ZendPDF” no es permitido mientras que “ZendPdf” sí lo es.

Archivos fuente	Nombre sustantivo singular completamente en minúscula y sin separación entre las palabras.
Constantes	Nombre sustantivo en mayúsculas. Para separar palabras se usará el guion bajo.
Paquetes y directorios	Nombre sustantivo singular en minúscula.

2.6.7 Conclusiones parciales

Los elementos que se muestran en este capítulo permiten la culminación de tareas trazadas en el inicio del trabajo investigativo, específicamente la tarea 4, lo cual se manifiesta en:

- Mediante la especificación de requisitos se identificaron, 36 requisitos funcionales, agrupados en 18 casos de uso, lo cual posibilitó aumentar la claridad de las funcionalidades que debe cumplir el sistema.
- La elaboración del modelo de dominio posibilitó un mejor entendimiento de los conceptos principales en el entorno de visualización de cámaras.
- La elaboración del diagrama de clases del diseño permitió obtener un enfoque claro del funcionamiento interno del software, facilitando así el descubrimiento temprano de posibles fallas en el diseño de la aplicación.
- El uso del estilo arquitectónico Llamada y Retorno y el patrón arquitectónico N-Capas, en conjunto con el uso de patrones de diseño GRASP Y GOF tributaron a una adecuada organización de la aplicación en dos componentes o capas (Presentación y Lógica de Negocio), lo que permitió obtener una solución escalable y flexible.
- El uso de los estándares de codificación definidos por el proyecto de Video Vigilancia SURIA junto con los definidos por Google, posibilitaron la obtención de un código legible.

CAPÍTULO III: IMPLEMENTACIÓN Y VALIDACIÓN DE LA SOLUCIÓN

3.1 Introducción

En el presente capítulo se hace un análisis de los resultados obtenidos durante el desarrollo de la solución propuesta. Se realiza el diagrama de componentes para describir la organización del sistema y sus relaciones, así como el modelo de despliegue. Además, se especifican las diferentes métricas para la validación de los requisitos y el diseño. Finalmente se realizan diferentes pruebas de software.

3.2 Modelo de Implementación

El modelo de implementación describe cómo los elementos del modelo del diseño se implementan en términos de componentes (ficheros de código fuente, ejecutables, entre otros). El modelo de implementación también describe cómo se organizan los componentes de acuerdo con los mecanismos de estructuración y modularización disponibles en el entorno de implementación, además en el lenguaje o los lenguajes de programación utilizados, y cómo dependen los componentes unos de otros (RUMBAUGH, et al., 2000).

3.2.1 Diagrama de Componentes

Según (Larman, 2003) *"Un diagrama de componentes muestra las dependencias lógicas entre componentes de software, sean éstos componentes: fuentes, binarios o ejecutables, ilustran las piezas del software, controladores embebidos, etc. Prevalecen en el campo de la arquitectura de software, pero pueden ser usados para modelar y documentar cualquier arquitectura de sistema, es decir para describir la vista de implementación estática de un sistema".* Llegado el momento de la implementación de la solución, todos los diagramas generados en fases anteriores se materializan en un sistema integrando las partes necesarias para la obtención de un producto final. Estas relaciones de dependencia se modelan a través del diagrama de componentes. Los Diagramas de Componentes se pueden clasificar en tres categorías como son:

-Componentes productos de trabajo: surgen durante el proceso de desarrollo y queda al final del mismo.

-Componentes de ejecución: se crean como consecuencia de un sistema de ejecución, por ejemplo: objetos que se instancian a partir de una dll[20].

[20] dll: Biblioteca de Enlace Dinámico

-Componentes de despliegue: necesarios y suficientes para formar un sistema ejecutable, por ejemplo: bibliotecas dinámicas (dll), ejecutables (*exe*).

A continuación, se muestra el diagrama de componentes donde se aprecia la división del sistema en componentes y las dependencias que se generan entre los mismos. Primeramente, se presenta la aplicación *SuriaVisor* que consiste en un ejecutable que depende de las bibliotecas *Java-websocket-1.3.1*, *gson-2.2.4*, *y* de los paquetes *Libvlc y FFmpegMediaMetadataRetiever*. La biblioteca *Java-websocket-1.3.1* permite establecer la comunicación con el servidor. La biblioteca *gson-2.2.4* permite la creación e interpretación de la información en la estructura de datos *JSON* que se intercambia con el servidor. El paquete *Libvlc* contiene un conjunto de bibliotecas como: *libanw.10.so, libc++_shared.so,* entre otras, las cuales son utilizadas para reproducir el flujo de video de las cámaras y el paquete *FFmpegMediaMetadataRetieve* contiene un conjunto de bibliotecas como: *libavcodec.so, libavformat.so*, entre otras, que posibilitan la toma de instantáneas y las ráfagas de instantáneas.

