UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS

FACULTAD 6

"Estación de Monitorización para Video Vigilancia."

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

Autor: Rayner Pupo Gómez

Tutor: Ing. Heliodoro Rodríguez Milián

Ciudad de La Habana, junio de 2011

"Año 53 de la Revolución"

DECLARACIÓN DE AUTORÍA

Declaro ser el único autor del presente trabajo y autorizo a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

Para que así conste firmamos la presente a los ___ días del mes de _____ del año 2011.

Autor:

_____

Rayner Pupo Gómez

Tutor:

_____

Ing. Heliodoro Rodríguez Milián.

DATOS DE CONTACTO

Tutor:

Ing. Heliodoro Rodríguez Milián (hrodriguezm@uci.cu)

Graduado de Ingeniero en Ciencias Informáticas en la UCI en el año 2007; es Profesor Instructor, durante su trayectoria como profesor ha impartido las asignaturas Gráfico por Computadoras, Programación Web, Práctica Profesional 1 e Historia de la Informática. Trabajador del GEYSED en el Proyecto Sistema de Video Vigilancia en el cual se desempeña como Líder de Proyecto.

RESUMEN

RESUMEN

La presente investigación tiene como objetivo la implementación de una estación de monitorización teniendo en cuenta el uso de tecnologías libres. Para su desarrollo se propone el uso de Herramientas y librerías libres y multiplataforma.

Se analizan características de sistemas similares existentes, además se exponen las tecnologías, herramientas y metodologías que fueron utilizadas en el desarrollo de la solución, se mencionan los principales componentes utilizados y los patrones de diseño empleados. Por último se muestran los resultados de las pruebas realizadas al sistema.

Palabras Claves:

Sistemas, video vigilancia, seguridad

ABSTRACT

ABSTRACT

This research intends to implement a monitoring station taking into consideration the use of free technologies. For its development proposes the use of free and multiplatform libraries and tools.

Are discussed features from existing similar systems also are exposed the technologies, tools and methodologies that were used in the development of the solution, also are mentioned the principal components and design patterns used. Finally we present the results of the tests performed to the system.

Keywords:

Systems, video surveillance, security

Contenido

CONTENIDO

ÍNDICE DE FIGURAS

INTRODUCCIÓN

Las formas de protección de recursos valiosos siempre han sido muy variadas y se encuentran en constante evolución, siendo la más usual y tradicional el uso exclusivo de personas que se dedican a la supervisión directa hacia el medio a proteger. De esta forma se hace necesario el empleo de mayor cantidad de fuerza laboral a medida que aumenta la zona a custodiar, además esta variante trae consigo errores de naturaleza humana los cuales pueden ser que un custodio se quede dormido o sufra algún tipo de accidente o percance.

Con el surgimiento y desarrollo de nuevas tecnologías surge una nueva forma de protección llamada video vigilancia que reduce la mayoría de los errores de antiguos métodos, logrando una mayor seguridad en el ámbito donde se encuentre. Las ventajas son amplias en cuanto a la reducción del personal de guardia, así como en la posibilidad de monitorizar varias áreas de forma simultánea y centralizada. Otra característica esencial es la posibilidad de grabar lo acontecido en los turnos de guardia para la posterior visualización, en busca de información relevante acerca de cualquier incidente. (1)

Cuba no se encuentra ajena a los problemas que ocasiona la poca efectividad de los métodos más antiguos de custodia de recursos principales de la empresa. Con la adquisición de tecnologías costosas en campos como la salud, la educación, la industria, se hace necesario un estricto control de estas para evitar pérdidas materiales; esto no puede ser logrado, o por lo menos no efectivamente, mediante el uso exclusivo de personas, por tanto se hace necesaria la implantación de un sistema de video vigilancia, el cual reduciría los gastos de la empresa en materia de seguridad a largo plazo. Con el uso de este sistema la cantidad de personal encargado a la custodia se reduciría considerablemente pues solo se hace necesario el empleo de personas en las estaciones de monitorización y en puntos específicos donde puedan responder rápidamente ante una alarma.

