Universidad de las Ciencias Informáticas

FACULTAD 6

Título: Módulo de Análisis para el Sistema de Video Vigilancia Suria en Qt

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

Autor: Neybel González Ibargollín

Tutor: Ing. Reynier Pupo Gómez

Año del 54 Aniversario del Triunfo de la Revolución

Junio de 2012

PENSAMIENTO

"No se vive celebrando victorias, sino superando derrotas"

DECLARACIÓN DE AUTORÍA

Declaramos ser autores de la presente tesis y reconocemos a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

Para que así conste firmo la presente a los ____ días del mes de _____ del año _____.

Neybel González Ibargollín

_____
Firma del Autor

Ing. Reynier Pupo Gómez

_____
Firma del Tutor

DATOS DE CONTACTO

Tutor:

Ing. Reynier Pupo Gómez.

Graduado de la Universidad de las Ciencias Informáticas (UCI) en el año 2010-2011. Es Instructor Recién Graduado. Pertenece al Departamento Señales Digitales del Centro de Geoinformática y Señales Digitales (GEYSED) de la Facultad 6. Actualmente se encuentra de jefe del módulo Video Vigilancia en Qt.

e-mail: rgomez@uci. cu

AGRADECIMIENTOS

Quiero agradecer primeramente a mi familia porque son la razón de mi vida, la luz que alumbran mis pasos, mi orgullo e inspiración.

En especial quiero agradecer a mi mamá que ha sido mi guía en todo momento, la persona que siempre ha estado conmigo y que me ha apoyado cuando más le necesité, tanto en mis tristezas como en mis alegrías.

Quiero agradecerle a Pipu que ha sido incondicional conmigo en mis necesidades y que si no fuera por él no me hubiese podido mantener aquí económicamente.

Quiero agradecerle a mi hermano Yoe por ser el promotor principal de que yo estudiara esta carrera, que es y será siempre la mejor decisión que he tomado en toda mi vida, además de ser para mí la luz de mis ojos, mi media mitad.

Quiero agradecerle a Raide la persona que nunca me falló en mis momentos de angustias, la persona que más me apoyó e impulsó en la carrera, a tí Raide te debo mucho, como tú existen pocas personas, humilde, noble y solidario.

Quiero agradecerle a todas mis amistades de la universidad que de una forma u otra contribuyeron en mi formación como profesional, en especial a "la flaca" y a "la enana" que siempre fueron excepcionales conmigo.

A todas las personas del proyecto Video Vigilancia Suria que me brindaron su apoyo y ayuda, en especial a mi tutor Pupo por haberme brindado sus conocimientos.

DEDICATORIA

Les dedico el presente trabajo de diploma a mis padres que son el faro que guían mis pasos así como a mi hermano que es la persona que siempre confío en mí porque tuvo la intuición de que yo llegaría lejos.

RESUMEN

Con la constante evolución de los medios tecnológicos y el auge de la informática, han surgido soluciones capaces de integrar cámaras y video sensores que facilitan a una persona a vigilar áreas que por sus dimensiones resultaba imposible. La Universidad de las Ciencias Informáticas cuenta con el proyecto productivo Video Vigilancia Suria, el cual es el responsable de fortalecer la seguridad mediante el procesamiento de imágenes y señales de video digital. Este posibilita optimizar y añadir nuevas prestaciones a los sistemas de vigilancia. Actualmente el proyecto posee limitantes debido a que los videos-sensores desarrollados por el sistema no pueden ser aprovechados por la falta de comunicación entre sus componentes. Con la elaboración de un módulo capaz de realizar análisis inteligente a través de flujos de videos para el sistema de Video Vigilancia Suria en QT, se establece la comunicación entre el gestor y los videos-sensores, se cargan los videos-sensores y la configuración que realiza un operador, de manera que cuando sea detectado algún suceso se notifique al gestor. La solución que se presenta es multiplataforma, lo que amplía las posibilidades de utilización y provecho del análisis inteligente de video a través de los videos-sensores.

Palabra claves

Análisis inteligente, comunicación, gestor, multiplataforma, videos-sensores, video vigilancia.

ÍNDICE DE CONTENIDO

ÍNDICE DE FIGURA

ÍNDICE DE TABLAS

INTRODUCCIÓN

Desde el punto de vista tecnológico, los sistemas de vigilancia han evolucionado atravesando diferentes etapas. La primera generación de sistemas de vigilancia basada en video empleó transmisión analógica. En la toma de decisiones, siempre es el operador el encargado de realizar todas las tareas de análisis de secuencias de video presentadas en varios monitores situados en una sala de control remota, donde las escenas monitorizadas por las distintas cámaras se concatenan en un orden periódico y predefinido. Adicionalmente, la vigilancia tradicional precisa gran cantidad de espacio de almacenamiento. Todo lo que captura una cámara de seguridad se graba en cintas que, o bien se sobrescriben periódicamente, o bien se guardan en un archivo de video. Este procedimiento limita la duración de video que puede guardarse y hace que el tiempo necesario para su revisión sea elevado.

La segunda generación de sistemas de vigilancia se basa, principalmente, en métodos de procesamiento y comunicación híbridos analógico-digitales, o completamente digitales. Aprovechan la flexibilidad ofrecida por los primeros algoritmos de procesamiento de videos que permiten centrar la atención del operador en un grupo de situaciones de interés. Además, las facilidades proporcionadas por los primeros métodos de compresión digital para aprovechar el ancho de banda de transmisión.

Con la evolución de los medios tecnológicos y el auge de la informática surgen soluciones capaces de integrar equipamiento como cámaras y sensores de detección de movimiento, captación de chapas de vehículos, sensores de perimetrado virtual; permitiendo que una persona pueda vigilar áreas que por sus dimensiones antes era imposible de realizar. Cada vez son más grandes las ciudades y edificios por lo que de igual forma es imprescindible que las soluciones informáticas también evolucionen pues el trabajo que antes consistía en monitorear, ahora consiste en automatizar dichas soluciones.

Cuba no está ajena del avance tecnológico actual, es por eso que el país dedica parte de su esfuerzo a promover y fomentar el desarrollo de la informática, lo cual se ha convertido en una tarea primordial. La industria cubana del software (Incusoft) en estos momentos se ha convertido en una significativa fuente de ingresos nacionales pues fue creada con el objetivo de aunar los esfuerzos individuales que han venido realizando diversas instituciones del país en este campo, para alcanzar una fortaleza que permita incursionar, con más efectividad, en los mercados extranjeros.

La Universidad de las Ciencias Informáticas (UCI) no solo está destinada a actividades formativas, sino que es también una fuente importante de ingresos al país. Los profesionales altamente calificados en la rama de la informática que contribuyen con su potencial científico al cumplimiento de los planes en el campo del desarrollo de software, se vinculan al estudio-trabajo como modelo de formación para contribuir al desarrollo de aplicaciones informáticas de acuerdo al centro al que pertenezcan.

Nuestra Universidad consta de 7 facultades donde cada una de ellas está íntimamente vinculada a la producción e investigación. La facultad 6 consta de un Centro de Desarrollo Geoinformática y Señales Digitales (GEYSED), este centro tiene la misión de desarrollar productos, servicios y soluciones informáticas en el campo del procesamiento de Señales Digitales y la Geoinformática. Uno de los proyectos que se lleva a cabo en este centro es Video Vigilancia Suria perteneciente al Departamento de Señales Digitales.

El proyecto Video Vigilancia Suria centra su atención en el proceso de video vigilancia que permite fortalecer la seguridad en cualquier institución donde sea desplegado. La ejecución de este proyecto incluye la infraestructura tecnológica para soportar un conjunto de cámaras de seguridad, que se puedan gestionar dentro de una red de datos asegurando la visualización, almacenamiento y trasmisión de los flujos de videos generados en cada uno de los dispositivos de adquisición.

Video Vigilancia Suria tiene sensores creados en el lenguaje de programación C++, los sensores se utilizan como instrumento para medir de forma automática una variable; dicha variable puede ser movimiento, velocidad. Los videos-sensores no poseen una forma de comunicación con el sistema Video Vigilancia Suria en Qt por lo que hace que los análisis inteligentes de videos sean desaprovechados.

Teniendo en cuenta lo anteriormente descrito se ha identificado como **problema a resolver** que: Los videos-sensores desarrollados por el sistema de Video Vigilancia Suria para análisis inteligente de videos no pueden ser aprovechados por el sistema Video Vigilancia Suria en Qt.

Por tanto se define como **objeto de estudio** de la investigación: la integración de componentes para el proyecto Video Vigilancia Suria con el módulo Análisis (Analytic), enmarcándose en el **campo de acción** la creación del sistema Analytic para el sistema Video Vigilancia Suria en Qt, definiendo como **objetivo general**: Desarrollar el módulo Analytic para el sistema Video Vigilancia Suria en Qt.

Se defiende la idea de que: Con el desarrollo de un módulo que aproveche los videos-sensores desarrollados, se podrá dotar de un sistema capaz de realizar análisis inteligente en videos.

Para el cumplimiento del objetivo especificado se trazaron las siguientes tareas de investigación:

1. Realizar un estudio de los sistemas de video vigilancia existentes en la actualidad que permitan realizar análisis inteligente en videos.
2. Realizar el levantamiento de requisitos del módulo Análisis.
3. Obtener los modelos del módulo Análisis.
4. Caracterizar la arquitectura basada en componentes.
5. Caracterizar la arquitectura Pizarra en su variante de tablero de control.
6. Realizar el diseño del módulo Análisis.
7. Implementar el módulo de Análisis.
8. Realizar pruebas de caja blanca al módulo Análisis.

Una vez concluida la investigación se espera obtener como resultado un módulo que permita aprovechar los videos-sensores desarrollados.

Para la presente investigación se tendrá en cuenta, varios métodos científicos:

- **Métodos teóricos:** Con la utilización de estos es posible estudiar las características del flujo de trabajo para la integración de un software que no son observadas directamente.
  - *Análisis Histórico–lógico**:*** Mediante este método se puede valorar la evolución de la integración de video sensores desarrollados con el Sistema de Video Vigilancia Suria Qt e investigar la información que se tiene hasta el momento del estado del arte de las métricas utilizadas, también permitirá determinar los principales conceptos relacionados con integración de software.
  - *Método analítico sintético:* Este método se utilizará con el objetivo de realizar un estudio detallado de las métricas propuestas para la creación del software que permitirá aprovechar los video sensores desarrollados para el Sistema de Video Vigilancia Suria.
  - *Método de Modelación:* este método se empleará para la confección de los diagramas del sistema.
- **Métodos empíricos:** Estos métodos son los que permiten hacer una retroalimentación de trabajos realizados con anterioridad sobre un tema en específico, en este caso las métricas propuestas para la integración del software que permitirá aprovechar por parte del Gestor Qt los video sensores desarrollados para el Sistema de Video Vigilancia Suria.
  - *Método de Observación:* Se utilizó en toda la investigación, pues se recoge información de los aspectos tratados en la tesis desde el punto de vista de otros autores así como sus definiciones y resultados para permitir realizar un análisis del estado del arte en ese campo.

El contenido de este trabajo se encuentra estructurado en cuatro capítulos, los que se definen de la siguiente manera:

Capítulo 1. En este capítulo se definirán algunos conceptos generales importantes para la investigación del módulo Análisis. Además se realizará un estudio del estado del arte de los sistemas de video vigilancia existentes en la actualidad que permitan realizar análisis inteligente en videos.

Capítulo 2. En este capítulo se identificarán las herramientas que darán soporte al proceso de análisis, diseño e implementación de la solución. Se seleccionarán las tecnologías a desarrollar para la aplicación y se definirá la metodología de desarrollo de software más apropiada.

Capítulo 3. En este capítulo se realizará el levantamiento de requisitos del módulo Análisis mediante el cual se obtendrán los modelos del sistema y se realizará el análisis. Se caracterizarán las arquitecturas para la creación de un sistema basado en componentes y arquitectura base en forma de Pizarra, donde se identificarán las principales particularidades de cada una de las arquitecturas escogidas. Se investigarán los patrones de diseño para así escoger los más apropiados para el diseño del módulo.

Capítulo 4. En este capítulo se implementará el módulo Análisis donde se le harán pruebas de caja blanca al módulo con el objetivo de validar la correcta implementación del software.

CAPÍTULO 1

Fundamentación Teórica.

1.1 Introducción

En el capítulo se abordarán los temas relacionados con la fundamentación teórica de la investigación, en el mismo se describirán los conceptos más significativos asociados al dominio del problema, además se abarcará en el estado del arte las soluciones existentes con sus diferencias. Se realizará una propuesta del software Análisis a desarrollar teniendo en cuenta las ventajas y características del mismo.

1.2 Conceptos asociados al dominio del problema

- **Video:** El video es una tecnología utilizada para capturar, grabar, procesar, transmitir y reproducir una secuencia de imágenes representativas de una escena que se encuentra en movimiento**.** El término, que proviene del latín "ver" (1)**.**

  El video es una tecnología que fue desarrollada por primera vez para los sistemas de televisión, pero ha derivado en muchos formatos para permitir la grabación de video de los consumidores y que además pueda ser visto a través de Internet. El video se puede grabar y transmitir en diversos medios físicos: en cinta magnética cuando las cámaras de video registran como PAL, SECAM o NTSC señales analógicas, o cuando las cámaras graban en medios digitales como MPEG-4 o DVD (MPEG-2).
- **Video Vigilancia:** Es una técnica de control y protección cada día más extendida tanto en aplicaciones de seguridad pública, como en ámbitos empresariales(2)**.**

  Con esta técnica se han creado sistemas de video vigilancia que a gran escala tienen una eficacia limitada, esto se debe a que para los operadores es difícil ver numerosos monitores y hacer un seguimiento de cada incidencia. Con las soluciones de video inteligente, menos operadores pueden supervisar instalaciones aún más grandes. El sistema de video inteligente apoya e informa a los operadores.

Sensores: Palabra formada sobre el latín *sentĭo*, sentir. Dispositivo que detecta una determinada acción externa, temperatura, presión (3)**.**

Los sensores han ayudado no solo a medir con mayor exactitud las magnitudes, sino a poder operar con dichas medidas. Los videos-sensores ofrecen nuevas posibilidades para explorar y entender el mundo en el que vivimos. Particularmente, un video sensor es un software de aplicación, que interpreta las imágenes a través de algoritmos que se ejecutan.

- **Componente de Software:** Un componente es una parte no trivial, casi independiente y reemplazable de un sistema que cumple una función dentro del contexto de una arquitectura bien definida. Un componente cumple con un conjunto de interfaces y provee la realización física de ellas (4)**.**

  Un componente de software se puede reutilizar a través de aplicaciones de software e intercambiable donde el mismo puede poseer varios componentes. Un componente de software puede ser: un diccionario de datos, un conjunto de reglas, símbolos gráficos. Un componente de software puede ser una unidad ejecutable que puede ser implantada independientemente.

1.3 Estado del Arte del Sistema Análisis

El mercado del software posee diversos sistemas de video vigilancia donde los hay desde los más simples que tienen una cámara y un grabador hasta los más complejos que poseen video sensores de movimiento, alarmas automáticas y detección de incidencias. En el ámbito internacional existen varias empresas dedicadas a crear sistemas de video vigilancia que realicen análisis inteligente.

SECUROS La empresa SecurOS posee un sistema de gestión y análisis de videos novedoso en el mercado de los Sistemas Inteligentes de Seguridad (ISS).

ISS es el líder global en gestión de video, con más de ochenta mil instalaciones en cincuenta y tres países, comandando más de un millón de cámaras y SecurOS es una solución de altas prestaciones y funcionalidades avanzadas dentro de ISS. SecurOS es el núcleo de administración de una completa topografía de vigilancia y seguridad. SecurOS puede administrar y supervisar una cantidad ilimitada de cámaras y dispositivos, aplicar capacidades de inteligencia forense e integrar una gran variedad de subsistemas distintos en una sola interfaz centralizada de comando y control (5)**.** La plataforma

SecurOS es ideal para grandes aplicaciones de misión crítica que incluye a cientos o miles de cámaras, sensores y sistemas de control unificados en una sola red. Dicho software es propietario**.**

Esta empresa SecurOS al ser una solución de altas prestaciones dentro de los Sistemas Inteligentes de Seguridad (ISS) posee algoritmos de análisis de video que proporcionan un nivel de inteligencia alto, donde su arquitectura abierta le permite trabajar con cualquier sistema ya sea analógico o IP. Dicha solución servirá de objeto de estudio pues tienes capacidades de análisis de imagen, como el reconocimiento de matrículas y reconocimiento facial donde cada uno de ellos serán creados futuramente. Además los análisis que realizan son de imágenes y el sistema que se desea crear es de análisis inteligente a través de flujos de video. El sistema que tiene esta empresa está creado bajo una licencia propietaria y el sistema que se desea crear se espera que se base en una licencia libre.