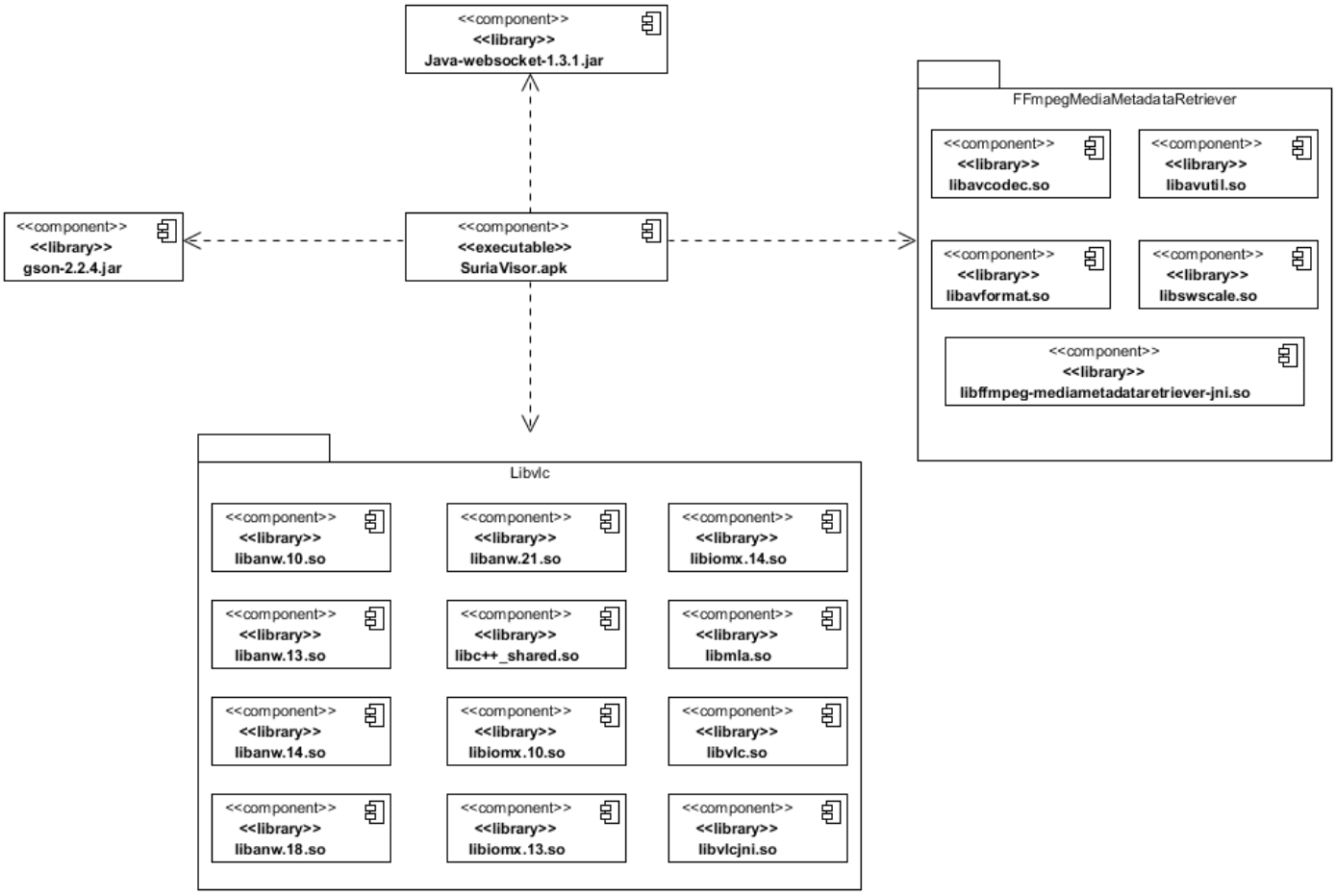


Fig. 9: Diagrama de Componentes de Código Ejecutable

3.2.2 Modelo de Despliegue

Ivar Jacobson y otros en el libro *El proceso unificado de desarrollo de* software definen el modelo de despliegue como: *"un modelo de objetos que describe la distribución física del sistema en términos de cómo se distribuye la funcionalidad entre los nodos del cómputo"* (RUMBAUGH, y otros, 2000).

Este modelo es primordial en las actividades de diseño e implementación, pues la distribución que posee el sistema tiene mucha influencia en su diseño. El modelo de despliegue representa en sí mismo una correspondencia entre el software y el hardware. Un nodo es un objeto físico que existe en tiempo de ejecución y que representa algún tipo de recurso computacional, en él se ejecutan los componentes y, además, posee relaciones que representan medios de comunicación entre ellos. Los componentes son los elementos que intervienen en la ejecución del sistema.