La situación anteriormente planteada lleva al problema a resolver ¿Cómo obtener una estación de monitorización funcional a partir de los requisitos identificados para el producto Suria Viewer utilizando software libre? Teniendo como objeto de estudio la seguridad de las instituciones del país. Se delimita como campo de acción la informatización de la video vigilancia en instituciones del país.

Para darle solución al problema planteado anteriormente, se tiene como objetivo general: Desarrollar una herramienta para la monitorización de flujos de video de cámaras IP para el sistema Suria. Se persigue tener como resultado un sistema que permita la vigilancia efectiva y centralizada en instituciones del país.

Para el cumplimiento del objetivo planteado se proponen las siguientes tareas de investigación:

1. Caracterizar los sistemas de video vigilancia existentes en la actualidad.
2. Caracterizar herramientas y tecnologías libres existentes para aplicaciones de escritorio y seleccionar las que se utilizarán.
3. Implementar las funcionalidades necesarias para la estación de monitorización.
4. Desarrollar los casos de prueba que verifiquen la correcta implementación del sistema.
5. Documentar los resultados de las pruebas funcionales realizadas al sistema.

Los métodos científicos utilizados en la investigación fueron:

Métodos Teóricos.

- Analítico-sintético: Es usado para estudiar y analizar documentos y bibliografías de diferentes autores con el objetivo de poder realizar una amplia investigación sobre los elementos que se relacionan con el objeto de estudio.
- Análisis histórico lógico: Se utiliza para hacer un estudio sobre la evolución y desarrollo que han tenido los sistemas de video vigilancia.

Métodos Empíricos.

- Observación: Permite realizar valoraciones y obtener informaciones a partir de lo observado sobre el funcionamiento de sistemas similares al que se desea desarrollar, lo que proporciona una visión de cómo tiene que ser el sistema a realizar en su forma externa.

INTRODUCCION

El presente documento consta de dos capítulos:

Capítulo 1: Fundamentación teórica.

Describe algunas de las características de las diferentes estaciones de monitoreo de flujos de Video Digital, surgimiento y desarrollo de estos, así como una referencia al estado del arte de los proyectos existentes en el mundo, semejantes al que se va a desarrollar. Además, se realiza un análisis de las tecnologías y herramientas actuales, seleccionando las que se van a utilizar para el desarrollo del sistema.

Capítulo 2: Implementación y pruebas de la solución.

Se presenta el estilo de codificación y documentación que será aplicado durante la implementación del sistema. Además se brindan muestras de código fuente relacionadas con el patrón de diseño que representa, perteneciente a las principales funcionalidades de la aplicación. Por último se presentan los resultados arrojados por las pruebas realizadas a la aplicación.

CÁPITULO1: FUNDAMENTACIÓN TEÓRICA.

Introducción

En este capítulo se abordarán los principales aspectos acerca de la evolución de los sistemas de video vigilancia. Se brinda una visión general del estado del arte en que se encuentran los sistemas similares tanto en la Universidad de las Ciencias Informáticas, en Cuba, y en el resto del mundo. Además se hace un análisis de las herramientas y tecnologías a usar para la implementación del sistema.

1.1 **Evolución de los sistemas de video vigilancia**.

Desde los inicios de los sistemas de video vigilancia, éstos han ido diversificándose tanto en los ambientes donde son usados como en las tecnologías que poseen. Una de las principales cualidades que los caracterizan es la capacidad de escalabilidad, lo que permite ajustarlos a todo tipo de situación, desde un hogar con pocas cámaras hasta un sistema con cientos de cámaras como los de vigilancia de autopistas. En la mayoría de las empresas el valor de los bienes a preservar excede ampliamente al costo de implantación de un sistema de video vigilancia, por lo que resulta una variante a tener en cuenta muy económica.