Video inteligente Axis

Axis ofrece diversas aplicaciones de video inteligente que apoyan a los operadores de seguridad en su trabajo diario al resolver los retos de video vigilancia del mundo real. Además, los socios de desarrollo de Axis cuentan con una amplia gama de video inteligente que funciona con los productos de video en red de Axis (6)**.**

Axis en la actualidad graba cantidades ingentes de video. Las aplicaciones abarcan desde el análisis, por ejemplo detección de movimiento por video y la detección de audio, a sistemas más avanzados que incluyen detección de manipulación de la cámara, recuento de personas, vallas virtuales y reconocimiento de matrículas de vehículos. A las aplicaciones que efectúan estos análisis también se las conoce como Análisis de contenido de video o Análisis de video. Estas aplicaciones de video inteligente que funcionan con los productos de video en red de Axis son propietarias.

Axis es una empresa con una amplia gama de aplicaciones que abarcan desde análisis de video inteligente como: detección de movimiento por video y la detección de audio explicados anteriormente, la misma está creada bajo una arquitectura que solo diseñan y definen las especificaciones de los sistemas de seguridad basados en IP. Dicha empresa aportará conocimientos para la creación del sistema propuesto pues cumple con las características fundamentales del sistema, pues Video Vigilancia Suria está diseñado para trabajar con un sistema de seguridad basado en IP y necesita crear análisis inteligente de videos. Dichas aplicaciones pertenecientes a esta empresa no constituyen

la solución directa al problema planteado pues posee una limitante fundamental y es que dichas aplicaciones están creadas bajo licencias propietarias y el sistema de Análisis que se desea crear se espera que se base bajo una licencia libre.

La línea de productos Siqura promueve el desarrollo de soluciones abiertas estándar, Optelecom-NKF ha unido fuerzas con la ISS para integrar sus cámaras de cajón Siqura y H. 264 de alta velocidad de red domo PTZ[1] en la suite de soluciones de ISS de tecnología de video de seguridad. Servidor Optelecom-NKF video H. 264[2] es ahora también compatible con la oferta de ISS.

Como socios de la alianza, Optelecom-NKF y los clientes ISS ofrecen la posibilidad de combinar las soluciones completas e inteligentes de cada empresa en sus sistemas a fin de crear la mejor solución de vigilancia para su aplicación específica.

Esta línea de productos Siqura es la unión de los sistemas ISS y Optelecom-NKF, donde ambos unidos están creando un robusto sistema de vigilancia. Los productos creados bajo esta línea poseen características específicas que servirán de apoyo para el estudio que se realice pues sus soluciones están creadas para trabajar con cámaras IP, para realizar análisis de video y análisis para aplicaciones específicas al aire libre**.** El software de análisis que se desea crear también trabajará con cámaras IP, además realizará análisis de video, pero los sistemas creados bajo esta línea Siqura no resuelve el problema planteado pues los mismos están creados bajo licencias propietaria y el sistema de análisis que se creará se basará bajo una licencia libre (7).

---

[1] **cámara de red Domo PTZ:** Es una clasificación de cámaras de red, donde pueden cubrir una amplia área al permitir una mayor flexibilidad en las funciones de movimiento horizontal, vertical y zoom. Así mismo, permiten un movimiento horizontal continuo de 360 grados y un movimiento vertical de normalmente 180 grados.

[2] **H. 264:** Es un estándar, un formato digital avanzado que permite codificar eficientemente video de alta definición

El proyecto Video Vigilancia Suria constituye una plataforma para la video vigilancia. Durante el desarrollo del producto, el equipo de trabajo ha centrado sus esfuerzos en lograr un sistema flexible y escalable para conseguir una mayor adecuación a las necesidades de los clientes. Destacándose dentro de sus principales ventajas la capacidad de operar con cámaras IP de distintos fabricantes y el análisis en tiempo real de los flujos de video.

El sistema Suria se encuentra estructurado en cinco subsistemas que interactúan entre sí para garantizar la monitorización de cualquier entorno de manera eficiente. El producto está diseñado además con la particularidad de que cada subsistema funciona de manera autónoma, ajustándose de este modo al alcance de la solución a instalar.

El Visor es la estación de visualización de la aplicación, puede haber varias instancias de esta estación corriendo en los dominios físicos del sistema. Tiene la capacidad de reflejar todo el aspecto organizativo con que el sistema maneja las cámaras internamente. También permite manipularlas en medida de las capacidades de cada una. Puede trabajar en manera cooperativa con el Recuperador, tras previa coordinación del Gestor, para la recuperación de video almacenado de un grupo determinado de cámaras.

El Grabador es el módulo encargado de almacenar video obtenido de las cámaras, bajo petición de un cliente determinado o por configuración, que puede ser por horarios determinados o por la ocurrencia de algún evento.

El Recuperador es el encargado de recuperar los datos almacenados por el Grabador, hacer búsquedas en estos, y servir los resultados al Visor.

El Módulo Análisis (Analytic) desarrollado en C#, es el encargado del procesamiento de los flujos de video, este sub-sistema está integrado por varios videos-sensores, entre ellos: detección de movimiento, perimetrado virtual, detección de objetos abandonados y conteo de personas.

Siendo fiel a la arquitectura propuesta, en aras de alcanzar un sistema completamente distribuido y teniendo en cuenta que debido a la biblioteca utilizada la implementación de los videos-sensores se realizó en visual C++.

La comunicación entre Suria Analytic y el Gestor se realiza mediante la tecnología .NET Remoting. Esta permite a los objetos en diferentes dominios de aplicación interactuar unos con otros. La verdadera fortaleza de Remoting es permitir la comunicación entre los objetos cuando sus ámbitos de aplicación se separan a través de la red. En este caso, la comunicación remota maneja de manera transparente los detalles relacionados con la comunicación de red (8). En la figura 1 se puede observar el funcionamiento de la tecnología .NET Remoting.

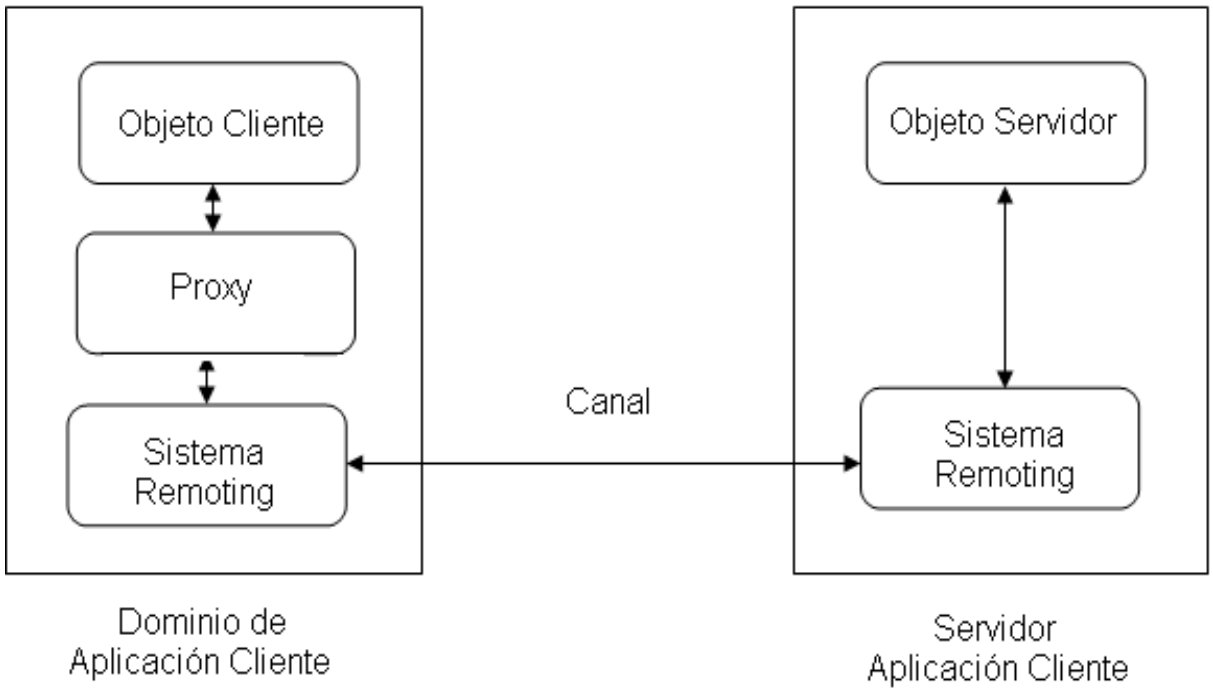

Figura 1: .NET Remoting

Cuando se realiza una petición de ejecución de un video-sensor al Gestor, se chequea la disponibilidad de los recursos (CPU y RAM) en las instancias de Suria Analytic del sistema. Si es posible realizar el procesamiento, entonces se le asigna la petición a la instancia con menos carga de procesamiento, especificándole cámara y el tipo de video sensor a ejecutar. En la figura 2 se observa la interacción Suria Analytic y el Gestor.

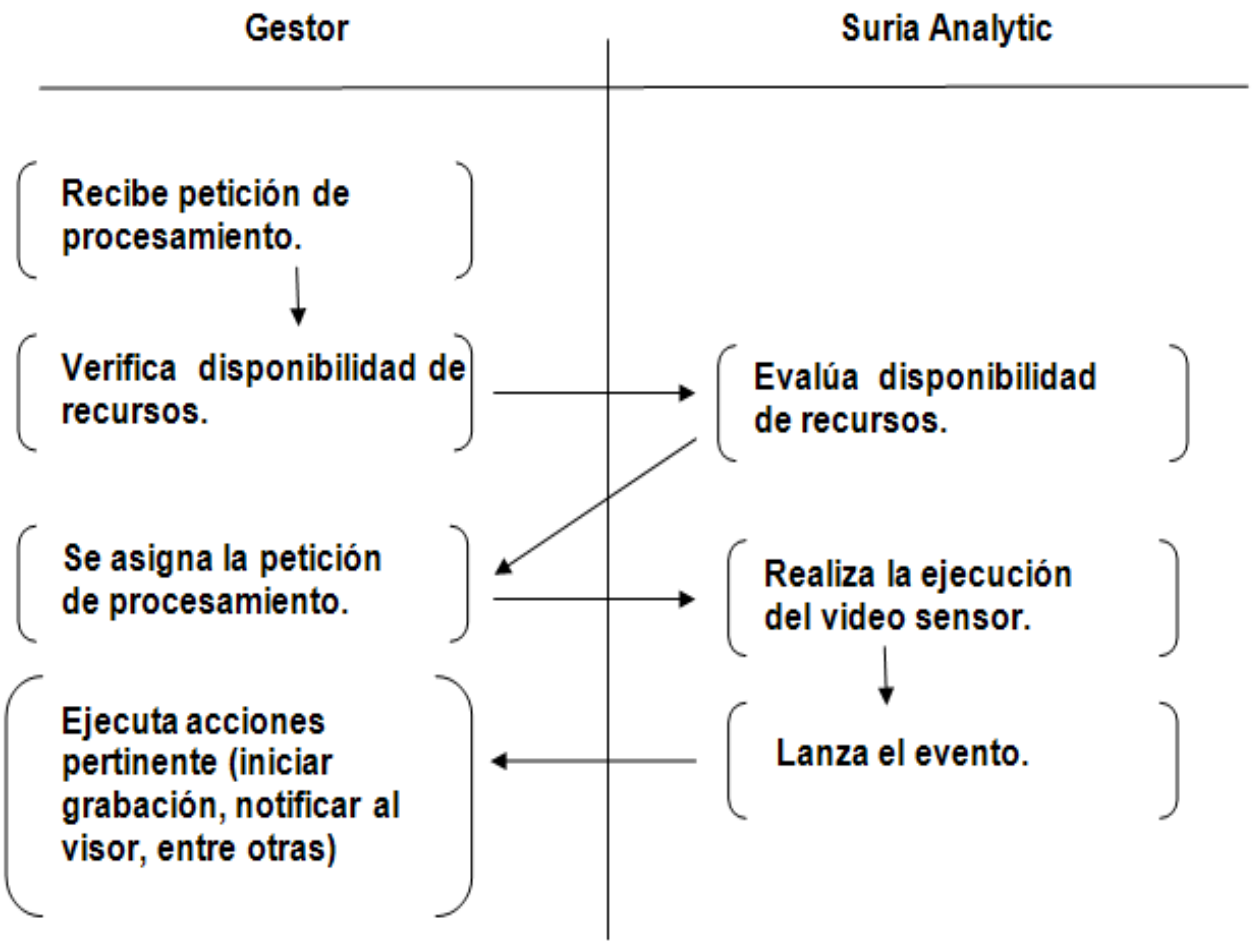

Figura 2: Interacción Suria Analytic y el Gestor.

El subsistema Analytic será capaz de manipular diversos tipos de video sensores. Cada video sensor en dependencia de la función para la que fue diseñado, funcionará de manera distinta, por lo cual se debe de buscar un mecanismo para dar solución a este problema. Para esto el subsistema soportará las nuevas funcionalidades de procesamientos de videos a través de plugins[3], que serán desarrollados de manera específica para cada video sensor, comportándose de manera semejante, de forma tal que el subsistema pueda utilizar cualquiera de estos plugins de la misma forma, no importa el video sensor que sea, manteniéndose así la sencillez del diseño y la flexibilidad del sistema (9). En la figura 3 se observan los componentes de Suria Analytic.

3 Un plugins es un módulo de hardware o software que añade una característica o un servicio específico a un sistema más grande.

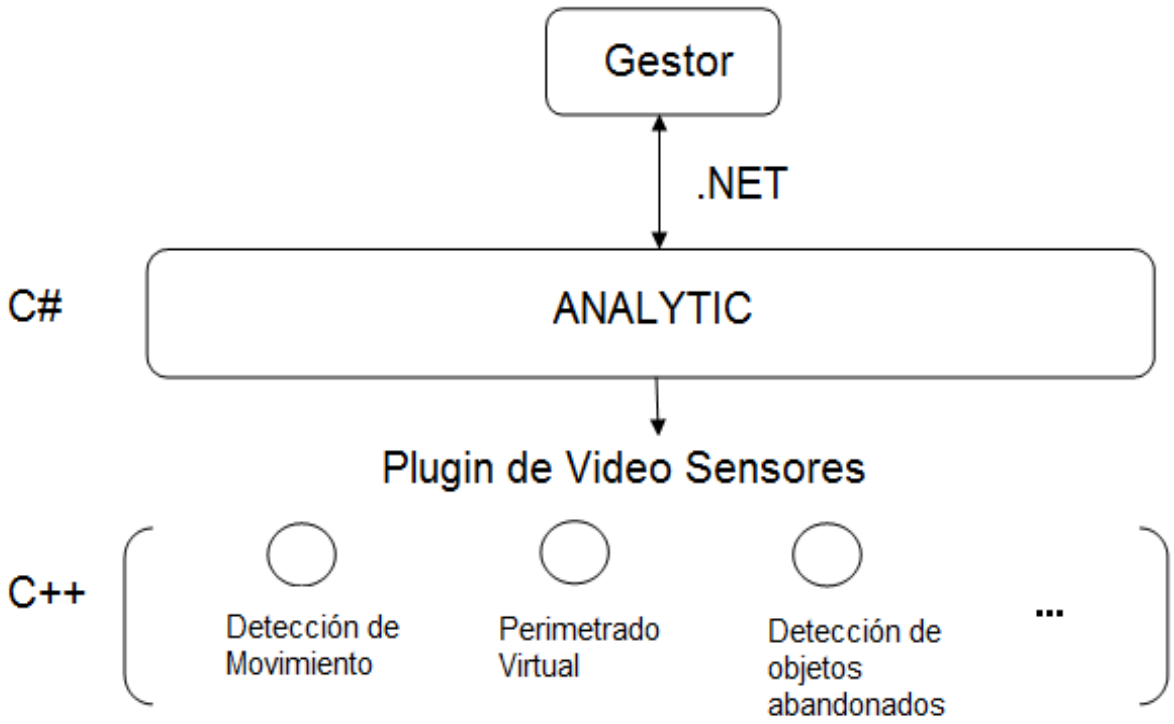

Figura 3: Componentes de Suria Analytic.