3.2.3 Diagrama de despliegue

El diagrama de despliegue muestra las relaciones físicas entre los componentes hardware y software en el sistema final, es decir la configuración de los elementos de procesamiento en tiempo de ejecución y los componentes de software (Pressman, 2010). En la Figura 7 se muestra el diagrama de despliegue perteneciente al sistema propuesto.

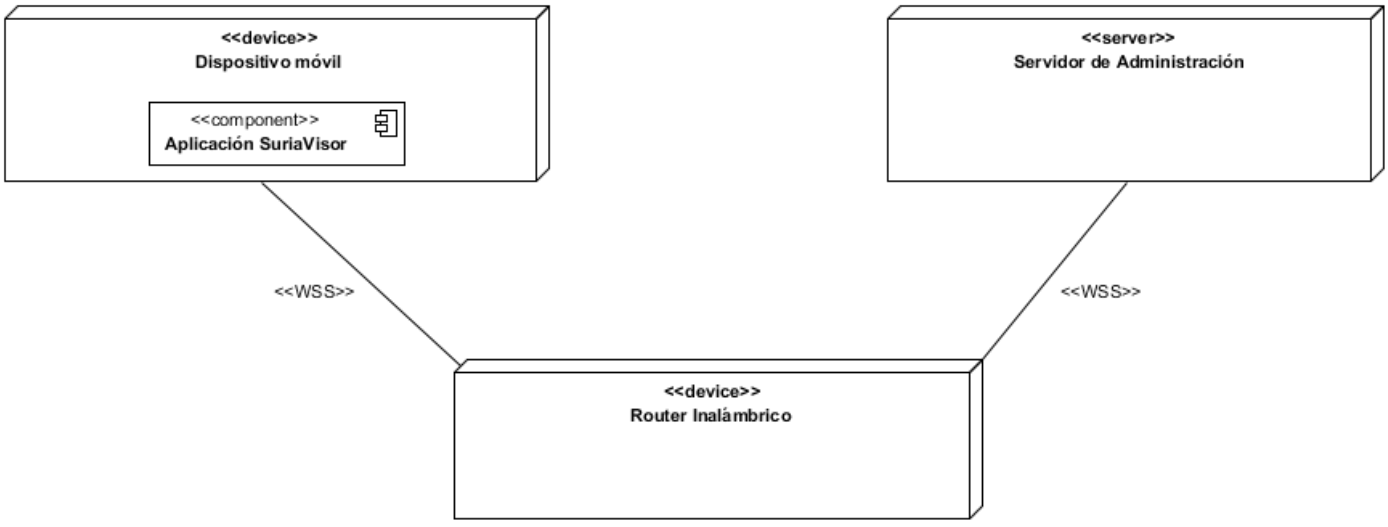


Fig. 10: Diagrama de Despliegue

Descripción de los nodos físicos del sistema:

<<*device*>> Dispositivo móvil: Dispositivo móvil donde se encuentra instalada la aplicación SuriaVisor.

<<*component*>> Aplicación SuriaVisor: Aplicación que se comunica con el Servidor de Administración del sistema de Video Vigilancia SURIA 3.0 y consume los servicios que este brinda.

<<*server*>> Servidor de Administración: Servidor que brinda servicios web a la aplicación SuriaVisor.

<<*device*>> Router Inalámbrico: Dispositivo de interconexión de red que brinda conexión inalámbrica al dispositivo móvil para establecer comunicación con el Servidor de Administración.

Nota: El dispositivo tiene que tener obligatoriamente el sistema operativo Android.

3.3 Validación de la solución

Ian Sommerville define que la validación tiene como objetivo: *"asegurar que el sistema de software satisface las expectativas del cliente"*. Así como también establecer la seguridad de que el software está hecho para un propósito, o sea, debe ser lo suficientemente bueno para su uso pretendido (Sommerville, 2011). A continuación, se especifican los tipos de pruebas a realizar para garantizar la calidad de la solución.

3.3.1 Pruebas Funcionales

Las pruebas funcionales tienen como objetivo asegurar que el software cumpla con las especificaciones requeridas y eliminar los posibles errores que este pueda tener. Para la realización de estas pruebas previamente se deben crear diseños de modelos de pruebas, donde se especifican las entradas y salidas del sistema, los flujos de eventos y descripción detallada de las respuestas del sistema (Pressman, 2010). Las pruebas funcionales se aplican de forma manual y desde el punto de vista del usuario.