Con los avances de la tecnología en la televisión surge una primera generación de sistemas de video vigilancia. Esta consta de un conjunto de cámaras analógicas dispersas en un recinto, de las cuales se obtiene una señal que es multiplexada y enviada a un VCR[1] y a un monitor analógico directamente, el operador cuenta con un conmutador que le permite elegir cuál o cuáles señales monitorizar. Con este sistema se hace evidente el uso de una videoteca para almacenar las grabaciones, lo que aumenta significativamente el espacio que se requiere además de que estas grabaciones son hechas sobre cinta las cuales son susceptibles a campos magnéticos (1). La forma en que las grabaciones son usadas posteriormente en búsqueda de hechos delictivos es un trabajo engorroso que involucra la reproducción de varias horas de material.

La instalación y mantenimiento de un sistema de primera generación tiene inconveniencias en cuanto a sistemas posteriores. Una de estas es la infraestructura de cableado coaxial que requiere, la decisión de mover una cámara de un lugar a otro conlleva al reposicionamiento de los cables e incluso de los multiplexores.

[1] Acrónimo del inglés video cassette recorder.

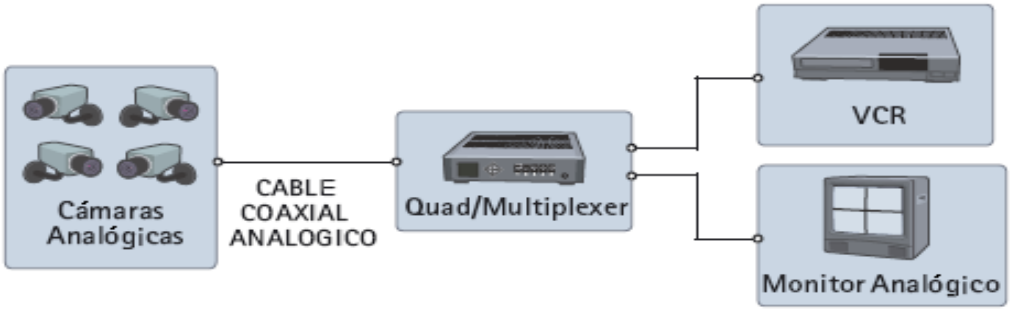


Figura 1: Sistema de Video Vigilancia Analógico.

Con el desarrollo de la tecnología digital surge la segunda generación de los sistemas de video vigilancia, los cuales pueden ser de dos tipos fundamentales, los digitales y los híbridos analógicos-digitales, los cuales presentan compatibilidad con antiguos sistemas. Por primera vez se utiliza software para la visualización y control de cámaras, constituyendo las estaciones de monitoreo. El control de cámaras se hace más fácil mediante el uso de cámaras PTZ[2], que son un ejemplo de la integración que existe en los sistemas de monitorización digital con la infraestructura de red existente en una empresa (2).

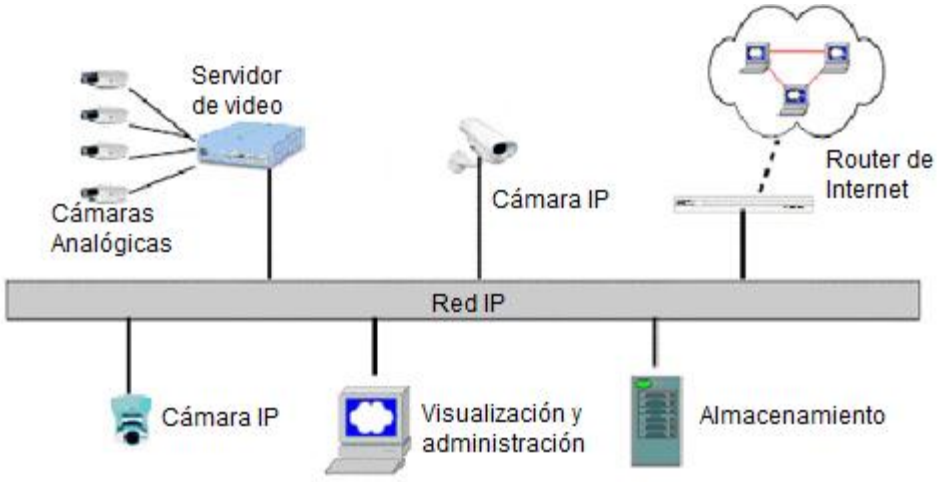


Figura 2: Sistema de Video Vigilancia Mixto Analógico-Digital.