Cuando el subsistema recibe una petición de procesamiento se crea una instancia del video sensor que responde a dicha petición con los parámetros requeridos en un proceso independiente.

Internamente el plugin contiene una clase "no manejada" en C++ que se encarga del procesamiento, y un "wrapper[4]", que no es más que una clase manejada de Visual C++ que encapsula la clase "no manejada". Esto permite, después en un entorno manejado, hacer referencia al plugin, construyendo un objeto de la clase manejada y para realizar el procesamiento en la no manejada. Permitiendo el procesamiento a bajo nivel utilizando OpenCV[5] con C++ garantizando el rendimiento que requiere este tipo de procesamiento, sin perder las ventajas que brinda las tecnologías del framework .NET

De esta manera se logra la integración de los videos-sensores implementados en forma de plugin al

---

4 Wrapper: (Wrapper en inglés es Envoltura) Función de contenedor, una función cuyo objetivo principal es llamar a una segunda función.

5 OpenCV: OpenCV (Open Source Computer Vision) es una biblioteca de funciones de procesamiento de imágenes para la visión de computadora en tiempo real.

sistema de Video Vigilancia Suria; convirtiéndolo así, en un sistema inteligente capaz de detectar movimiento ya sea de manera general o en un perímetro señalado, detectar objetos abandonados y el conteo de personas. Siendo fiel a la arquitectura del sistema Suria, queda abierta la posibilidad de implementar un sin número de videos-sensores, los cuales mediante la solución propuesta serán fáciles de integrar.

La información que provee el sistema Video Vigilancia Suria ayudará para el estudio de la creación del sistema a desarrollar porque posee casi todas las características necesarias para crear el sistema “Módulo de Análisis para el sistema Video Vigilancia Suria en Qt” pues coincide la arquitectura a utilizar y la lógica de comunicación entre el Gestor y el Analytic. Este sistema no establece la solución inmediata al problema planteado pues el módulo Analytic está creado bajo el framework de desarrollo .NET el cual es propietario y el nuevo Analytic que se desea crear se espera que utilice el framework de desarrollo Qt que se basa bajo una licencia libre, además la comunicación entre Suria Analytic y el Gestor se realiza mediante la tecnología .Net Remoting y la tecnología de comunicación que se espera utilizar para este sistema con el Gestor es XMLRPC.

1.4 Conclusiones

Los sistemas de Video Vigilancia hoy en día constituyen un eslabón principal en la seguridad y los sistemas que se le integran hacen más eficiente su funcionamiento. Su capacidad para grabar, detectar y analizar hace que los problemas de seguridad tengan soluciones a corto plazo.

En el presente capítulo se vieron las principales definiciones de la investigación para así lograr un mejor entendimiento del problema a resolver: en el estado del arte se abarcaron las soluciones que existen en el mundo, determinando que aunque no constituyan la solución directa al problema planteado, cada una de ella sirvieron como objeto de estudio. Se determinó hacer un sistema de Análisis que utilice una licencia libre.

CAPÍTULO 2

Características del Sistema.

2.1 Introducción

En el presente capítulo se abordarán los temas relacionados con las tendencias y tecnologías actuales a desarrollar, en el mismo se determinará la metodología de desarrollo de software más adecuada, el lenguaje de modelación y programación con el que se trabajará, así como el entorno de desarrollo y marco de trabajo que se utilizará para realizar el módulo de Análisis.

2.2 Metodología de desarrollo de software

Las Metodologías de Desarrollo de Software surgen ante la necesidad de utilizar una serie de procedimientos, técnicas, herramientas y soporte documental a la hora de desarrollar un producto de software.

Las metodologías se clasifican en metodologías ágiles y pesadas donde las ágiles están orientadas a la interacción con el cliente y el desarrollo incremental del software, mostrando versiones parcialmente funcionales del software al cliente en intervalos cortos de tiempo, para que pueda evaluar y sugerir cambios en el producto según se va desarrollando. Las metodologías pesadas están orientadas al control de los procesos, estableciendo rigurosamente las actividades a desarrollar, herramientas a utilizar y notaciones que se usarán.

2.2.1 Rational Unified Process (RUP)

Una de las metodologías pesadas más conocidas y utilizadas es la Metodología RUP (Rational Unified Process) que divide el desarrollo en cuatro fases que definen su ciclo de vida:

- **Inicio:** El objetivo es determinar la visión del proyecto y definir lo que se desea realizar.
- **Elaboración:** Etapa en la que se determina la arquitectura óptima del proyecto.
- **Construcción:** Se obtiene la capacidad operacional inicial.
- **Transmisión:** Obtener el producto acabado y definido (10).

RUP se basa en la realización de ciclos que delimita la vida del proyecto, este ciclo está compuesto por cuatro fases: Inicio, Elaboración, Construcción y Transición y cada una de ellas se divide en iteraciones, a su vez cada iteración trabaja en un número de disciplinas, entre ellas se encuentran: Modelado del negocio, Requisitos, Análisis y Diseño, Implementación, Pruebas.

Proceso Unificado de Desarrollo es un proceso de desarrollo de software que se adapta según las necesidades de los desarrolladores y del cliente, debido a que la comunicación entre estos debe ser fluida, permitiendo llegar a un acuerdo para evitar desacuerdos futuros, además integra el Lenguaje Unificado de Modelado (UML) y está basado en componentes interconectados a través de una interfaz bien definida.

RUP propone un proceso de desarrollo iterativo e incremental. De manera que el trabajo se divida en subproyectos o en partes más pequeñas donde cada una de estas representa una iteración que propicia un incremento en el software; las iteraciones hacen referencia a los pasos del flujo de trabajo y el incremento al crecimiento del producto. Cada iteración debe ser controlada para disminuir el riesgo de un incremento, para que el software sea concluido en el tiempo planificado, se trabaje con eficiencia donde el esfuerzo tenga un resultado a corto plazo y no indefinidamente.

La metodología RUP tiene 6 principios básicos: adaptación del proceso, balancear prioridades, colaboración entre equipos, demostrar valor iterativamente, elevar el nivel de abstracción, y enfocarse en la calidad. En la figura 4 se observan las disciplinas y fases de la metodología de desarrollo RUP.

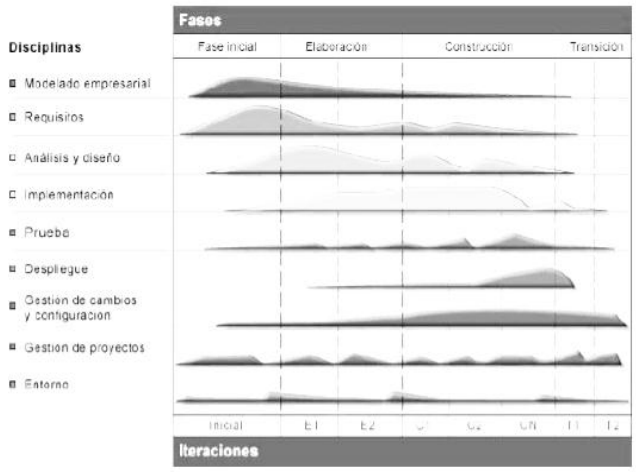

Figura 4: Disciplina de desarrollo de RUP

2.3 Lenguaje de modelación

Lenguaje de modelación surge no sólo para comunicar las ideas a otros desarrolladores sino también para servir de apoyo en los procesos de análisis de un problema.

Prescribe un conjunto de notaciones y diagramas estándar para modelar sistemas orientados a objetos, y describe la semántica esencial de lo que estos diagramas y símbolos significan (11).

2.3.1 Unified Modeling Lenguaje (UML)

El lenguaje UML tiene una notación gráfica muy expresiva que permite representar en mayor o menor medida todas las fases de un proyecto informático: desde el análisis con los casos de uso, el diseño con los diagramas de clases, objetos hasta la implementación y configuración con los diagramas de despliegue.

Este modelado visual es independiente del lenguaje de implementación, de tal forma que los diseños realizados usando UML se puedan implementar en cualquier lenguaje que soporte las posibilidades de UML (principalmente lenguajes orientados a objetos).

UML es ante todo un lenguaje. Un lenguaje proporciona un vocabulario y reglas para permitir una comunicación. En este caso, este lenguaje se centra en la representación gráfica de un sistema. Este lenguaje nos indica cómo crear y leer los modelos, pero no dice cómo crearlos (12).

UML es además un método formal de modelado. Esto aporta las siguientes ventajas:

- Mayor rigor en la especificación.
- Permite realizar una verificación y validación del modelo realizado.
- Se pueden automatizar determinados procesos y permite generar código a partir de los modelos y a la inversa (a partir del código fuente generar los modelos). Esto permite que el modelo y el código estén actualizados, con lo que siempre se puede mantener la visión en el diseño, de más alto nivel, de la estructura de un proyecto.

2.4 Herramienta CASE

Se puede definir a las herramientas CASE como un conjunto de programas y ayudas que dan asistencia a los analistas, ingenieros de software y desarrolladores, durante todos los pasos del Ciclo de Vida de desarrollo de un Software. Como es sabido, los estados en el Ciclo de Vida de desarrollo de un Software son: Investigación Preliminar, Análisis, Diseño, Implementación e Instalación (13).

2.4.1 Visual Paradigm

Visual Paradigm para UML es una herramienta UML profesional que soporta el ciclo de vida completo del desarrollo de software: análisis y diseño orientados a objetos, construcción, pruebas y despliegue. El software de modelado UML ayuda a una más rápida construcción de aplicaciones de calidad. Permite dibujar todos los tipos de diagramas de clases, código inverso, generar código desde diagramas y generar documentación. La herramienta UML CASE también proporciona abundantes tutoriales de UML, demostraciones interactivas de UML y proyectos UML.

Visual Paradigm se caracteriza por ser un sistema multiplataforma, es una aplicación de software libre, distribuido bajo una licencia comercial y gratuita. Su diseño se centra en los casos de usos enfocados al negocio que generan un software de mayor calidad además de ser capaz de ofrecer ingeniería directa e inversa (código a modelo, código a diagrama). Por otra parte tiene un modelo y código que permanecen sincronizados en todo el ciclo de desarrollo así como la generación de código para Java y exportación como HTML.

Se seleccionó Visual Paradigm como herramienta de modelado por sus principales características:

- Disponibilidad de múltiples versiones, para cada necesidad.
- Disponibilidad de integrarse en los principales IDEs.
- Disponibilidad en múltiples plataformas.
- Uso de un lenguaje estándar común a todo el equipo de desarrollo que facilita la comunicación.
- Entorno de creación de diagramas para UML (14).

2.5 Lenguaje de Programación

Un lenguaje de programación es aquel elemento dentro de la informática que nos permite crear programas mediante un conjunto de instrucciones, operadores y reglas de sintaxis; que pone a

disposición del programador para que este pueda comunicarse con los dispositivos hardware y software existentes (15).

2.5.1 C++

C++ es un lenguaje de programación basado en C donde retiene los recursos de bajo nivel y la eficiencia de C, soporta conceptos de orientación a objetos. C++ es un lenguaje imperativo, orientado a objetos, que exige del programador un completo cambio de mentalidad. Mantiene una considerable potencia para programación a bajo nivel aunque se le han añadido elementos que le permiten también un estilo de programación con alto nivel de abstracción. C++ es considerado un lenguaje híbrido ya que en él coexiste la programación estructurada y los conceptos de programación orientada a objetos. Se le ha incorporado nuevos tipos de datos, clases, plantillas, mecanismo de excepciones, sistema de espacios de nombres, funciones de expansión en línea, sobrecarga de operadores, referencias, operadores para manejo de memoria persistente, y algunas utilidades adicionales de librería.

Se ha escogido a C++ como lenguaje de programación por su rapidez, portabilidad, documentación en el procesamiento de imágenes y por política del proyecto Video Vigilancia Suria. Es un lenguaje versátil, potente y general. C++ es un lenguaje de propósito general, por lo que se puede emplear para resolver cualquier tipo de problema. Está estandarizado y un mismo código fuente se puede compilar en diversas plataformas. Es un lenguaje de programación muy utilizado para el desarrollo de aplicaciones de escritorio, posibilita orientar la programación a objetos permitiéndole al programador diseñar aplicaciones desde un punto de vista más cercano a la vida real; permite la reutilización del código de una forma más lógica y productiva.

2.6 Entorno de desarrollo Integrado (IDE)

Un Entorno Integrado de Desarrollo (IDE) es un entorno de programación o programa informático compuesto por un conjunto de herramientas que utilizan los programadores para generar código. Un IDE puede estar creado para soportar un sólo lenguaje de programación o varios. Las herramientas que oficialmente componen un IDE son: un editor de código que generalmente tiene resaltado de sintaxis, completamiento de código y ayuda sensible al contexto, no necesariamente debe tener un compilador, un depurador, un constructor de interfaz gráfica o un sistema de control de versiones aunque existan IDEs que tengan todas estas herramientas. Tiene como objetivo acortar la brecha entre el usuario y el lenguaje de programación proporcionando un marco de trabajo amigable.

2.6.1 Qt Creator

Qt Creator es un IDE multiplataforma para la realización de aplicaciones con las bibliotecas Qt, creadas por la compañía noruega Trolltech (actualmente QT Software). En marzo del 2009 liberó su primera versión estable, teniendo gran aceptación. Puede ser ejecutado en los sistemas operativos desde Windows98 hasta Windows XP y Vista.

Proporciona características que ayudan a los nuevos usuarios de Qt a aprender y comenzar a desarrollar rápidamente, también aumenta la productividad de los desarrolladores con experiencia en Qt.

Qt Creator se ejecuta en Windows, Linux/X11 y Mac OS X sistemas operativos de escritorio, y permite a los desarrolladores crear aplicaciones para escritorio. Utiliza el estándar C++.

Se decidió utilizar Qt Creator para el desarrollo del sistema, porque puede elaborar aplicaciones de escritorio y dan soporte al lenguaje de programación C++ (seleccionado para el desarrollo). Ofrece gran compatibilidad con varios sistemas operativos donde permite el desarrollo de interfaces amigables y alto rendimiento con C++.

2.7 Framework

2.7.1 Qt

Qt es un Marco de Trabajo (framework) de desarrollo de aplicaciones multiplataforma que viene acompañado de un conjunto de herramientas para facilitar su uso. Proporciona una excelente compatibilidad para desarrollar aplicaciones para:

- Microsoft Windows: 98, NT 4. 0, ME, 2000, y XP
- Unix/X11 Linux, Sun Solaris, HP-UX, HP Tru64 UNIX, IBM AIX, SGI IRIX y muchas otras
- Mac OS X Mac OS X 10. 3+
- Plataformas Linux con soporte frame buffer

Las librerías Qt no están sólo disponibles para C++, sino también ofrece soluciones para utilizarse con otros lenguajes tales como:

- Java (QJambi)

- Python (PyQt)
- Ruby
- JavaScript (módulo QtScript)
- PHP

Se escogió este framework Qt porque tiene la característica de presentar el código fuente disponible, posee gran documentación y calidad en cuanto a soporte, por política del proyecto Video Vigilancia, ha sido adaptado a las características de los entornos de desarrollo actuales y está pensado para sistemas operativos recientes.

2.8 Conclusiones

Para la obtención de los objetivos trazados se han escogido herramientas acordes a las necesidades del sistema. En el presente capítulo se definió la metodología de desarrollo de software a emplear para el módulo Análisis, la cual se basa en la realización de ciclos que delimita la vida del proyecto. Se determinó hacer un sistema de Análisis que utilice como entorno de desarrollo integrado Qt Creator y como marco de trabajo Qt para así lograr desarrollar una aplicación multiplataforma que se ajuste a las necesidades de los desarrolladores. Se escogió el lenguaje de modelado y de programación según las características del software a realizar.