Pruebas de caja negra

El método de prueba caja negra o prueba de comportamiento se centra en los requisitos funcionales del sistema. La correcta aplicación de este tipo de prueba puede arrojar no conformidades como: funciones incorrectas o faltantes, errores de interfaz, errores en estructura de datos o en el acceso a bases de datos externas, errores de comportamiento o desempeño y errores de inicialización y término (Pressman, 2010). Con el fin de garantizar el correcto funcionamiento del sistema de Video Vigilancia SURIA se ejecutaron los casos de prueba diseñados para todos los casos de uso del sistema, utilizando la técnica Partición de

Equivalencia. En la Tabla 4 se muestra el diseño de caso de prueba del caso de uso Autenticar usuario en el visor. Para los otros casos de prueba dirigirse al Anexo no.2.

Tabla 4: Caso de prueba del caso de uso Autenticar usuario en el visor

Escenario	Descripción	Usuario	Contraseña	Respuesta del sistema	Flujo central
EC 1.1 Autenticar usuario en el Visor APK correctamente.	El usuario introduce los parámetros requeridos para autenticarse en la aplicación.	V apk	V apk	El sistema valida los datos de entrada. Muestra la interfaz principal de la APK.	Ejecutar APK/ Introducir datos/ seleccionar botón Aceptar.
EC 1.2 Autenticar usuario en el Visor APK incorrectamente.	El usuario introduce incorrectamente los datos de autenticación y/o deja campos vacíos.	I texto_23/vacío	V 1234567	Muestra el siguiente mensaje: Estableciendo conexión ¨Por favor espere...¨ El sistema valida los datos de entrada. Muestra los siguientes mensajes según el/los error/es cometido/s: -¨No se ha podido establecer conexión con el servidor, inténtelo más tarde¨ -¨Campo obligatorio¨ (Este mensaje se muestra cuando existe algún campo obligatorio vacío)	Ejecutar APK/ Introducir datos/ seleccionar botón Aceptar.
		V apk	I texto_23/vacío		
EC 1.3 Expira la sesión del usuario autenticado.	El usuario queda en estado de inactividad.	N/A	N/A	Cierra la sesión del usuario autenticado. Muestra el mensaje de error: ¨Su sesión ha expirado. Por favor, vuelva a identificarse¨	Muestra la interfaz principal de la aplicación

Resultados de las pruebas de caja negra:

Durante el transcurso de la ejecución de las pruebas de caja negra se identificaron 15 no conformidades (NC) en 3 iteraciones como se muestra en la Figura 11:

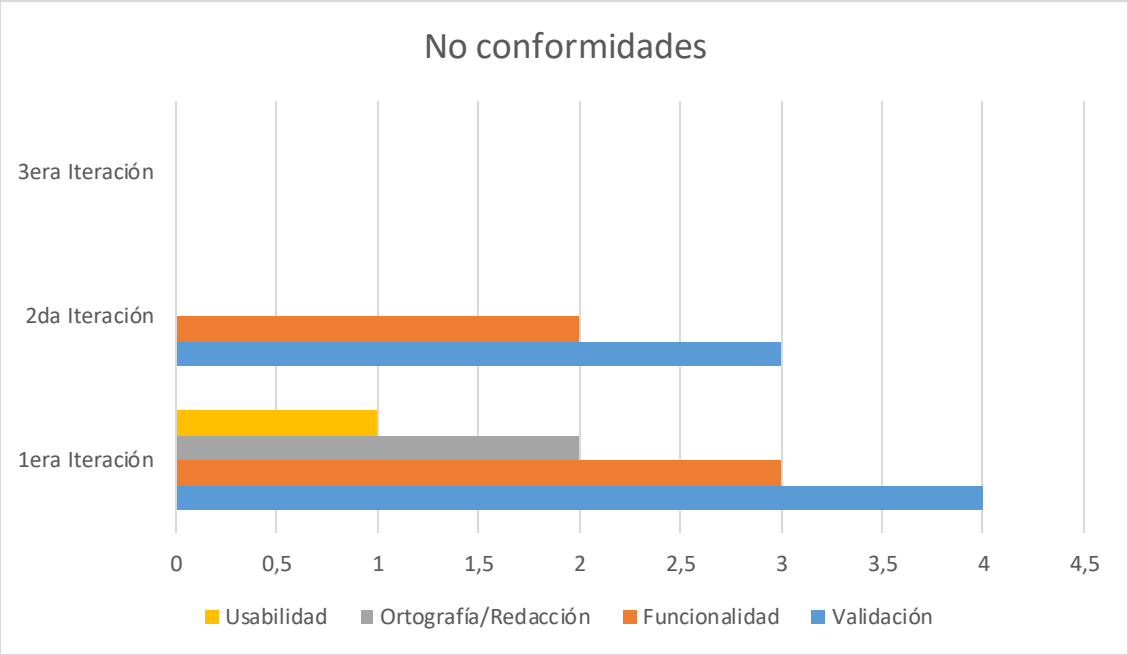


Fig. 11: Resultados de las pruebas de caja negra

Entiéndase por NC de validación a los errores relativos en todo tipo de validación, NC de funcionalidad a cuando el resultado mostrado al realizar una acción no es el esperado, NC de usabilidad cuando el sistema no cumple con las interacciones de un usuario y las NC de Ortografía/Redacción a los errores ortográficos que se hallen en la aplicación por ejemplo el nombre de un botón y a los errores de redacción que se hallen en la misma, ejemplo en los mensajes mostrados.