[2] pan-tilt-zoom (rotación horizontal, rotación vertical y acercamiento).

Con el uso de esta nueva estructura se reducen los costos de implantación, mantenimiento y operación de los sistemas de vigilancia. Mediante el uso de cámaras IP se puede conectar éstas a cualquier toma RJ45 sin necesidad de cambiar el cableado. El almacenamiento de los videos se realiza en discos duros de forma descentralizada lo que permite un mayor volumen de información en un menor espacio, además de una mejor calidad de imagen mediante el uso de una amplia gama de normas de compresión como JPEG, JPEG2000, MPEG-1, 2, 4, Wavelet y H.261/H.263(1). Estas normas posibilitan una mejor relación entre la taza de transferencia y la calidad de imagen (3).

1.2 Estado del Arte de las estaciones de monitoreo de los sistemas de video vigilancia.

Actualmente existe una amplia gama de empresas que realizan productos dedicados a la video vigilancia, la mayoría de estas empresas tienen como compradores a grandes compañías y sectores del gobierno. En organizaciones de uso privado como son los restaurantes y las tiendas de comestibles, es usual que se utilicen los circuitos cerrados de televisión con cámaras tanto en el interior como en el exterior de los edificios, éstas son aplicadas para tener respuesta en caso de cualquier movimiento que levante sospechas, además es un método para que los gerentes de las tiendas, puedan observar en tiempo real, el grado de aceptación de sus mercancías y los puntos de venta en específico. Los dispositivos de seguridad que con el avance de la tecnología se han vuelto de tamaño más reducido, son muy eficaces y poseen una claridad del circuito cerrado de televisión aumentada, que cede a los usuarios eficientemente el paso de información visual en datos de utilidad (4).

1.2.1 Principales Productos de video vigilancia desarrollados en la UCI.

Sistema de Video Vigilancia Suria.

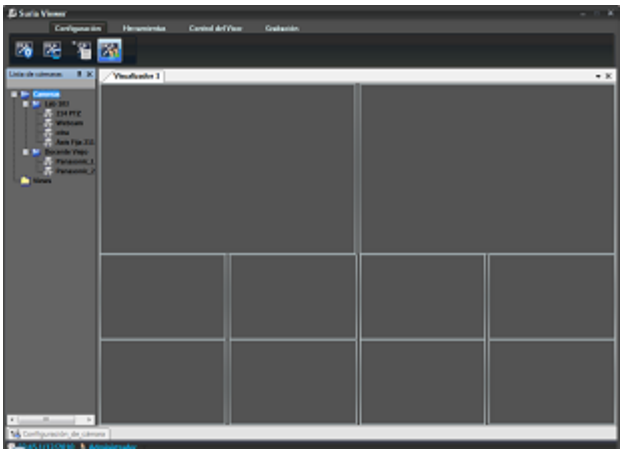


Figura 3: Sistema de video vigilancia Suria.

Propiedades:

- Utiliza las librerías de desarrollo .NET
- Posee un sistema de plugins para la incorporación de nuevos tipos de cámaras.
- Posee múltiples tipos de vistas.
- Posibilita la interacción con cámaras PTZ.
- Posibilita la grabación de secuencias de video.

Esta aplicación cuenta con numerosas características que la hacen eficiente en términos de usabilidad, además cuenta con una amplia lista de plugins para cámaras, pero presenta como desventaja su acoplamiento al sistema operativo Windows, por lo que no es posible su instalación en lugares donde se usen diferentes sistemas operativos.

1.2.2. Principales empresas cubanas productoras de sistemas de vigilancia:

Figura 4: Logo de la empresa DATYS.

Producto: Xyma Safe Vision.