CAPÍTULO 3

Análisis y Diseño del Sistema.

3.1 Introducción

En este capítulo se construirá la propuesta de solución para llevar a cabo la elaboración del módulo Análisis. Se determinarán los requisitos funcionales y no funcionales del sistema. Esta propuesta incluye los modelos de análisis y diseño, los cuales aclararán las bases para comenzar con la implementación de la propuesta de solución del sistema. Esto se logrará mediante los diferentes diagramas de clase e interacción que incluyen dichos modelos. Se explicarán los patrones de diseño empleados para garantizar la calidad requerida, al igual que la viabilidad. Además se definirá la arquitectura de software a utilizar para la correcta creación del módulo Análisis.

3.2 Entorno donde trabajará el sistema

3.2.1 Modelo de Dominio

El Modelo de Dominio muestra las clases conceptuales significativas en el dominio del problema en cuestión, abarcando los ejemplares más importantes de objetos existentes o los eventos que suceden en el entorno donde se desempeñará el sistema.

El modelo de dominio se describe mediante diagramas UML (especialmente mediante diagrama de clases). Estos diagramas muestran las clases del dominio y cómo se relacionan unas con otras mediante asociaciones. Proporciona a los usuarios, clientes, desarrolladores y otros interesados, un vocabulario común.

Teniendo en cuenta que no se tienen bien definidos los procesos del negocio se realizará una modelación del dominio y se procederá a explicar cada uno de los conceptos que forman parte del mismo. Todo ello para tener una mejor comprensión de la estructura y dinámica de la organización, los problemas actuales dentro de esta e identificar las mejoras potenciales. En la figura 5 se puede observar el diagrama de entidad del dominio.

3.2.2 Diagrama de clases del Modelo de Dominio

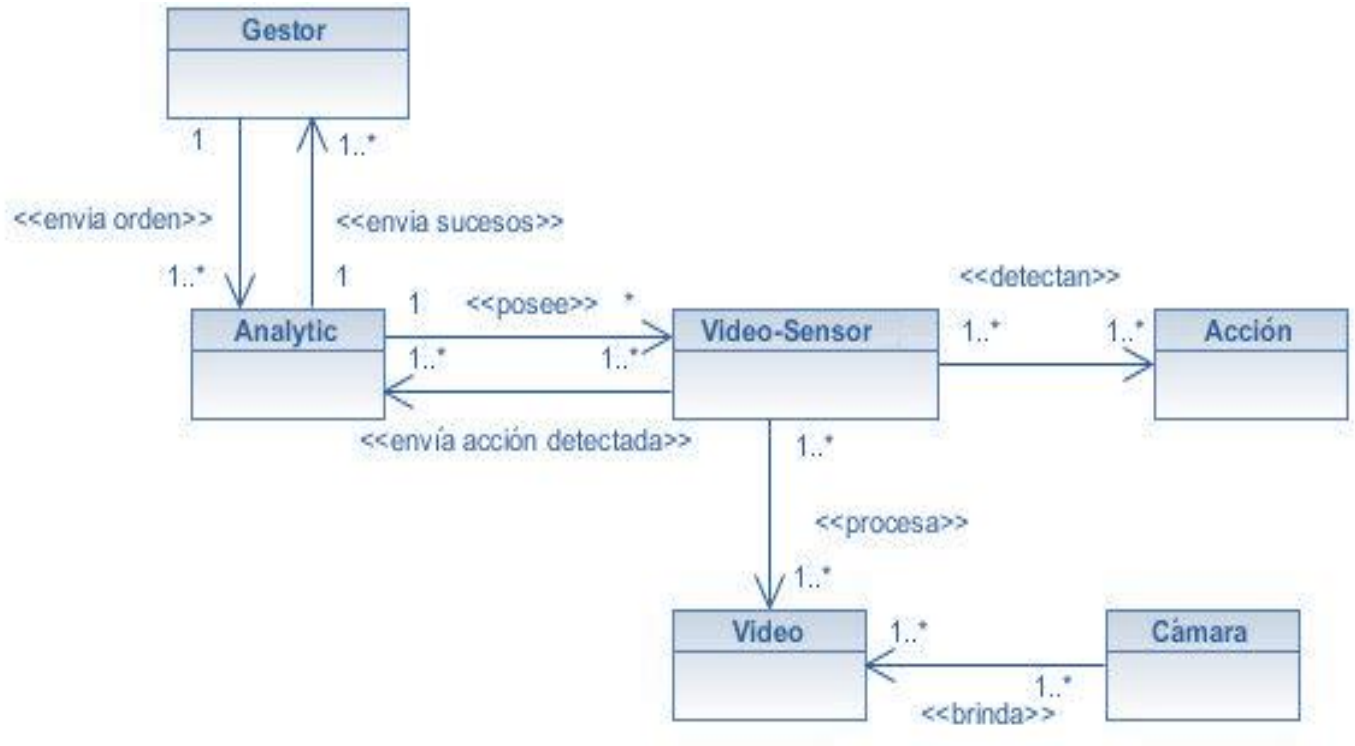

Figura 5: Diagrama de entidades del dominio.

3.2.3 Descripción del Modelo de Dominio

El gestor le envía una orden al Analytic el cual posee video sensores que detectan una determinada acción ya sea de detección de movimiento o de perimetrado virtual y procesa el video que es brindado por la cámara, luego el Analytic le envía al gestor los sucesos detectados por los videos-sensores.

3.2.4 Glosario de Términos del Dominio

Clases	Descripción
Gestor	Es el sistema central del proyecto Video Vigilancia Suria encargado de la intercomunicación entre los módulos.
Analytic	Es un sistema que gestiona el procesamiento inteligente de video, a través de video sensores adicionados a este a

	través de plugins.
Video-Sensor	Un video-sensor describe una técnica de análisis de imagen digital. El mismo es un software de aplicación, que interpreta las imágenes y utiliza algoritmos de identificación de objetos.
Acción	La acción es: Movimiento o intrusión.
Video	Archivo que almacena los flujos de video de una cámara.
Cámara	Dispositivo de captura de imagen y video.

Tabla 1: Glosario de Términos.

3.3 Requerimientos Funcionales

Los Requerimientos Funcionales son capacidades que el sistema debe cumplir, suficientemente buenas como para llegar a un acuerdo entre los clientes sobre qué debe y qué no debe hacer el sistema” (16).

A continuación se muestran los requerimientos funcionales del sistema:

No. Requisitos Funcionales	**Requisitos Funcionales**
RF 1	Configurar el Sistema
RF 2	Cargar Configuración
RF 3	Cargar Video-Sensor para iniciar procesamiento
RF 4	Establecer Comunicación con el Gestor
RF 5	Notificar Sucesos al Gestor
RF 6	Detener Procesamientos.

Tabla 2: Requisitos Funcionales.

3.4 Requerimientos No Funcionales

Según Roger Pressman, "Los requerimientos no funcionales son propiedades o cualidades que el producto debe tener, además son aspectos importantes que el producto debe cumplir para lograr un producto atractivo, usable, rápido o confiable" (17). A continuación se enuncian, separados en categorías, los diferentes requisitos no funcionales que el componente debe satisfacer.
A continuación se muestran los requerimientos no funcionales del sistema:

- **RNF1. Rendimiento**

  RNF1. 1 El máximo de memoria RAM que debe de consumir el sistema es de 200 Mb.
- **RNF2. Usabilidad**

  RNF2. 1 El sistema podrá ser utilizado por aquellas personas que tengan conocimientos básicos de computación.

  RNF2. 2 El módulo debe describir con claridad los parámetros de configuración del sistema.
- **RNF3 Fiabilidad y disponibilidad**

  RNF3. 1 El componente puede estar funcionando a 24 horas, tiempo completo.
- **RNF4 Eficiencia**

  RNF4. 1 El tiempo de respuesta estará dado según la cantidad de videos-sensores cagados al mismo tiempo.
- **RNF5 Portabilidad**

  RNF5. 1 El componente debe poder ser utilizado sobre diferentes plataformas como Windows, Linux y Mac OS.
- **RNF6 Hardware**

  RNF6. 1 Se debe tener como mínimo 1Gb de RAM y procesador Pentium 4 a 3 GHz.
- **RNF7 Restricciones del diseño**

  RNF7. 1 Lenguaje:

  - El lenguaje que se utilizará para el desarrollo del sistema será C++.
- **RNF8 Soporte**

  RNF8. 1 El sistema recibirá mantenimiento en el período de tiempo determinado por el equipo de desarrollo y el Centro GEYSED.

3.5 Modelo de Casos de Uso del Sistema Propuesto

El modelo de casos de uso describe la funcionalidad propuesta del nuevo sistema. Un caso de uso representa una unidad discreta de interacción entre un usuario (humano o máquina) y el sistema. Un Caso de Uso es una unidad simple de trabajo significativo. Cada caso de uso tiene una descripción que describe la funcionalidad que se construirá en el sistema propuesto. Un caso de uso puede "incluir" la funcionalidad de otro caso de uso o "extender" a otro caso de uso con su propio comportamiento.

3.5.1 Descripción de los Actores

Actor del Sistema	Descripción
Operador	Es la persona que inicia la Configuración del Sistema.
Proceso Automático Sistema Analytic	Es el proceso automático que se encarga de Cargar la Configuración editada por el operador y Establecer la Comunicación con el Gestor.
Gestor	Es el sistema que manda a Cargar Video-Sensor para iniciar procesamiento donde se notifican los sucesos detectados al Gestor, este sistema se encarga de Detener Procesamientos.

Tabla 3: Descripción de los Actores

3.5.2 Diagrama de Casos de Uso del Sistema

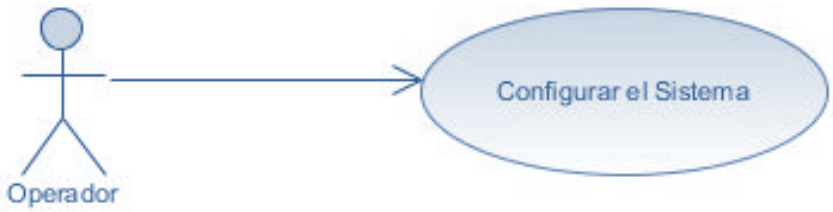

Figura 6: Caso de Uso "Configurar el Sistema"

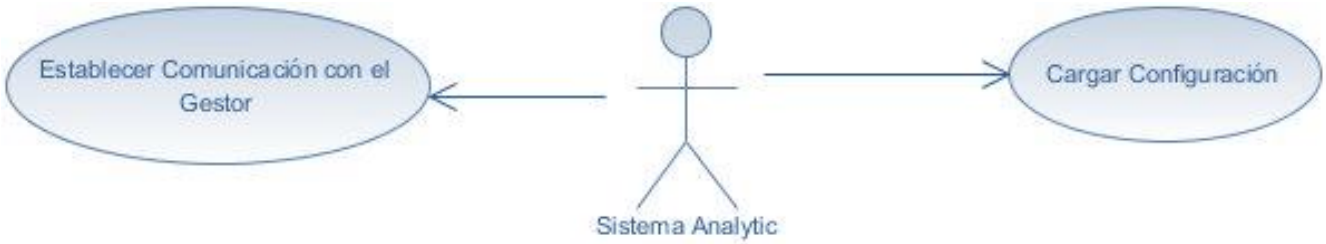

Figura 7: Caso de Uso del Sistema

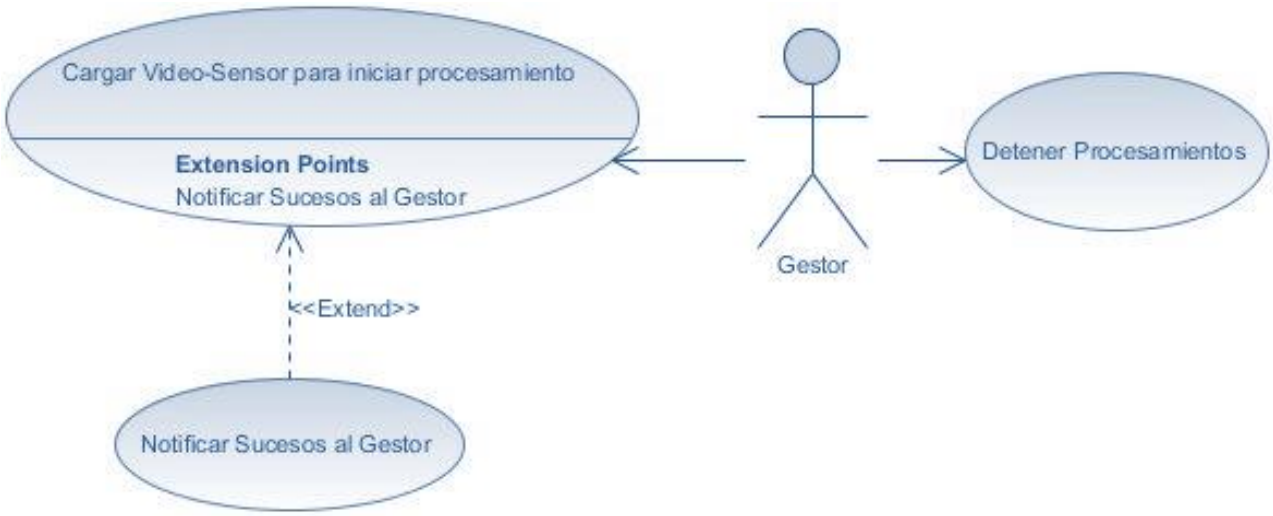

Figura 8: Caso de Uso "Cargar Video-sensor para iniciar procesamiento, Detener Procesamientos y Notificar sucesos al Gestor"

3.5.3 Descripción extendida de los Casos de Uso

Caso de uso	Configurar el sistema
Actores	Operador
Propósito	Este caso de uso debe permitir configurar todos los parámetros de comunicación entre el Gestor y el Analytic, además de especificar la ruta de los plugin.
Resumen	El caso de uso inicia cuando el operador desea configurar el sistema.
Precondiciones	-
Referencias	RF 1
Prioridad	Crítico

Flujo Normal de Eventos	
Acción de actor	**Respuesta del Sistema**
1. El caso de Uso inicia cuando el operador desea Configurar el Sistema.	1.1 El sistema le pide los parámetros a configurar para la comunicación entre el Gestor y el Analytic: • IP(Formato: cadena, Obligatorio: Si) • Puerto(Formato: numérico, Obligatorio :Si)
	1.2 El sistema pide la ruta donde se encuentran los plugins a cargar. • Ruta(Formato: cadena, Obligatorio: Si)
2 El operador da clic sobre el botón "aceptar"	2.1 El caso de uso termina cuando el sistema actualiza o crea en caso de no existir, un fichero de configuración con el ip, el puerto y la ruta de los plugins especificado en la entrada de los datos.
Flujos Alternos de Eventos	
Acción del actor	**Respuesta del Sistema**
	1.1 El sistema muestra un mensaje de error "Formato de IP incorrecto" cuando el usuario introduce datos incorrectos y la aplicación se queda abierta para que se vuelvan a llenar los campos.
	1.1 El sistema muestra un mensaje de error "Debe llenar todos los campos" cuando el usuario deja el campo del IP y el Puerto sin llenar y la aplicación se queda abierta para que se vuelvan a llenar los campos.
	1.2 El sistema muestra un mensaje de error "Debe llenar todos los campos" cuando el usuario deja el campo de la Ruta sin llenar y la aplicación se queda abierta para que

	se vuelvan a llenar los campos.
Prototipo de Interfaz	
	
Poscondiciones	Se configuraron todos los parámetros de la aplicación (parámetros de comunicación entre el Gestor y el Análisis, más la ruta donde se encuentran los plugins).