No conformidades detectadas en la 1ra iteración

NC de Validación:

- Validar cuando expira la sesión del usuario autenticado.
- Mostrar mensaje cuando no existe cámaras ni zonas en el sistema. (CU 5: Mostrar árbol de cámaras y zonas adicionadas en el sistema según los permisos del usuario autenticado en el visor y CU 6: Visualizar flujo de video de una cámara en el visor.)

- Validar el campo Nombre de la vista de la interfaz Adicionar Vista. (El campo solo debe admitir letras y números)
- Validar que no se pueda agregar una vista sin nombre.

NC de Funcionalidad:

- Mostrar mensaje de eliminación en la gestión de las vistas.
- Mientras que el usuario está haciendo uso del sistema, este cierra la sesión inesperadamente mostrándole un mensaje que se ha perdido la conexión con el servidor.
- Mostrar mensaje de listado de vistas en blanco (no existen vistas en el sistema).

NC de Usabilidad:

- Resaltar los campos obligatorios en el APK.

NC de Ortografía/Redacción:

- Cambiar el nombre del botón de autenticación (Login) por Aceptar.
- Falta de ortografía en la palabra “area” situada en la interfaz para adicionar vista.

No conformidades detectadas en la 2ra iteración

NC de Validación:

- Establecer límite de caracteres en el campo nombre de la vista de la interfaz de Adicionar vista.
- Realizar las validaciones de la configuración con el servidor en la interfaz correspondiente.
- No reproducir el flujo de una cámara con la dirección vacía.

NC de Funcionalidad:

- Establecer una X cantidad de instantáneas en el visor.
- No se pueden realizar peticiones de grabación manual en el visor. Se cierra la aplicación.

Al culminar cada iteración de pruebas se procedió a resolver las no conformidades detectadas, dando paso a una nueva iteración. Al concluir la 3ta iteración, no se detectaron no conformidades, posibilitando así el correcto funcionamiento y comportamiento de todos los casos de uso.

3.3.2 Pruebas de rendimiento

Las pruebas de rendimiento se realizan para determinar lo rápido y capaz que realiza una tarea un software en condiciones particulares de trabajo. En este tipo de pruebas se valora el tiempo de respuesta, los retardos en la entrega de mensajes, si la aplicación se bloquea, el consumo de memoria y del CPU[21] o la carga de los problemas de velocidad (Díaz, 2013).

Al realizar la evaluación del rendimiento se seleccionaron los parámetros: tiempo de respuesta, uso del CPU y consumo de memoria RAM. Estos fueron evaluados en varios dispositivos móviles realizando las funcionalidades de mayor consumo de recursos de hardware. A continuación, se muestran específicamente los parámetros de evaluación:

- Tiempo de respuesta, uso del CPU y consumo de memoria RAM al acceder al servidor.

- Tiempo de respuesta, uso del CPU y consumo de memoria RAM al reproducir el flujo de video de 1 a *n* cámaras.

- Tiempo de respuesta, uso del CPU y consumo de memoria RAM al realizar una ráfaga de instantáneas.

Resultados de las pruebas de rendimiento

Las pruebas fueron ejecutadas en 3 dispositivos móviles con distintas propiedades, los tiempos de respuesta fueron tomados con un cronómetro y registrados en una tabla comparativa para su posterior análisis. Para comparar el uso del CPU y el consumo de memoria RAM se utilizó la herramienta *Android Monitor* del IDE *Android Studio*, que muestra en tiempo real el comportamiento de estos parámetros durante la ejecución de una aplicación específica.