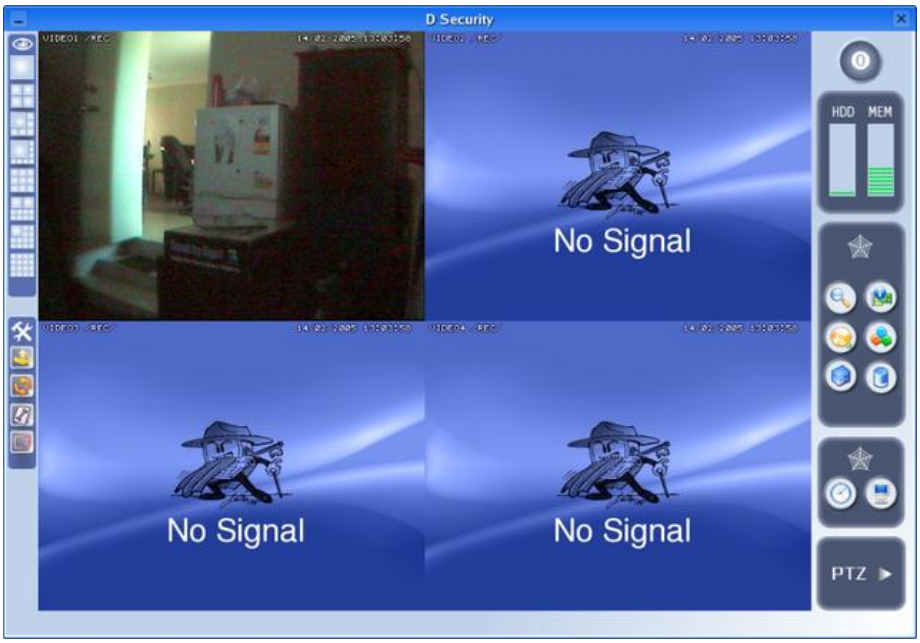
Figura 5: Logo del producto Xyma Safe Vision.

Xyma Safe Vision es un software de video vigilancia profesional basado en tecnología IP con un alto grado de modularidad, adaptable, flexible y escalable, que permite el uso de tecnologías analógicas. Administra la seguridad monitorizando y controlando en tiempo real y de forma histórica cada uno de los movimientos que ocurren en las áreas sensibles que se identifiquen. Las posibilidades del sistema están distribuidas para darle flexibilidad y su arquitectura modular lo convierte en un potente sistema que permite administrar, monitorizar, grabar, revisar, configurar alertas y alarmas y obtener reportes para cualquier solución de video vigilancia que se requiera implementar (5).

A pesar de la gran variedad de prestaciones que posee esta aplicación, este producto está destinado solo a plataformas Windows.

1.2.3. Principales productos basados en software libre en el ámbito internacional.

iDVR[3] (6)



Fígura 6: Interfaz Principal de iDVR

Propiedades:

- Manejo de hasta 16 cámaras.
- Detección de movimiento.
- Grabación al detectar movimiento.
- Grabación de video MPEG4 hasta de 25 cuadros por segundo.
- Multidifusión en vivo hacia la red local.
- Unidifusión hacia la Internet
- Múltiples temas de X11.
- Interfaz basada en la web.

Este sistema cuenta con funcionalidades que lo hacen utilizable en lugares con un número reducido de cámaras. Limita las posibilidades de grabación hasta una tasa de 25 cuadros por segundo, cuando la

[3] El software iDVR es formalmente conocido como Devolution Security.

mayoría de las cámaras IP pueden emitir a una frecuencia de 30 cuadros por segundo en flujos MPEG4. Solo puede ser ejecutado en entornos Linux.

1.2.4. Resumen:

Con la actual migración que se está desarrollando en Cuba hacia sistemas operativos y tecnologías libres, los sistemas Video Vigilancia Suria y Xyma Safe Vision no son aplicables en un gran sector de las empresas. Por lo tanto los lugares donde estas aplicaciones son instaladas deben cambiar su propia política de migración y permitir que permanezcan algunas estaciones de trabajo con el sistema privativo Windows, o estos sistemas de video vigilancia quedarían en desuso. Por otra parte, existen lugares donde la migración no va a tener lugar en un futuro cercano por lo que el sistema iDVR tampoco puede ser utilizado.