Tabla 4: Descripción extendida para el CU Configurar el sistema

Caso de Uso	Cargar Video-Sensor para iniciar procesamiento
Actores	Gestor
Propósito	Este caso de uso debe permitir cargar los diferentes Video Sensores que se requieran según la demanda del sistema e iniciar el procesamiento de los mismos.
Resumen	El caso de uso inicia cuando el Gestor envía la orden de cargar un video sensor.
Precondiciones	-
Referencias	RF 3
Prioridad	Crítico
Flujo Normal de Eventos	

Acción del actor	**Respuesta del Sistema**
1. El caso de uso inicia cuando el Gestor envía la orden de cargar un video sensor.	1.1 El sistema Analytic recibe una lista de parámetros (tipo de plugin, id cámara) enviado por el gestor.
	1.2 El sistema busca el plugin a cargar en la ruta especificada en la configuración.
	1.3 El caso de uso termina cuando el sistema carga el plugin de acuerdo a la orden enviada y lo manda a procesar.
Flujos Alternos de Eventos	
Acción del actor	**Respuesta del Sistema**
	1.3 Si el tipo de plugin a cargar no se encuentra, se emite un mensaje de que "no se encuentra disponible el plugin".
Poscondiciones	El sistema cargó los plugins que se requirieron según la demanda del sistema.

Tabla 5: Descripción extendida para el CU Cargar Video-Sensor para iniciar procesamiento

Caso de Uso	Cargar Configuración
Actores	Proceso Automático Sistema Analytic
Propósito	Este caso de uso debe permitir cargar la configuración editada por el operador.
Resumen	El caso de uso inicia cuando el Analytic se ejecuta.
Precondiciones	Es necesario Configurar el Sistema, para luego poder Cargar Configuración.
Referencias	RF 2
Prioridad	Crítico
Flujo Normal de Eventos	
Acción del actor	**Respuesta del Sistema**
1. El caso de uso inicia cuando el Analytic se ejecuta.	1.1 El sistema carga el fichero de configuración.

	1.2 El caso de uso termina cuando el Analytic almacena la información, donde carga la configuración editada por el operador.
Flujos Alternos de Eventos	
Acción del actor	**Respuesta del Sistema**
	1.2 Si el archivo de configuración no existe, lo crea nuevo con valores predeterminados: Para Linux: • IP: localhost • Puerto: "24" • Ruta: "/home/$${USER}/plugins" Para Windows • IP: localhost • Puerto: “24” • Ruta: ”C:\Documents and Settings\User”
Poscondiciones	El sistema cargó el archivo de configuración.

Tabla 6: Descripción extendida para el CU Cargar Configuración.

Caso de Uso	Notificar Sucesos al Gestor
Actores	Gestor
Propósito	Este caso de uso debe permitir enviar al gestor los diferentes sucesos detectados por los videos-sensores.
Resumen	El caso de uso inicia cuando el gestor manda a cargar un plugin para iniciar procesamiento para enviar los diferentes sucesos detectados por los videos-sensores.
Precondiciones	-
Referencias	RF 5
Prioridad	Critico
Flujo Normal de Eventos	
Acción del actor	**Respuesta del sistema**

1. El caso de uso inicia cuando el gestor manda a cargar un plugin para iniciar procesamiento para enviar los diferentes sucesos detectados por los videos-sensores.	1.1 Cuando alguno de los plugins cargado detecta una acción, se emite una señal con el mensaje de dicha acción.
	1.2 El HiloProcesador correspondiente a ese plugin emite concurrentemente una señal con el mismo mensaje.
	1.3 El Analytic emite concurrentemente una señal con el mismo mensaje.
	1.4 El ComunicacionGestor captura la señal emitida por el Analytic.
	1.5 El caso de uso termina cuando ComunicacionGestor le envía el suceso detectado al Gestor mediante el protocolo TCP/IP.
Flujos Alternos de Eventos	
Acción del actor	**Respuesta del Sistema**
	1.5 Si la comunicación no se pudo establecer, no se le notifica al Gestor ningún suceso.
Poscondiciones	El sistema notifica los sucesos al Gestor.

Tabla 7: Descripción extendida para el CU Notificar Sucesos al Gestor

Caso de Uso	Establecer Comunicación con el Gestor
Actores	Proceso Automático Sistema Analytic
Propósito	Este caso de uso debe permitir establecer la comunicación entre el Gestor y el Analytic.
Resumen	El caso de uso inicia cuando el Analytic se ejecuta.
Precondiciones	-
Referencias	RF 4

Prioridad	Crítico
Flujo Normal de Eventos	
Acción del actor	**Respuesta del Sistema**
1. El caso de uso inicia cuando el Analytic se ejecuta.	1.1 El caso de uso termina cuando el sistema establece la conexión mediante el protocolo TCP/IP según la configuración cargada.
Flujos Alternos de Eventos	
Acción del actor	**Respuesta del Sistema**
	(si no se logra conectar que pasa)
Poscondiciones	

Tabla 8: Descripción extendida para el CU Establecer Comunicación con el Gestor

Caso de Uso	Detener Procesamientos
Actores	Gestor
Propósito	Este caso de uso debe permitir detener procesamientos.
Resumen	El caso de uso inicia cuando el Gestor envía la orden de detener un proceso específico.
Precondiciones	-
Referencias	RF 6
Prioridad	Crítico
Flujo Normal de Eventos	
Acción del actor	**Respuesta del Sistema**
1. El caso de uso inicia cuando el Gestor envía la orden de detener un proceso específico.	1.1 Busca el proceso en la lista de Hilos que posee el Analytic.
	1.2 El caso de uso culmina cuando el proceso es detenido.
Flujos Alternos de Eventos	
Acción del actor	**Respuesta del Sistema**

Poscondiciones	

Tabla 9: Descripción extendida para el CU Detener Procesamientos

3.6 Descripción de la Arquitectura

“La Arquitectura de Software(AS) constituye un puente entre el requerimiento y el código, ocupando el lugar que en los gráficos antiguos se reservaba para el diseño” (18).

La Arquitectura del Software es el estudio de la estructura a gran escala y el rendimiento de los sistemas de software. La arquitectura de un sistema que incluye la división de funciones entre los módulos de un sistema, los medios de comunicación entre los módulos y la representación de la información compartida (17). Establece los fundamentos para que los desarrolladores de un proyecto de software trabajen en una línea común permitiendo alcanzar los objetivos del sistema de información, cubriendo todas las necesidades.

La Arquitectura de Software está compuesta por componentes, definiendo el ¿qué? Dela aplicación, posee relaciones y restricciones que definen el ¿cómo? realizar la aplicación. Posee requisitos tanto funcionales (RF) como no funcionales (RNF) ligada más a este último, que junto a las decisiones significativas definen el ¿por qué? de las decisiones a tomar, teniendo en cuenta rendimiento, confiabilidad y disponibilidad, escalabilidad. Permite a los miembros del grupo de desarrollo encaminar el producto por una línea de trabajo común, logrando alcanzar los objetivos propuestos.
La arquitectura de software se relaciona con el diseño y la implementación de estructuras de software de alto nivel, ensambla un cierto número de elementos arquitectónicos de forma adecuada para satisfacer la mayor funcionalidad y requerimientos de un sistema.

3.6.1 Estilos y Patrones Arquitectónicos

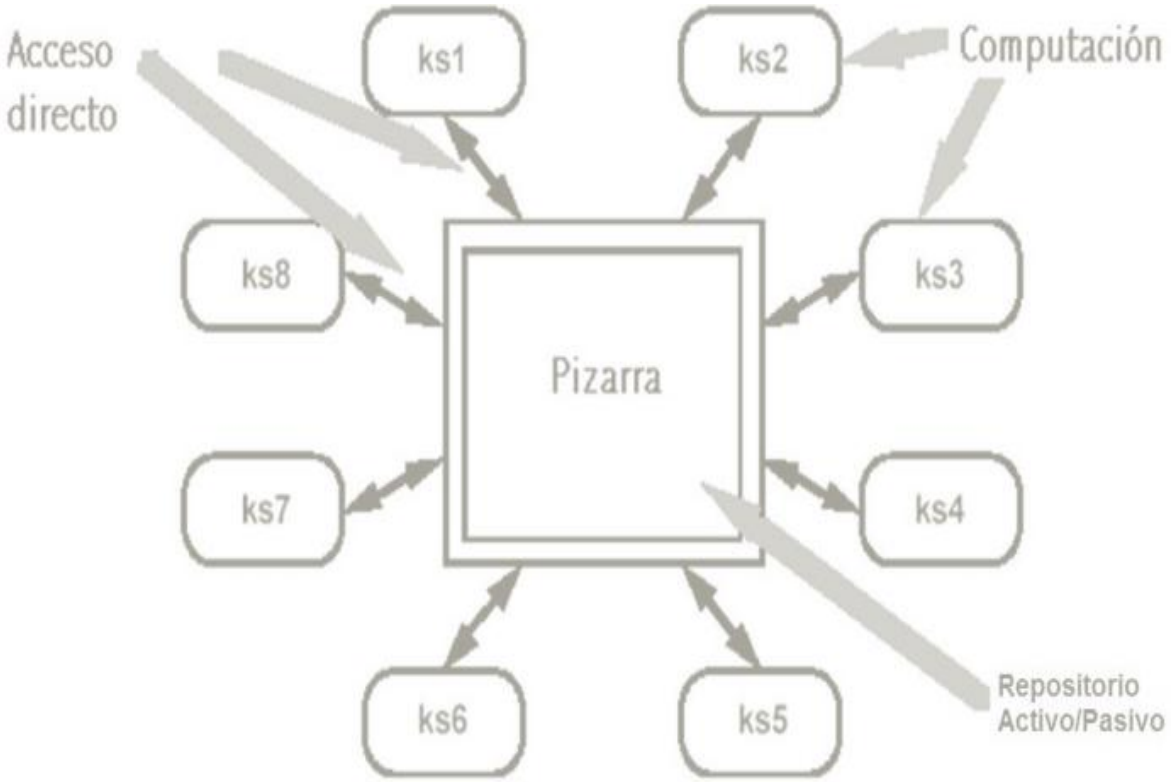

Figura 9: Arquitectura base en forma de pizarra

“El sistema Suria está diseñado siguiendo un estilo arquitectónico centrado en datos específicamente utilizará el patrón arquitectónico “Arquitectura base en forma de pizarra, en su variante de tablero de control. Ésta arquitectura desacopla el sistema en componentes denominados agentes autónomos, los cuales son independientes en la realización atómica de su funcionalidad. Pero dependen de una entrada de información externa, que es provista por otros agentes, y a su vez, producen un resultado que puede ser entrada de otros agentes. Cada agente se rige por interfaces estándares que permiten cambios en el ambiente externo al agente sin que éste sufra cambios en su funcionamiento interno. Todo el funcionamiento de los agentes autónomos, está coordinado por un elemento central, denominado Repositorio Activo, el cual entrega y recibe información de los agentes y coordina su funcionamiento” (19). En la figura 9 se puede observar dicha arquitectura.

El sistema Video Vigilancia Suria posee un Gestor que tiene la función de repositorio activo al que se le pueden conectar diferentes agentes autónomos, uno de los agentes que compone este sistema es el módulo Analytic donde su función consistirá en gestionar el procesamiento inteligentes de video a

través de los video sensores Detección de Movimiento y Perimetrado virtual. Este módulo Analytic será la fachada entre el Gestor y los video sensores.

Los videos-sensores son los que realizan todo el procesamiento inteligente de video, poseen un estilo arquitectónico llamado Pipes & Filters (Tuberías y Filtros), donde cada componente tiene un conjunto de entradas y un conjunto de salidas. Un componente lee un flujo de datos en la entrada y produce un flujo de datos diferente en su salida. Esto es logrado aplicando una transformación local al flujo de entrada mientras este se lee, de tal forma que el flujo de salida empieza antes que se consume todo el flujo de entrada. Este patrón divide la tarea de un sistema en varios pasos de procesamiento secuenciales. Estos pasos están conectados por el flujo de datos a través del sistema, cada paso del procesamiento está encapsulado en un componente de filtro. Los datos pasan a través de las tuberías que son los conectores que sirven como conductos para transmitir las salidas de un filtro a las entradas de otro.

Este patrón de arquitectura es particularmente efectivo a la hora de descomponer el problema en pasos independientes, reutilizar filtros, facilitar el mantenimiento, independencia y ejecución concurrente de filtros. Este patrón tiene algunas restricciones, como que los filtros deben ser entidades independientes: en particular no deben compartir estados con otros filtros. Los filtros no conocen la identidad del filtro de donde proviene el flujo que reciben como entrada y tampoco la identidad del filtro a donde llega el flujo de salida. En la figura 10 se observa una vista lógica del módulo Analytic.

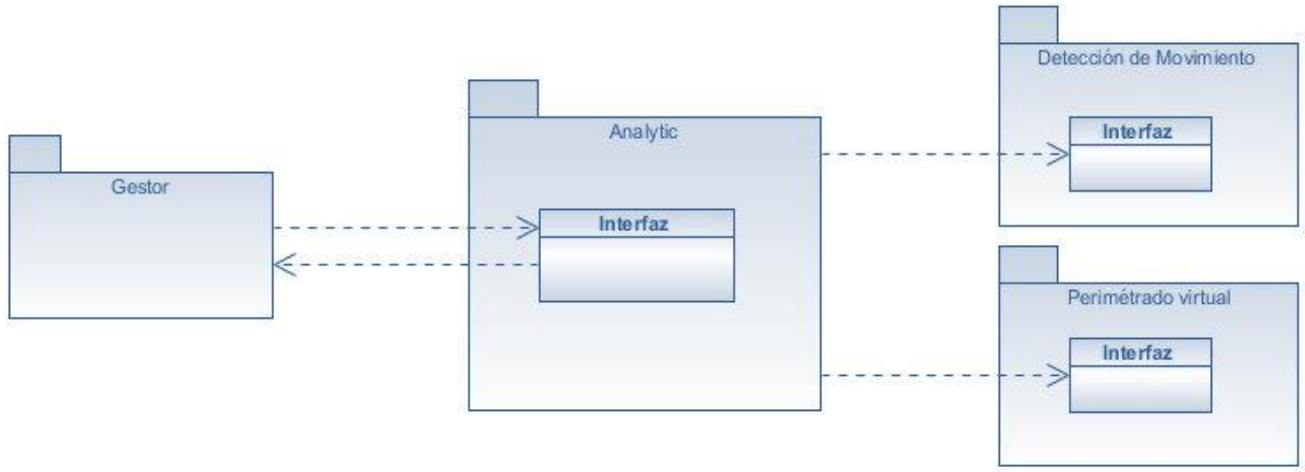

Figura 10: Vista lógica del módulo Analytic

3.6.2 Arquitectura basada en componentes

Una arquitectura basada en componentes describe una aproximación de ingeniería de software al diseño y desarrollo de un sistema. Esta arquitectura se enfoca en la descomposición del diseño en componentes funcionales o lógicos que expongan interfaces de comunicación bien definidas. Esto provee un nivel de abstracción mayor que los principios de orientación a objetos y no se enfoca en asuntos específicos de los objetos como los protocolos de comunicación y la forma como se comparte el estado.

Una arquitectura basada en componentes en la creación del módulo Análisis se pudiese aprovechar en la capacidad de flexibilidad y adaptación del sistema al poder incorporar nuevos plugin con el objetivo de ganar en funcionalidad, aunque tiene que quedar claro que los plugin y los componentes son familia, pero no son lo mismo, también se aprovecharía la escalabilidad aunque no sea realmente un desarrollo basado en componentes pues se optimiza el sistema de forma lógica ya que la escalabilidad constituye un factor influyente en el crecimiento del sistema.

El estilo de arquitectura basado en componentes tiene las siguientes características:

- Es un estilo de diseño para aplicaciones compuestas de componentes individuales.
- Pone énfasis en la descomposición del sistema en componentes lógicos o funcionales que tienen interfaces bien definidas.
- Define una aproximación de diseño que usa componentes discretos, los que se comunican a través de interfaces que contienen métodos, eventos y propiedades (20).

Principios Fundamentales

Un componente es un objeto de software específicamente diseñado para cumplir con cierto propósito. Los principios fundamentales cuando se diseña un componente es que estos deben ser:

- **Reusable**. Los componentes son usualmente diseñados para ser utilizados en escenarios diferentes por diferentes aplicaciones, sin embargo, algunos componentes pueden ser diseñados para tareas específicas.
- **Sin contexto especifico**. Los componentes son diseñados para operar en diferentes ambientes y contextos. Información específica como el estado de los datos deben ser pasadas al componente en vez de incluirlos o permitir al componente acceder a ellos.