[21] CPU: Central Processing Unit

A continuación, se muestra una tabla comparativa donde se evidencia el promedio de los resultados de las pruebas:

Tabla 5: Resultados de las pruebas de rendimiento

Parámetros	Tiempo de respuesta (ms)			Uso del CPU (%)			Consumo de RAM (Mb)		
	1	2	3	1	2	3	1	2	3
Loguin	1060	4012	5638	10,4	5,78	13,45	35,95	14,3	43,84
Reproducir flujo (vista 1)	4840	2345	863	2,2	10,43	3,25	37,02	11,33	43,68
Reproducir flujo (vista 1)	6200	2637	1566	3,17	11,21	5,41	44,59	13,4	44,86
Reproducir flujo (vista 1)	7360	3185	1846	5,92	17,63	6,33	45,6	13,72	48,77
Reproducir flujo (vista 1)	7841	4687	2687	5,87	19,42	7,34	45,97	13,76	54,31
Reproducir flujo (vista 1)	10074	4863	3468	6,01	20,98	7,41	49,2	15,48	54,73
Reproducir flujo (vista 1)	12687	5126	4073	5,56	22,03	7,66	51,37	15,79	55,19
Instantáneas	12354	5496	10257	18,01	35,48	27,95	21,83	11,36	39,81

Leyenda:

1: Sony Xperia M4 Aqua

2: Lenovo IdeaTab

3: Samsung Galaxy Tab

La realización de esta prueba arrojó como resultado que la aplicación hace un uso razonable de los recursos del dispositivo, no sobrecargando la RAM ni el microprocesador y respondiendo en un tiempo no agotador para el usuario. Pues haciendo uso de esta prueba en diferentes ambientes, desde microprocesadores más potentes hasta más sencillos, desde mucha hasta poca RAM; se comprobó que en todas hubo un equilibrio del tiempo de respuesta y consumo por parte de la aplicación, permitiendo ser apta para su uso en cualquier dispositivo.

Dispositivos probados:

Tabla 6: Características de los dispositivos probados

Modelo y características físicas	Sony Xperia M4 Aqua
Sistema Operativo	Lollipop 5.0
CPU Procesador / Núcleos	Octa-Core, 2 procesadores: 1.5Ghz Quad-Core ARM Cortex-A53 1Ghz Quad-Core ARM Cortex-A53
Memoria RAM	2GB LP-DDR3

Modelo y características físicas	Lenovo IdeaTab
Sistema Operativo	Jelly Bean 4.2.2
CPU Procesador / Núcleos	MediaTek 6575 Single-Core Cortex-A9 1.2 GHz
Memoria RAM	RAM: 1GB LP-DDR2

Modelo y características físicas	Samsung Galaxy Tab
Sistema Operativo	Lollipop 5.0.2
CPU Procesador / Núcleos	Procesador octa-core Exynos 5 Octa (4 Cortex-A15 1.9GHz + 4 Cortex-A7 1.3GHz) / quad-core Qualcomm Snapdragon 800 2.3GHz
Memoria RAM	3GB LP-DDR3

3.3.3 Pruebas de integración

Las pruebas de integración son una técnica sistemática para construir la arquitectura del software. Son utilizadas para comprobar que los módulos o partes de un software funcionan correctamente como un todo. Existen dos formas de realizar la integración del software: de forma no incremental y de forma incremental. La forma no incremental no es recomendable debido al gran cúmulo de errores que puede generar (Pressman, 2010). Para la realización de este tipo de prueba se aplicó un enfoque incremental de integración ascendente.

Incremental ascendente

La prueba de integración ascendente se aplica con la construcción y prueba de módulos atómicos. Debido a que los componentes se integran de abajo hacia arriba, siempre está disponible el procesamiento requerido para los componentes subordinados a un determinado (Pressman, 2010).

En la medida que se avanzó en la implementación de la solución esta prueba se fue llevando a cabo, pues una vez terminada la implementación de una funcionalidad del sistema, se procedió a probar su correcta ejecución en conjunto con el servidor de administración, comprobando así la integración de la funcionalidad con el mismo. Al terminar la implementación de otra funcionalidad se repitió el procedimiento anterior, sumado con la comprobación de las funcionalidades ya implementadas. Este procedimiento se repitió a lo largo de todo el desarrollo de la solución hasta quedar implementadas todas las funcionalidades. Una vez conformada la solución se procedió a probar la aplicación de forma general para constatar su integración al Servidor de Administración.

Resultados de las pruebas de integración

A medida que se fueron implementando las funcionalidades del sistema se fue probando su correcta ejecución con el servidor de administración. Durante estas pruebas iterativas se detectaron no conformidades tales como: al realizar la funcionalidad *Login* desde dos dispositivos con el mismo usuario, se cerraba la aplicación sin decir el motivo, el estado que se mostraba de una cámara en la aplicación no correspondía con el que tenía dicha cámara en el servidor, no se mostraban los sensores disponibles en el servidor debido a que la información que se enviaba del servidor no era la que se esperaba en la aplicación.

Aplicar este tipo de prueba durante todo el proceso de implementación permitió corregir el mal comportamiento de las funcionalidades con respecto a su interacción con el servidor de administración. De esta manera, al culminar la aplicación y ejecutar la última iteración de la prueba de integración se aseguró la correcta integración de la aplicación SuriaVisor.