1.3 Metodología, herramientas y tecnologías a utilizar para el desarrollo de la solución.

1.3.1 Metodología de desarrollo.

Proceso Unificado de Desarrollo de Software (RUP por sus siglas en inglés).

RUP es una infraestructura flexible de desarrollo de software que proporciona prácticas recomendadas probadas y una arquitectura configurable. Permite escoger fácilmente el conjunto de componentes de proceso que se ajustan a las necesidades específicas del proyecto. Se podrán alcanzar resultados predecibles unificando el equipo con procesos comunes que optimicen la comunicación y creen un entendimiento común para todas las tareas, responsabilidades y artefactos. Los aspectos que definen el Proceso Unificado se resumen en tres características: dirigido por casos de uso, centrado en la arquitectura, e iterativo e incremental (7).

1.3.2 Tecnologías a usar para el desarrollo del sistema.

1.3.2.1. Lenguaje de programación: C++

Ventajas:

- Difusión: al ser uno de los lenguajes más empleados en la actualidad, posee un gran número de usuarios y existe una gran cantidad de libros, cursos, páginas web, etc., dedicados a él.

- Versatilidad: C++ es un lenguaje de propósito general, por lo que se puede emplear para resolver cualquier tipo de problema.
- Portabilidad: el lenguaje está estandarizado y un mismo código fuente puede ser compilado en diversas plataformas.
- Eficiencia: C++ es uno de los lenguajes más rápidos en cuanto a ejecución.
- Herramientas: existe una gran cantidad de compiladores, depuradores, librerías, etc. desarrolladas para este lenguaje (7).

1.3.2.2. Librerías de desarrollo: Qt

Qt es una librería de desarrollo multiplataforma que permite la creación de aplicaciones con interfaz gráfica o de consola (8).

Sus características son:

- Presenta una librería de clases de C++ intuitiva.
- Es portable desde sistemas de escritorio hasta sistemas operativos embebidos.
- Herramientas y entornos de desarrollo multiplataforma.
- Alto rendimiento en todas las plataformas.

Características de la **interfaz gráfica** de usuario de Qt:

- Gran cantidad de controles, desde botones y diálogos hasta tablas.
- Distribución y representación automática de fuentes, lenguajes y orientación.
- Soporte para gráficos vectoriales.
- Uso de hojas de estilo[4] en los componentes gráficos.
- Soporte para aceleradores de hardware y escritorios múltiples en sistemas embebidos.

Características de la **programación multihilo** en Qt:

- Funcionalidades para el fácil manejo de hilos, datos y objetos.

[4] Comúnmente llamadas CCS.

- Comunicación segura entre objetos a través de hilos usando el mecanismo de signals-slots.
- Una API de alto nivel que hace posible escribir programas multi-hilos sin usar primitivas de bajo nivel.

Características de las librerías **multimedia** de Qt:

- Trabajo con contenido multimedia independiente de la plataforma.
- Capacidad de leer archivos tanto localmente como flujos a través de la red.
- Abstracción de QuickTime® en Mac, DirectShow® en Windows y GStreamer en Linux.

Características del acceso a la **red** usando Qt:

- Abstracción completa de los sockets del cliente y el servidor.
- Soporte para HTTP, FTP, DNS y HTTP 1.1 asíncrono.
- Acceso a todo tipo de datos, desde HTML y XML hasta archivos de imágenes y medias.

Características del trabajo con **bases de datos** con Qt:

- Soporte para la mayoría de los tipos de bases de datos, ODBC, MySQL, PSQL, SQLite, ibase, Oracle, Sybase, DB2.
- Presentación de datos en una gran variedad de vistas.

Por las características antes expuestas se decide utilizar las librerías de **Qt** como el principal marco de trabajo a utilizar.

1.3.3 Herramientas.

Elección del entorno integrado de desarrollo.

Teniendo en cuenta el lenguaje elegido y las librerías de desarrollo propuestas se tienen tres principales IDEs[5] candidatos a utilizar.

1.3.3.1. KDevelop

[5] Entorno Integrado de Desarrollo.