- **Extensible**. Un componente puede ser extendido desde un componente existente para crear un nuevo comportamiento.
- **Encapsulado**. Los componentes exponen interfaces que permiten al programa usar su funcionalidad. Sin revelar detalles internos, detalles del proceso o estado.
- **Independiente**. Los Componentes están diseñados para tener una dependencia mínima de otros componentes. Por lo tanto los componentes pueden ser instalados en el ambiente adecuado sin afectar otros componentes o sistemas.

Beneficios

La arquitectura basada en componentes posee significativos beneficios donde los mismos son de gran importancia según el estilo de arquitectura basado en componentes:

- **Facilidad de Instalación**. Cuando una nueva versión esté disponible, se podrá reemplazar la versión existente sin impacto en otros componentes o el sistema como un todo.
- **Costos reducidos**. El uso de componentes de terceros permite distribuir el costo del desarrollo y del mantenimiento.
- **Facilidad de desarrollo**. Los componentes implementan una interfaz bien definida para proveer determinada funcionalidad permitiendo el desarrollo sin impactar otras partes del sistema.
- **Reusable**. El uso de componentes reutilizables significa que ellos pueden ser usados para distribuir el desarrollo y el mantenimiento entre múltiples aplicaciones y sistemas.
- **Mitigación de complejidad técnica**. Los componentes mitigan la complejidad por medio del uso de contenedores de componentes y sus servicios. Ejemplos de servicios de componentes incluyen activación de componentes, gestión de la vida de los componentes, gestión de colas de mensajes para métodos del componente y transacciones (19)**.**

3.7 Modelo de Análisis

El modelo de análisis es un modelo conceptual y genérico, es una abstracción del sistema, este define una estructura para modelar el sistema, puede no mantenerse durante todo el ciclo de vida del software. El modelo de análisis realiza un bosquejo del diseño del sistema, este modelo es menos formal que el modelo de diseño.

3.7.1 Clases del Análisis

Las clases de análisis representan una abstracción de una o varias clases o subsistemas del diseño del sistema. Las clases del análisis se centran en los requisitos funcionales y deja los no funcionales, el comportamiento se especifica mediante responsabilidades de nivel más alto y menos formal, tiene atributos de nivel de abstracción muy alto, participa en relaciones del modelo conceptual.

Para la realización del modelo de análisis se utilizan los siguientes tipos de clases:

Clase Interfaz

“Las clases interfaz se utilizan para modelar la interacción entre el sistema y sus actores (es decir, usuarios y sistemas externos). Esta interacción a menudo implica recibir (y presentar información) y peticiones de (y hacia) los usuarios y los sistemas externos” (16).
Cada clase interfaz debería asociarse con al menos un actor, y viceversa. Representan ventanas, formularios, paneles, interfaces de comunicación.

Clase Entidad

“Las clases de entidad se utilizan para modelar información que posee una vida larga y que es a menudo persistente. Las clases de entidad modelan la información y el comportamiento asociado de algún fenómeno o concepto, como una persona, un objeto del mundo real, o un suceso del mundo real” (16).
Suelen sacarse de las clases de entidad del negocio.

Clase Control

“Las clases de control representan coordinación, secuencia, transacciones, y control de otros objetos y se usan con frecuencia para encapsular el control de un caso de uso en concreto. Las clases de

control también se utilizan para representar derivaciones y cálculos complejos, como la lógica del negocio, que no pueden asociarse con ninguna información concreta, de larga duración, almacenada por el sistema" (16).
Los aspectos dinámicos y delegaciones a otras clases del sistema se modelan con estas clases.

3.7.2 Diagrama de clases de análisis

En la figura 11 se muestra el diagrama de clases del análisis para el caso de uso Cargar Video-Sensor para iniciar procesamiento. Se muestra la clase interfaz Cargar Plugins que es la que se encargará de comunicarse con la clase de control CargarPlugin que manejará todos los pasos que se van a ejecutar.

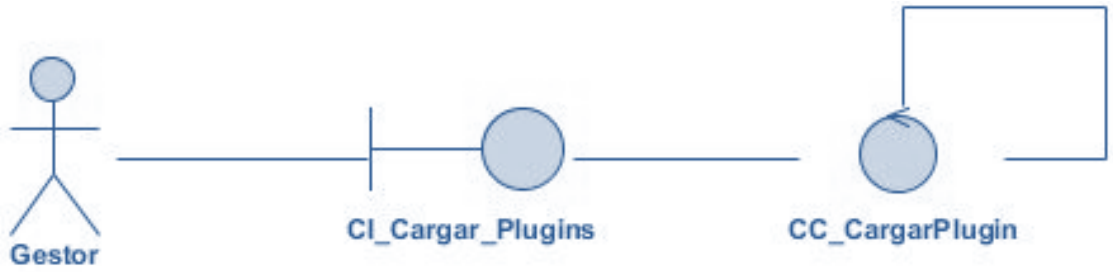

Figura 11: Diagrama de Clases del Análisis para el CU Cargar Video-Sensor para iniciar procesamiento.

3.7.3 Diagrama de Interacción

Craig Larman hace un análisis de los diagramas de interacción donde explica las interacciones existentes entre las instancias (y las clases) del modelo de estas. El punto de inicio de las interacciones es el cumplimiento de las poscondiciones de los contratos de operación. Estos diagramas le muestran a los desarrolladores la manera en que las clases interactúan una con otras y los mensajes que se envían.

UML define dos tipos de estos diagramas; ambos sirven para expresar interacciones semejantes o idénticas de mensajes:

- **Diagrama de colaboración**
- **Diagrama de secuencia**

Diagramas de colaboración:

Un diagrama de colaboración es un diagrama de clases que describen las interacciones entre los objetos en un formato de grafo o red, contiene roles de clasificador y roles de asociación en lugar de sólo clasificadores y asociaciones. Los roles de clasificador y los roles de asociación describen la configuración de los objetos y de los enlaces que pueden ocurrir cuando se ejecuta una instancia de la colaboración.

Diagramas de secuencia:

Un diagrama de secuencia describe las interacciones en una especie de formato de cerca o muro, representa una interacción como un gráfico bidimensional. La dimensión vertical es el eje de tiempo, que avanza hacia abajo de la página. La dimensión horizontal muestra los roles de clasificador que representan objetos individuales en la colaboración. Cada rol de clasificador se representa mediante una columna vertical-línea de vida. Durante el tiempo que existe un objeto, el rol se muestra por una línea discontinua. Durante el tiempo que dura una activación de un procedimiento en el objeto, la línea de vida se dibuja como una línea doble. Se muestra un mensaje como una flecha desde la línea de vida de un objeto a la del otro. Las flechas se organizan en el diagrama en orden cronológico hacia abajo.

3.7.4 Diagrama de Colaboración

En la figura 12 se muestra el diagrama de colaboración para el caso de uso Cargar Video-Sensor para iniciar procesamiento. Se observan los principales mensajes intercambiados entre las clases para realizar el caso de uso en cuestión.

Figura 12: Diagrama de Colaboración para el CU Cargar Video-Sensor para iniciar procesamiento.

3.8 Modelo del Diseño

El modelo de diseño es un modelo físico y concreto, debe ser mantenido durante todo el ciclo de vida del software. Este modelo le da forma al sistema, es un plano de la implementación. El modelo de diseño es una realización del diseño del sistema y es más formal con respecto al modelo de análisis.

3.8.1 Clases del Diseño

Una clase del diseño es una abstracción sin costuras de una clase o construcción similar en la implementación del sistema. Lo cual significa que el lenguaje utilizado para especificar una clase del diseño es lo mismo que el lenguaje de programación, esto conlleva a que las operaciones, atributos, tipos y demás son especificados utilizando la sintaxis del lenguaje de programación elegido.
A continuación se muestra en la figura 13 el diagrama de clases del diseño para el CU Cargar Video-Sensor para iniciar procesamiento.

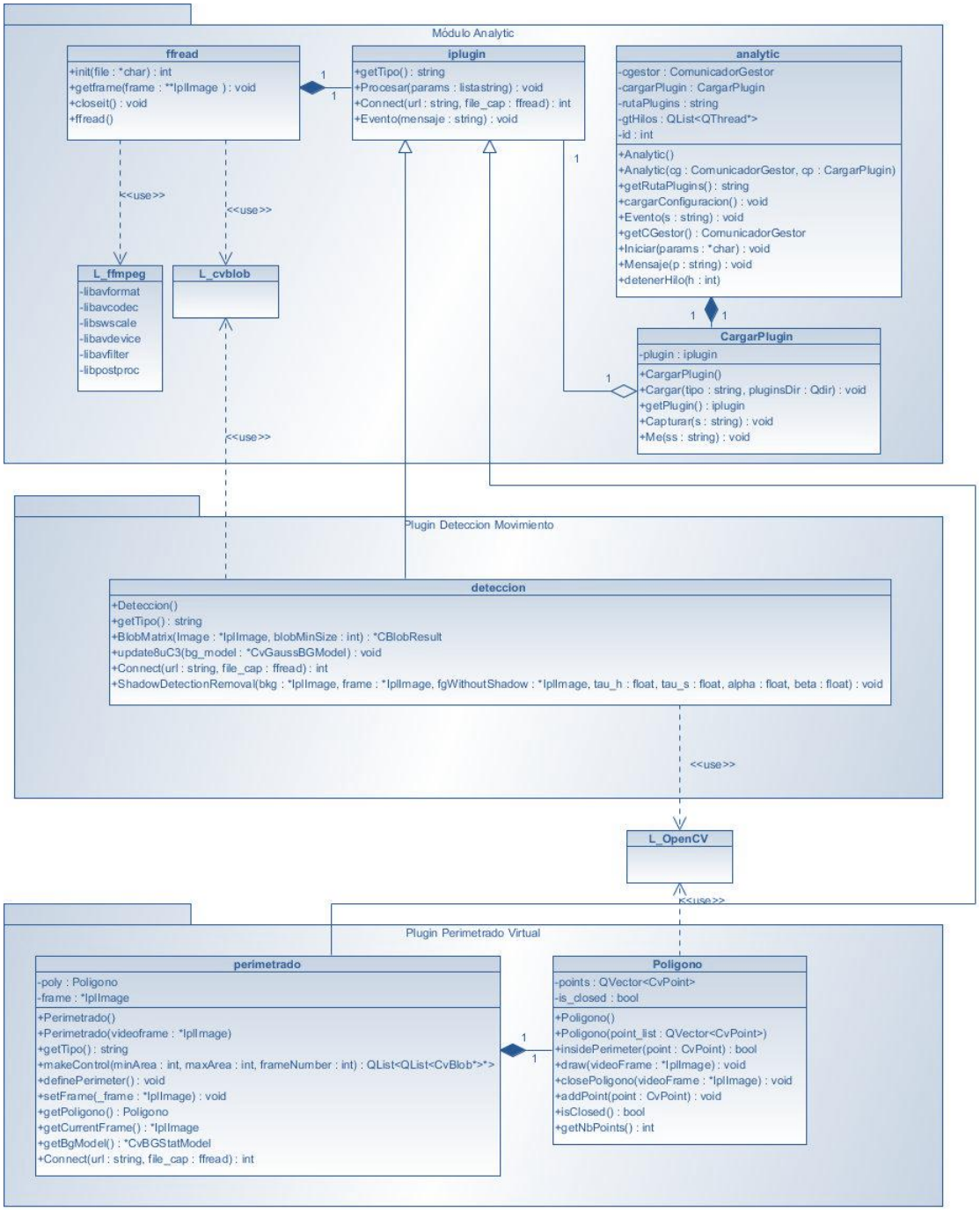

Figura 13: Diagrama de clase del Diseño para el CU Cargar Video-Sensor para iniciar procesamiento.

3.8.2 Diagrama de Secuencia del Diseño

La secuencia de acciones en un caso de uso comienza cuando un actor invoca el caso de uso mediante el envío de algún tipo de mensaje al sistema. Si se considera el “interior”, se tendrá algún

objeto del diseño que recibe el mensaje del actor. Después el objeto de diseño llama a algún otro objeto, y de esta manera los objetos implicados interactúan para realizar y llevar a cabo el caso de uso. En el diseño es preferible representar esto con diagramas de secuencia ya que el centro de atención principal es el encontrar secuencias de interacciones detalladas y ordenadas en el tiempo. En la figura 14 se muestra el diagrama de secuencia para el caso de uso Cargar Video-Sensor para iniciar procesamiento.

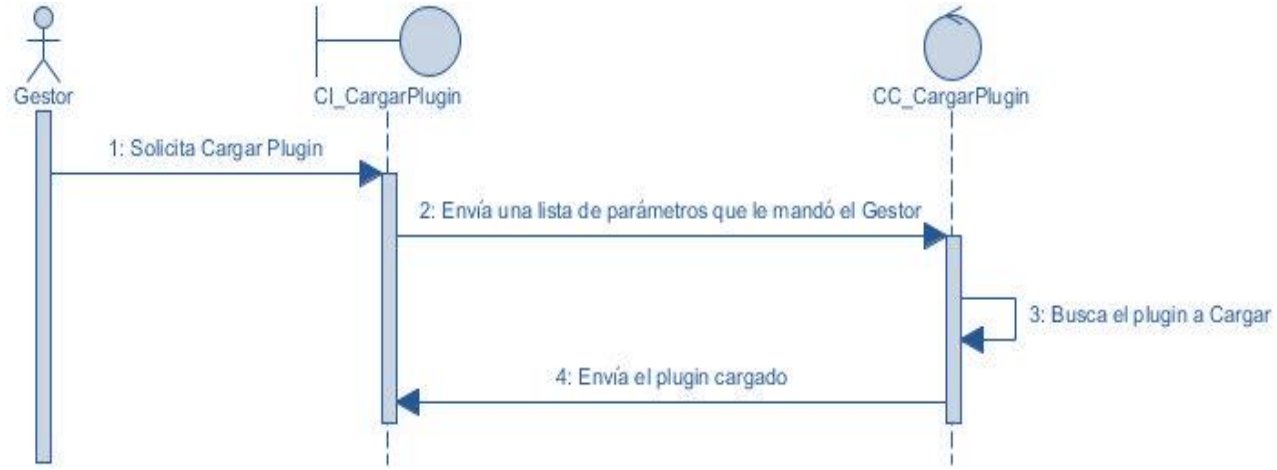

Figura 14: Diagrama de Secuencia para el CU Cargar Video-Sensor para iniciar procesamiento.

3.8.3 Descripción de las clases

Nombre	configurador
Tipo de Clase	Control
Atributo	**Tipo**
No tiene	No tiene
Para cada Responsabilidad	
Nombre	cargarSettings()
Descripción	Busca el fichero de configuración en un lugar por defecto.
Nombre	guardarSettings()
Descripción	Guarda el ip, puerto y dirección de los plugins.

Tabla 10: Descripción de la clase “configurador”

Nombre	analytic
Tipo de Clase	Control
Atributo	**Tipo**
cgestor	ComunicacionGestor
cargarPlugin	CargarPlugin
rutaPlugins	QString

Para cada Responsabilidad	
Nombre	cargarConfiguracion()
Descripción	Carga lo que está en el fichero de configuración.
Nombre	getRutaPlugins()
Descripción	Devuelve la ruta del plugin.
Nombre	Evento(QString texto)
Descripción	Emite una señal con texto que le pasan por parámetro.
Nombre	Iniciar(char* params)
Descripción	Divide lo que le pasan por parámetro por ";"coge el tipo de sensor a cargar y manda a cargar el plugin. Si se logra cargar levanta un hilo con ese plugin para ponerlo a procesar.
Nombre	getCGestor()
Descripción	Se encarga de comunicarse con el gestor.

Tabla 11: Descripción de la clase "analytic"

Nombre	cargarplugin
Tipo de Clase	Control
Atributo	**Tipo**
plugin	IPlugin
Para cada Responsabilidad	
Nombre	Cargar(QString tipo, QDir pluginsDir)
Descripción	Busca el plugin que mandan a cargar en la ruta de los plugins si lo encuentra lo asigna al objeto plugin de la clase cargar plugin sino emito señal que no está disponible ese plugin.
Nombre	Capturar(QString evento)
Descripción	Es un slot que captura una señal y emite otra señal llamada Me (texto a enviar).
Nombre	getPlugin()
Descripción	Devuelve el plugin a cargar.