3.4 Conclusiones parciales

Los elementos que se muestran en este capítulo permiten la culminación de tareas trazadas en el inicio del trabajo investigativo, específicamente las tareas 5 y 6, lo cual se manifiesta en:

- La realización del diagrama de componentes permitió comprender mejor las dependencias en cuanto a bibliotecas, apreciándose la interacción entre los componentes.
- El diagrama de despliegue ayudó a entender mejor la relación entre los componentes físicos que intervienen en el sistema final.
- Con la realización de las pruebas de caja negra, rendimiento e integración, se encontraron no conformidades referentes a las funcionalidades de la aplicación, las cuales fueron erradicadas posteriormente. La realización de estas pruebas aportó positivamente a la calidad del software.

Conclusiones Generales

Al concluir el presente trabajo de diploma se alcanzó dar cumplimiento a las tareas de investigación propuestas al inicio del mismo, por lo cual se obtuvo satisfactorio cumplimiento al objetivo general planteado inicialmente. A partir de estos resultados se arriba a las siguientes conclusiones:

- La caracterización y revisión del estado del arte de los sistemas de Video Vigilancia, permitió afirmar que las soluciones existentes, de una forma u otra tributan a la investigación, pero no resuelven la problemática planteada, evidenciándose el desarrollo de la solución propuesta en esta investigación.
- Las herramientas y tecnologías seleccionadas permitieron la obtención de una amplia documentación, el desarrollo de una solución siguiendo estándares actuales utilizados internacionalmente, además de sentar las bases para futuras versiones.

- La especificación de los requisitos del software posibilitó trazar claramente las funcionalidades y características que debe cumplir la aplicación, lo cual fue de gran importancia durante el desarrollo de la aplicación.
- La realización de pruebas de caja negra, rendimiento e integración demostró el correcto funcionamiento de la aplicación SuriaVisor para dispositivos móviles con sistema operativo Android.
- El desarrollo de una aplicación nativa contribuyó a la portabilidad del sistema al extender su uso a dispositivos móviles con sistema operativo Android.
- El desarrollo de la aplicación SuriaVisor enfocada a la experiencia de usuario mejoró el uso en la visualización y el control de las cámaras en el sistema de Video Vigilancia SURIA.

Recomendaciones

Luego de haber analizado los resultados del presente trabajo de diploma, resulta factible proponer la siguiente recomendación:

- Permitir al usuario la manipulación de la cámara en dependencia de las capacidades que posea (movimiento arriba, abajo, izquierda, derecha, zoom).

Referencias Bibliográficas

Aguado Terrón, Juan Miguel and Martínez Martínez, Inmaculada J. 2009. *De la Web social al Móvil 2.0: el paradigma 2.0 en el proceso de convergencia mediática de la comunicación móvil.* s.l. : El profesional de la información, 2009. 2.

Android Studio. 2016. Android Studio The Official IDE for Android. [Online] 2016. https://developer.android.com/studio/index.html?hl=es#features..

Apple. 2017. Discussions Apple. [Online] 2017. https://discussions.apple.com/.

Axis Communications. 2016. Axis Communications. [Online] 2016. https://www.axis.com/global/es/products/axis-camera-station.

Benbourahla, Nazim. 2015. *Android 5: principios del desarrollo de aplicaciones Java.* s.l. : Ediciones Eni, 2015.

Blog UCI de Android. 2017. Comunidad Android de la UCi. [Online] 2017. https://android.uci.cu/.

Brito, Nacho. 2009. *Manual de desarrollo web con GRAILS.* 2009.

Buschmann, Frank, y otros. 2001. *Pattern-Oriented Software Architecture.* Germany : John Willey & Sons, 2001.

Curbera, Francisco, et al. 2002. *Unraveling the Web services web: an introduction to SOAP, WSDL, and UDDI.* s.l. : IEEE Computer Society, 2002. Vol 6.

CVRL Industries Inc. 2015. WECU Surveillance.com. [Online] 2015. [Cited: octubre 25, 2016.] http://www.wecusurveillance.com/cctvhistory.

Desarrollo de aplicaciones sobre Android. **VANEGAS, Carlos Alberto. 2013.** 2, 2013, Vol. 9.

Díaz, Santiago. 2013. *Mejores prácticas en las pruebas de aplicaciones móviles.* Madrid : s.n., 2013.

Empresas Reqquality. 2016. Reqquality sistemas. [Online] 2016. [Cited: octubre 27, 2016.] http://www.reqquality.com/ventajasde-la-seleccion-de-camaras-ip-para-instalaciones-cctv.

Ferras, Claudia Beatriz Larramendi. 2012. *Propuesta de arquitectura para desarrollo de aplicaciones compuestas para dispositivos móviles con Android.* La Habana : s.n., 2012.

Google. 2017. Android Developer. [Online] 2017. [Cited: abril 10, 2017.] https://developer.android.com/studio/build/index.html.

Google Developers. 2017. Developers. [Online] 2017. https://developer.android.com/.

Google Play. 2016. Google Play. [Online] 2016. https://play.google.com/store/apps/details?id=com.alejandrobravo.android.mobotix.

IBM. 2016. IBM developerWorks. *Introducción a SOA y servicios web.* [Online] 2016. [Cited: diciembre 15, 2016.] https://www.ibm.com/developerworks/ssa/webservices/newto/service.html.

Jimenez Marin, Alonso and Perez Montes, Francisco Manuel. 2016. *Aprende a programar con Java 2da edición.* s.l. : Ediciones Paraninfo, 2016.

Largo García, Carlos Alberto y Marín Mazo, Arledy. 2005. *Guía técnica para la evaluación de software.* 2005.

Larman, Craig. 2003. *UML Y PATRONES. Una introducción al análisis y diseño orientado a objetos y al proceso unificado.* Madrid : Pearson Educación. SA, 2003. 2.

Lionbridge. 2016. Lionbridge. [Online] 2016. [Cited: Noviembre 10, 2016.] http://www.lionbridge.com/files/2012/11/Lionbridge-WP_MobileApps2.pdf.

Marset, Rafael Navarro. 2006. *Modelado, diseño e implementación de servicios web.* 2006. 15.

Microsoft. 2017. Microsoft. *Revisiones de código y estándares de codificación.* [Online] 2017. https://msdn.microsoft.com/eses/library/aa291591%28v=vs.71%29.aspx..

Oracle Corporation. 2016. Java. *¿Cómo puedo empezar a desarrollar programas Java con Java Development Kit (JDK)?* [Online] 2016. https://www.java.com/es/download/faq/develop.xml.

Pozo, Julián David Morillo. 2011. *Introducción a los dispositivos móviles.* 2011.

Pressman, Roger S. 2010. *Ingeniería de Software. Un enfoque práctico.* s.l. : McGraw-Hill , 2010. 7.

Pulido, Nieves Pavón. 2013. *Componentes básicos de Android.* 2013.

RAO, Jinghai and SU, Xiaomeng. 2004. *A survey of automated web service composition methods. En International Workshop on Semantic Web Services and Web Process Composition.* Berlin Heidelberg : s.n., 2004.

Read the Docs. 2012. Android OS. [Online] 2012. [Cited: diciembre 5, 2016.] http://androidos.readthedocs.io/en/latest/data/historia/.

REYNOSO, Carlos Billy. 2004. *Introducción a la Arquitectura de Software.* 2004.

Rubio, Miguel Vélez. 2015. *Evaluación de "Web Sites" utilizando principios de Ingeniería de Usabilidad.* 2015.

RUMBAUGH, James, JACOBSON, Ivar and BOOCH, Grady. 2000. *El proceso unificado de desarrollo de software.* 2000.

Sanchez, Tamara Rodríguez. 2014. *Metodología de desarrollo para la actividad productiva de la UCI.* La Habana : s.n., 2014.

SCHILDT, Herbert. 2001. *Manual de referencia: Java 2.* s.l. : Mc Graw Hill, 2001.

Seguridad Vía IP. 2017. seguridad via ip. [Online] 2017. [Cited: octubre 30, 2016.] http://www.seguridadviaip.com/preguntas_frecuentes_camaras_de_seguridad_cctv_ip _wifi_dvr.php.

Sommerville, Ian. 2011. *Software Engineering.* New York : Pearson Education Inc., 2011. 9.

Source. 2017. Source. [Online] 2017. https://source.android.com/source/,.

Swift. 2017. Swift. [Online] 2017. https://swift.org/.

Udacity. 2017. Udacity. [Online] 2017. https://www.udacity.com/courses/android.

UsabilityNet. 2006. UsabilityNet. [Online] Enero 2006. [Cited: Abril 25, 2017.] http://www.usabilitynet.org/tools/r_international.htm.

Vásquez, Solangel Rodríguez. 2016. *Sistema de Video Vigilancia SURIA 3.0.* 2016.

Visual Paradigm. 2016. Visual Paradigm. [Online] 2016. http://www.visual-paradigm.com.

VIVOTEK inc. 2016. VIVOTEK. [Online] 2016. http://www.vivotek.com/iviewer/#views:view=jplist-grid-view.

Zayas, Carlos Álvarez de. 2015. *Metodología de la Investigación Científica.* Cochabamba : Kipus, 2015. 7.