Tabla 12: Descripción de la clase "cargarplugin"

Nombre	hiloprocesador
Tipo de Clase	Control
Atributo	**Tipo**
plugin	IPlugin
params	QStringList
Para cada Responsabilidad	
Nombre	HiloProcesador(IPlugin *plug, QStringList params)
Descripción	Es el constructor la clase donde se conectan las señales del plugin con la del hilo.

Nombre	run()
Descripción	Método para cuando el hilo corra, el plugin comience a procesar.

Tabla 13: Descripción de la clase "hiloprocesador"

Nombre	comunicaciongestor
Tipo de Clase	Control
Atributo	**Tipo**
ip	QString
puerto	int
Para cada Responsabilidad	
Nombre	etIP()
Descripción	Devuelve el ip.
Nombre	setIP(QString)
Descripción	Cambia el ip que existe por el pasado por parámetro.
Nombre	getPuerto()
Descripción	Devuelve el puerto.
Nombre	setPuerto(int)
Descripción	Cambia el puerto que existe por el pasado por parámetro.

Tabla 14: Descripción de la clase "comunicaciongestor"

Nombre	iplugin
Tipo de Clase	Interfaz
Atributo	**Tipo**
No tiene	No tiene
Para cada Responsabilidad	
Nombre	getTipo()
Descripción	Devuelve el tipo "deteccionMovimiento" o "PerimetradoVirtual"
Nombre	Procesar(QStringList params)
Descripción	Manda a procesar un grupo de parámetros.
Nombre	Evento(QString mensaje)
Descripción	Emite una señal con texto que le pasan por parámetro.

Tabla 15: Descripción de la clase "iplugin"

3.9 Patrones de Diseño

Mediante un patrón se llega a entender un problema y su solución puesto que para llegar a entender el mismo se siguen un conjunto de pasos donde se logra finalmente obtener la respuesta del problema o solución. A un patrón se le da un nombre donde se pueda aplicar a nuevos y diferentes contextos,

proporciona consejos sobre el modo de aplicarlo en varias circunstancias, y considera los puntos fuertes y compromisos. La gran mayoría de los patrones posee guías sobre la forma en que debería asignarse las responsabilidades a los objetos, dada una categoría específica del problema.

Los patrones de diseño ayuda a como se debe estructurar las clases y objetos para guiar todo el procesos de creación de software. Componen soluciones concretas a problemas que se presentan durante el diseño de una aplicación. Los patrones de diseño utilizados en la presente investigación son: los patrones GRASP y los patrones GoF.

Los patrones GRASP (General Responsibility Assignment Software Patterns) son patrones generales para asignar responsabilidades para diseñar con éxito el software orientado a objetos. En el desarrollo del sistema se aplicaron los patrones: Experto, Creador, Bajo Acoplamiento, Alta Cohesión y Controlador.

- **Experto**: Este patrón se utilizó para la asignación de responsabilidades indicando que la clase que cuenta con la información necesaria para cumplir una tarea, debe ser la responsable de ejecutar la misma, proporcionando que los objetos exploten su propia información para cumplir con sus funcionalidades.

- **Creador**: Este patrón se utilizó por ser el responsable de crear una nueva instancia de algunas clases del sistema.

- **Bajo Acoplamiento**: Se utilizó dicho patrón para establecer una escasa dependencia entre las clases, reduciendo el impacto de posibles cambios en el sistema.

- **Alta Cohesión**: Este patrón se utilizó porque se ocupó de que las clases del diseño realizarán las funcionalidades necesarias para cumplir con las tareas que tenían definidas.

- **Controlador**: Se utilizó este patrón pues el mismo se encargó de atender eventos del sistema.

Los patrones de diseño GoF (Gang of Four) se clasifican en patrones creacionales, estructurales y de comportamiento. Los creacionales resuelven problemas relativos a la creación de objetos, mientras los estructurales ofrecen solución a problemas relativos a la composición de objetos. Por su parte, los patrones de comportamiento resuelven los problemas referentes a la interacción entre los objetos.

El módulo Analytic utilizó el patrón estructural **Fachada**, porque proporciona una interfaz unificada que representa un conjunto de clases en un subsistema. La aplicación simplifica el acceso de

comunicación entre los plugin a través de una única interfaz donde varias clases del sistema se comunican con el subsistema (plugin) enviando peticiones a la fachada, la cual dirige las peticiones a los objetos apropiados.

3.10 Conclusiones

La modelación de las entidades del modelo de dominio y de análisis permite una mayor comprensión del problema a la hora de modelar la solución, ya que en este flujo se refinan y estructuran los requisitos obtenidos. El modelo de diseño es la base fundamental para la implementación, se enfoca en cómo va a estar estructurado e implementado el software. El establecimiento de la línea base de la arquitectura para el componente, posibilita una mayor comprensión del mismo y hace que aumenten las posibilidades de reutilizar tanto la arquitectura como el componente. La utilización de patrones de diseño permite ahorrar cantidades considerables de tiempo en el desarrollo, propiciando facilidades para la comprensión de la solución que se propone.

CAPÍTULO 4

Implementación y Prueba.

4.1 Introducción

En este capítulo quedarán expuestos todos los detalles referentes a la implementación del sistema, así como las pruebas que se le realicen al mismo. Se presentará el diagrama de despliegue donde se muestra la distribución física del sistema en los diferentes elementos de hardware que le darán soporte y se hará referencia además a los diagramas de componentes proporcionando una vista específica que detalle las organizaciones y dependencias lógicas entre los componentes de la aplicación que se desarrolla.

4.2 Diagrama de Componentes

Un diagrama de componentes describe la forma en que se organizan los componentes de software, así como la dependencia lógica entre los mismos, siendo estos componentes de código fuente, binarios o ejecutables. Los componentes representan todos los tipos de elementos software que incluye la fabricación de una aplicación informática.

Un componente es el empaquetamiento físico de los elementos de un modelo, como son las clases en el modelo de diseño, algunos estereotipos estándar de componentes son los siguientes:

- <<executable>> es un programa que puede ser ejecutado en un nodo.
- <<file>> es un fichero que contiene código fuente o datos.
- <<library>> es una librería estática o dinámica.
- <<document>> es un documento.
- <<table>> es una tabla de una base de datos.
- <<databases>> es una base de datos.

En la figura 15 que se muestra a continuación se observa el diagrama de componentes del sistema.

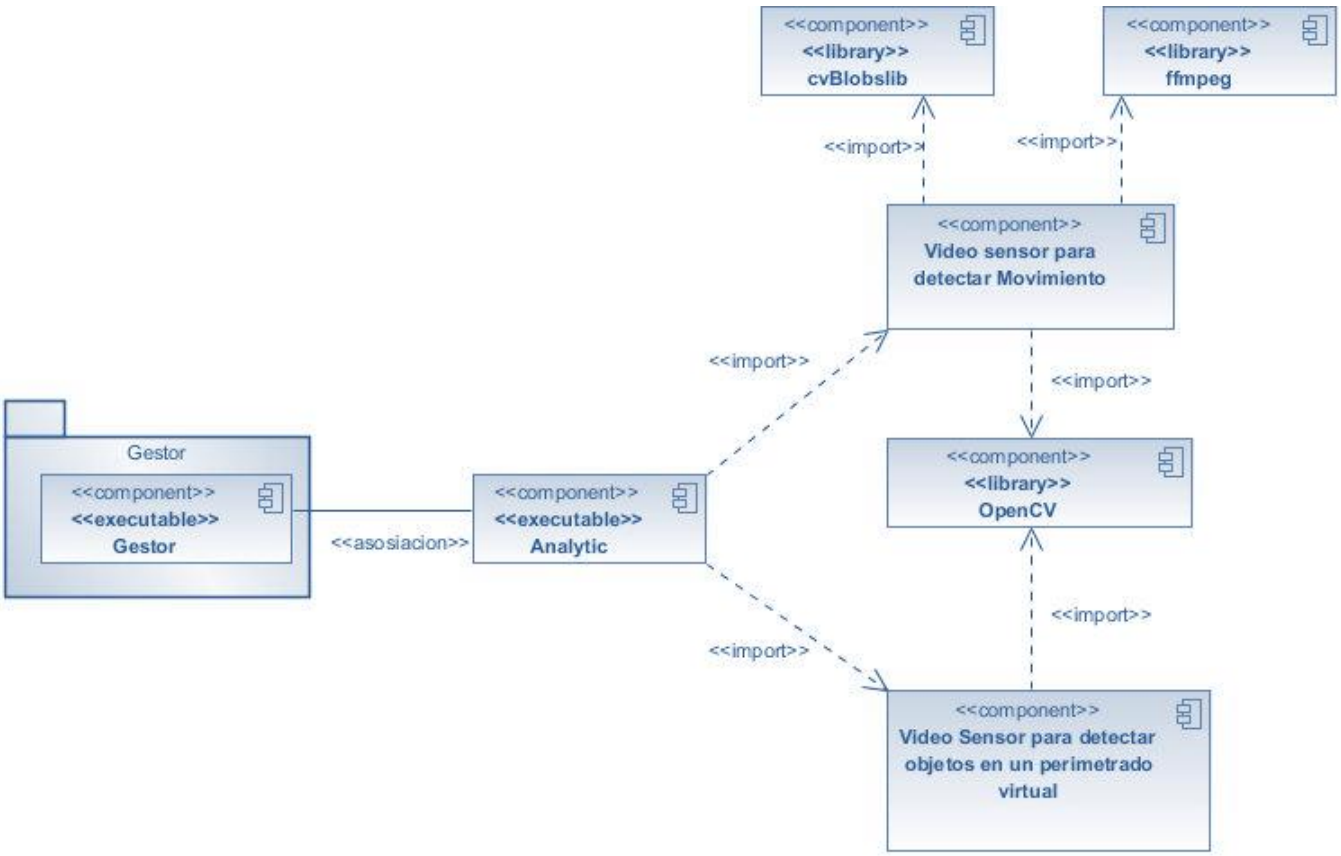

Figura 15: Diagrama de Componentes para el Sistema.

4.3 Modelo de Despliegue

“El modelo de despliegue es un modelo de objetos que describe la distribución física del sistema en términos de cómo se distribuye la funcionalidad entre nodos de cómputo. Se utiliza como entrada fundamental en las actividades de diseño e implementación debido a que la distribución del sistema tiene una influencia principal en su diseño” (16).

A continuación se muestra en la figura 16 la distribución física del sistema a través del diagrama de despliegue:

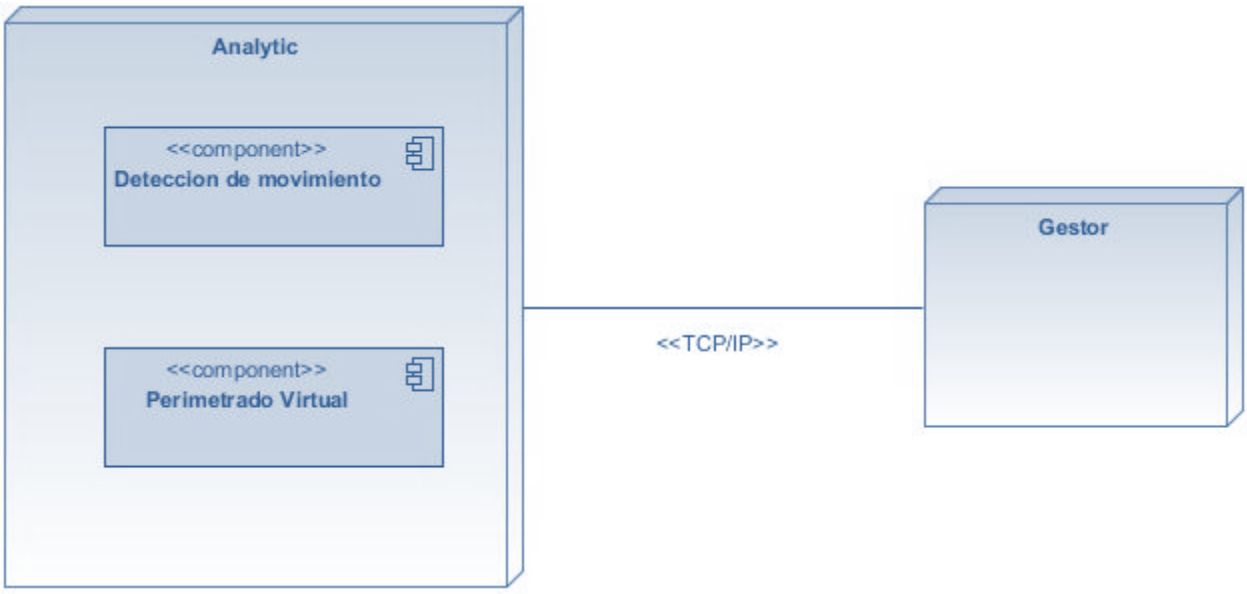

Figura 16: Modelo de despliegue del Sistema

4.4 Pruebas de software

Probar un sistema resulta prácticamente imprescindible si se quiere verificar la calidad del mismo. Una prueba no es más que un proceso de ejecución de un programa con el propósito de encontrar errores. Se reconoce como una buena prueba a aquella que tiene altas probabilidades de encontrar faltas que no han sido detectadas hasta el momento, ya que su objetivo fundamental es demostrar la existencia de errores, nunca la ausencia de estos.

La prueba es aplicada para diferentes tipos de objetivos, en diferentes escenarios o niveles de trabajo. Se distingues los siguientes niveles de pruebas:

- Prueba de desarrollador
- Prueba independiente
- Prueba de unidad
- Prueba de integración
- Prueba de sistema
- Prueba de aceptación

4.4.1 Pruebas de Unidad

Se trabajará sobre el nivel de *Prueba de Unidad,* es la prueba enfocada a los elementos más pequeño del software. Es aplicable a componentes representados en el modelo de implementación para verificar que los flujos de control y de datos están cubiertos, y que ellos funcionen como se espera. La prueba de unidad siempre está orientada a caja blanca. Para este método de caja blanca se usará específicamente la técnica del camino básico. Con esta prueba es posible evaluar el funcionamiento de la estructura interna del sistema, para verificar y revelar la calidad del mismo.

"Camino básico es un método que permite al diseñador de casos de prueba obtener una medida de la complejidad lógica de un diseño procedimental y usar esa medida como guía para la definición de un conjunto básico de caminos de ejecución. Los casos de prueba obtenidos del conjunto básico garantizaran que durante la prueba se ejecuta por lo menos una vez cada sentencia del programa" (21).

Los pasos del diseño de pruebas a seguir mediante el camino básico son:

1. A partir del diseño o del código fuente, se dibuja el grafo de flujo asociado.
2. Se calcula la complejidad ciclomática del grafo de flujo.
3. Se determina el conjunto básico de caminos independientes.
4. Se preparan los casos de prueba que obliguen a la ejecución de cada camino del conjunto básico.

Grafo de flujo

"Grafo de flujo o grafo del programa: este representa el flujo de control lógico de un programa y se utiliza para trazar más fácilmente los caminos de éste. (Cada nodo representa una o más sentencias procedimentales y cada arista representa el flujo de control)" (21).

Para la elaboración del mismo se utiliza los tres elementos siguientes:

- Nodos: representan cero, una o varias sentencias en secuencia. Cada nodo comprende como máximo una sentencia de decisión (bifurcación).
- Aristas: líneas que unen dos nodos.
- Regiones: áreas delimitadas por aristas y nodos. Cuando se contabilizan las regiones de un programa debe incluirse el área externa como una región más.

Complejidad Ciclomática

"Para contar el número de ciclos diferentes que se siguen en un fragmento de código de un programa, habiendo creado una rama imaginaria desde el nodo de salida al nodo de entrada se utiliza la Complejidad Ciclomática (Cyclomatic Complexity), esta es una métrica del software que proporciona una medición cuantitativa de la complejidad lógica de un programa. Es una de las métricas de software más ampliamente aceptada, ya que ha sido concebida para ser independiente del lenguaje" (21).

El resultado obtenido en el cálculo de la complejidad Ciclomática define el número de caminos independientes dentro de un fragmento de código y determina la cota superior del número de pruebas que se deben realizar para asegurar que se ejecuta cada sentencia al menos una vez.

Existen varias formas de calcular la complejidad ciclomática de un programa a partir de un grafo de flujo:

1. V (G)= Número de Regiones.
2. V (G)= Número de Aristas – Número de Nodos +2
3. V (G)= Número de Nodos Predicados +1

Camino independiente

Un camino independiente es cualquier camino del programa que introduce un nuevo conjunto de sentencias. El conjunto de caminos independientes de un grafo no es único. No obstante, a continuación, se muestran algunas heurísticas (término con el que nos referimos al método o procedimiento usado en la investigación o en el descubrimiento de algo) para identificar dichos caminos:

A. Elegir un camino principal que represente una función válida que no sea un tratamiento de error. Debe intentar elegirse el camino que atraviese el máximo número de decisiones en el grafo.
B. Identificar el segundo camino mediante la localización de la primera decisión en el camino de la línea básica alternando su resultado mientras se mantiene el máximo número de decisiones originales del camino inicial.
C. Identificar un tercer camino, colocando la primera decisión en su valor original a la vez que se altera la segunda decisión del camino básico, mientras se intenta mantener el resto de decisiones originales.

D. Continuar el proceso hasta haber conseguido tratar todas las decisiones, intentando mantener como en su origen el resto de ellas.

Derivación de casos de prueba

Luego de tener elaborados los grafos de flujos y los caminos a recorrer, se preparan los casos de prueba que forzarán la ejecución de cada uno de esos caminos. Se escogen los datos de forma que las condiciones de los nodos predicados estén adecuadamente establecidas, con el fin de comprobar cada camino.

Pruebas de unidad para el caso de uso Configurar el Sistema

A continuación se muestra en la figura 17 la implementación del método guardarSettings donde a partir del mismo de define el grafo de flujo del mismo observado en la figura 18.

```
bool Configurador::guardarSettings()
{
  1  QSettings settings("UCI", "Configurador");
  1  QString ip = ui->textIP->text();
  1  QStringList partes = ip.split(".");

  2  if(partes.length() == 4)
     {
  3      bool ok1 = true;
  4      for(int i = 0; i < partes.length(); i++)
         {
  5          bool ok;
  6          int dec = partes.at(i).toInt(&ok, 10);
  7          if(!ok)
             {
  8              ok1 = ok;
  8              break;
             }
  9      }
 10      if(ok1)
         {
 11          settings.setValue("gestor/ip", ui->textIP->text());
 11          settings.setValue("gestor/puerto", ui->spinPuerto->value());
 11          settings.setValue("plugins/ruta", ui->textRuta->text());
 11          return true;
         }
     }
 12   QMessageBox* box = new QMessageBox(QMessageBox::Critical, "Error",
                                         "Formato del IP incorrecto.",QMessageBox::Ok,this);
 12   box->show();
 12   return false;
}
```

Figura 17: Método "guardarSettings".

1. **Grafo de flujo**

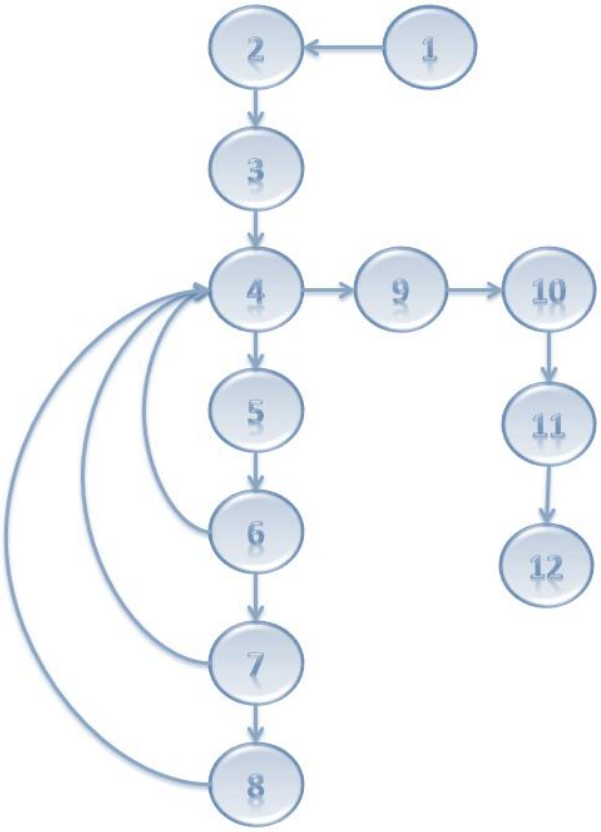

Figura 18: Grafo de Flujo para el método "guardarSettings"

2. **Cálculo de la complejidad ciclomática**

   V (G) = Número de Aristas – Número de Nodos +2

   V (G) = 14 – 12 + 2

   V (G) = 4

3. **Caminos independientes**

   1) 1-2-3-4-9-10-11-12
   2) 1-2-3-4-5-6-4-9-10-11-12
   3) 1-2-3-4-5-6-7-4-9-10-11-12
   4) 1-2-3-4-5-6-7-8-4-9-10-11-12

Pruebas de unidad para el caso de uso Cargar Video-Sensor para iniciar procesamiento

A continuación se muestra en la figura 19 la implementación del método Cargar donde a partir del mismo de obtiene el grafo de flujo del mismo observado en la figura 20.

```
void CargarPlugin::Cargar(QString tipo, QDir pluginsDir)
{
1    foreach (QString fileName, pluginsDir.entryList(QDir::Files))
     {
2        QPluginLoader pluginLoader(pluginsDir.absoluteFilePath(fileName));
2        QObject *objPlugin = pluginLoader.instance();
3        if (objPlugin)
         {
4            IPlugin *plug = qobject_cast<IPlugin *>(objPlugin);
5            if (plug)
             {
6                if(plug->getTipo() == tipo)
                 {
7                    plugin = plug;
7                    return;
                 }
             }
8        }
     }
9    plugin = NULL;
9    emit Me("No se encuentra disponible el plugin " + tipo + ".");
}
```

Figura 19: Método “Cargar(string tipo, QDir pluginDir)”.

1. Grafo de flujo

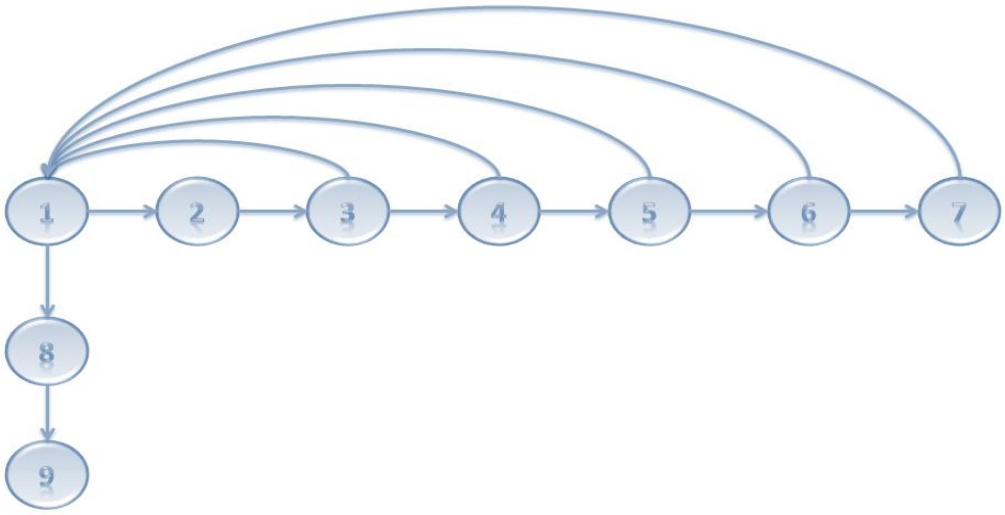

Figura 20: Grafo de Flujo para el método "Cargar (string tipo, QDir pluginDir)"

2. **Cálculo de la complejidad ciclomática**

   V (G) = Número de Aristas – Número de Nodos +2

   V (G) = 13 – 9 + 2

   V (G) = 6

3. **Caminos independientes**

   1) 1-8-9
   2) 1-2-3-1-8-9
   3) 1-2-3-4-1-8-9
   4) 1-2-3-4-5-1-8-9
   5) 1-2-3-4-5-6-1-8-9
   6) 1-2-3-4-5-6-7-1-8-9

Pruebas de unidad para el caso de uso Detener Procesamiento

A continuación se muestra en la figura 21 la implementación del método detenerHilo donde a partir del mismo de obtiene el grafo de flujo del mismo observado en la figura 22.

```
void Analytic::detenerHilo(int num)
{
    //if(num < gtHilos.count())
        //gtHilos.at(num)->terminate();
1    for (int i = 0; i < gtHilos.count(); i++)
    {
2        HiloProcesador *este = dynamic_cast<HiloProcesador*>(gtHilos.at(i));
3        if(este->getId() == num)
        {
4            este->terminate();
4            return;
        }
5    }
}
```

Figura 21: Método “detenerHilo (int num)”.

1. **Grafo de flujo**

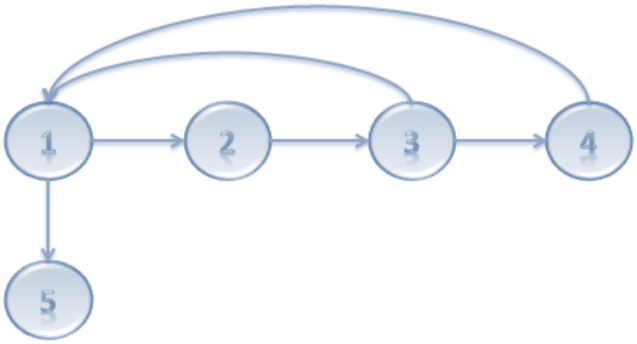

Figura 22: Grafo de Flujo para el método “detenerHilo (int num)”

2. **Cálculo de la complejidad ciclomática**

   V (G) = Número de Aristas – Número de Nodos +2

   V (G) = 6 – 5 + 2

   V (G) = 3

3. **Caminos independientes**

   1) 1-5
   2) 1-2-3-1-5

3) 1-2-3-4-1-5

Resultados obtenidos para los casos de prueba

- **Nombre del caso de uso:** Configurar el Sistema
- **Para el camino 4:**1-2-3-4-5-6-7-8-4-9-10-11-12
- **Caso de prueba:** Probando la función **guardarSettings()**
- **Resultado:** Permite configurar todos los parámetros de comunicación entre el Gestor y el Analytic.

- **Nombre del caso de uso:** Cargar Video-Sensor para iniciar procesamiento
- **Para el camino 5:** 1-2-3-4-5-6-8-9
- **Caso de prueba:** Probando la función **Cargar(string tipo, QDir pluginDir)**
- **Resultado:** Permite cargar el video-sensor de la demanda del sistema.

- **Nombre del caso de uso:** Detener Procesamientos
- **Para el camino 3:** 1-2-3-4-1-5
- **Caso de prueba:** Probando la función **detenerHilo(int num)**
- **Resultado:** Permite detener procesamientos.

No del camino.	Caso de Prueba.	Objetivo.	Resultado.
4	Probando la función guardarSettings()	Permite configurar todos los parámetros de comunicación entre el Gestor y el Analytic.	Satisfactorio.
5	Probando la función Cargar(string tipo, QDir pluginDir)	Permite cargar el video-sensor de la demanda del sistema.	Satisfactorio.
3	Probando la función detenerHilo(int num)	Permite detener procesamientos.	Satisfactorio.

Tabla 16: Casos de Prueba

4.5 Conclusiones

Con la realización de este capítulo se desarrollaron los flujos de trabajo Implementación y Prueba. Durante la etapa de implementación se crearon los artefactos correspondientes a la misma: se detalló la relación hardware y software del sistema a través del diagrama de despliegue, se describió la manera en que se organizan los componentes de software, así como la dependencia lógica entre los mismos mediante el diagrama de componentes.

Se llevaron a cabo pruebas de caja blanca con el objetivo de evaluar el cumplimiento de los requisitos funcionales definidos en los inicios del proceso de desarrollo, obteniendo como resultados satisfactorios de estas, por lo que se puede afirmar que el sistema satisface las necesidades para las que fue creado.

CONCLUSIONES

Una vez completado el proceso investigativo el autor concluyen lo siguiente:

1. La caracterización y revisión del estado del arte del tema de la investigación permitió afirmar que las soluciones que existen hoy, de alguna manera tributan a la investigación, pero no resuelven la problemática planteada, expresándose la necesidad del desarrollo de la propuesta.
2. Las herramientas, tecnologías y lenguajes que se proponen, en todos los casos, se corresponden con las políticas de soberanía tecnológica que impulsa la universidad y el país.
3. Todos los requisitos funcionales que se definieron fueron debidamente implementados, se incluyeron en la propuesta las exigencias de todos los requisitos no funcionales detectados.
4. La solución que se presenta es multiplataforma, lo que amplía las posibilidades de utilización y la gama de usuarios de la misma.
5. Los artefactos generados durante el proceso de desarrollo del software permitirán continuar escalando la solución en el futuro.
6. La utilización de estilos y patrones promueve buenas prácticas en el desarrollo de la solución al proporcionar uniformidad en la implementación, siendo más entendible y escalable en el tiempo. Por otra parte, tanto estilos como patrones, permiten ahorrar grandes cantidades de tiempo en la implementación.
7. Los diseños de casos de prueba desarrollados, como parte de las pruebas de caja blanca, permitieron validar los requisitos de la aplicación con las funcionalidades implementadas.
8. La utilización de este sistema, le permitirá al proyecto Video Vigilancia Suria en Qt, perteneciente al Departamento de Señales Digitales, realizar análisis inteligente a través de flujos de videos.

RECOMENDACIONES

El autor recomienda:

1. Incluir como parte de la solución la posibilidad de crear más plugin para ser cargados por el sistema Analytic.

BIBLIOGRAFÍAS CITADAS

1. **Sales, Ricardo Cañizares.** 30,

2. **Krutchet, Philippe.** *Rational Rose.*

3. **Española, Diccionario de la Real Academia.** [En línea] [Citado el: 21 de 11 de 2011.] http://buscon. rae. es/.

4. Intelligent Security Systems. [En línea] [Citado el: 20 de 11 de 2011.] http://www. isscctv. com/upload/iblock/bb3/SecurOS_Professional_Spanish. pdf.

5. SecurOS. [En línea] [Citado el: 16 de 11 de 2011.] http://www. isstechnology. eu/productos/securos.

6. Axis. [En línea] [Citado el: 10 de 11 de 2011.] http://www. axis. com/es/products/video/about_networkvideo/iv/products. htm.

7. Solutions, Siqura Surveillance. [En línea] [Citado el: 25 de 11 de 2011.] http://www. siqura. com/. .

8. **Rammer, Ingo and Szpuszta, Mario.** *Advanced .NET Remoting. s. l. : Apress. .* 2005.

9. **Deivi, Edmis.** *Sistema de Video Vigilancia. .* 2009.

10. **Isaías Carrillo Pérez, otros.** *metodologia de desarrollo del software. .*

11. **Pockin.** 2008.

12. **Orallo, Enrique Hernández.**

13. *Ingeniería de Softaware I (VI Ciclo).*

14. **Rumbaugh, otros.** 2000.

15. Definición. org. [En línea] [Citado el: 20 de 02 de 2012.] http://www. definicion. org/lenguaje-de-programacion. .

16. **Jacobson.** 2000.

17. *Pressman.* 2005.

18. **Reynoso, Carlos.** *Introducción a la Arquitectura de Software.* 2004.

19. *Aldana.* 2009.

20. **Cuesta. , Carlos E.** *Arquitecturas del Software. .* Universidad Rey Juan Carlos. : s. n.

21. **Arias. , Marvin David.** Editboard. com. [En línea] 16 de 10 de 2008. [Citado el: 05 de 03 de 2012.] http://catedraprogramacion. foroactivo.NET.

BIBLIOGRAFÍAS CONSULTADAS

- Salvador Alemany. 2009. (Framework Qt)
- Alain D. Osorio Rojas. 2008 (Qt 4 Manual introductorio)
- Garcia de Jalon, y otros. 1998.
- Pérez, Chaves. 2010.
- Jacobson
- Guía de Videovigilancia
- [En línea] [Citado el: 25 de 04 de 2012.] http://www. camarasip. cl/seguridad_y_vigilancia_por_internet. htm. Cámaras IP.

1