Surgió en 1998 con el fin de desarrollar un IDE fácil de usar para KDE (K Desktop Environment). Desde entonces está públicamente disponible bajo licencia GPL y soporta lenguajes de programación como: C, C++, Java, Ada, SQL, Python, Perl y Pascal. Solo se ejecuta en sistemas GNU/Linux y otros sistemas Unix. Su última versión es la 3.5.5 y salió el 5 de agosto del 2009. Tiene como limitantes principales que su entorno gráfico es algo pobre y sólo corre sobre plataforma GNU/Linux (9).

1.3.3.2. Eclipse:

Es un IDE multiplataforma desarrollado por IBM. En la actualidad lo mantiene la Fundación Eclipse, una organización independiente sin ánimo de lucro que fomenta una comunidad de código abierto y un conjunto de productos, capacidades y servicios complementarios. Pese a que está escrito en su mayor parte en Java (salvo el núcleo), se ejecute sobre máquina virtual de ésta y su uso más popular sea como un IDE para Java, Eclipse es neutral y adaptable a cualquier tipo de lenguaje, por ejemplo C/C++, Cobol, C#, XML, etc. Sus mayores ventajas radican en su gran comunidad de desarrollo que lo ubica como el mejor IDE Java.

1.3.3.3. QtCreator

Es un IDE creado por Trolltech para el desarrollo de aplicaciones con las bibliotecas Qt, utilizando su versión 4.x. Los sistemas operativos que soporta en forma oficial son: GNU/Linux 2.6.x, para versiones de 32 y 64 bits con Qt 4.x instalado, además hay una versión para GNU/Linux con gcc 3.3. . Mac OS X 10.4 o superior, requiriendo Qt 4.x. Windows XP y Vista, requiriendo el compilador MinGW y Qt 4.4.3 para MinGW. Este IDE tiene la ventaja de ser soportado por la misma compañía que da soporte a Qt, por lo que propicia una mayor integración y aumenta el número de funcionalidades (5).

Teniendo en cuenta las características de los tres IDEs anteriormente expuestas se propone la utilización del QtCreator, este proporciona una gran facilidad de uso además de que posee un amplio soporte por parte de los desarrolladores de la propia librería Qt. En adición a esto QtCreator posee una integración a los principales sistemas de control de versiones como son Subversion y Git. Este IDE es multiplataforma por lo que puede ser usado en varios sistemas operativos, además de soportar diversos compiladores (8).

1.3.3.4 Generador de documentación Doxygen.

Doxygen es un sistema generador de documentación a partir de código fuente. Soporta los lenguajes C++, C, Java, Objetive-C, Python, IDL, Fortran, VHDL, PHP, C#, y algunas extensiones de D.

Puede generar documentos HTML y manuales de referencia LaTex desde un grupo de documentos con código fuente. También es posible generar archivos RTF (MS-WORD), PostScript, PDF con vínculos, HTML comprimido y manuales Unix (man). La documentación es extraída directamente desde el código por lo que es mucho más fácil mantener la consistencia con el código fuente.

Aunque Doxygen está desarrollado en los sistemas operativos Linux y Mac OS X, es portable para la mayoría de los sistemas basados en Unix, además están disponibles ejecutables para Windows. (11)

1.3.4. Sistema de control de versiones:

1.3.4.1. Subversion:

Sistema diseñado para remplazar al popular CVS, cuenta con una amplia gama de características como envíos atómicos y versionado de metadatos e integración con servidores como Apache.

1.3.5. Técnicas a utilizar:

El sistema cuenta con una estructura basada en componentes lo que le permite obtener una mayor escalabilidad. El proceso de la instalación de una cámara de un nuevo fabricante no implica la re implementación de todo el sistema sino que solo se debe desarrollar un nuevo plugin para dicha cámara y agregarlo a las librerías existentes, donde es cargado en tiempo de ejecución para no afectar la disponibilidad del sistema.

1.3.6. Tecnologías utilizadas.

1.1.6.1. SSL Sockets.

SSL[6] es un proceso que administra la seguridad de las transacciones que se realizan a través de Internet. El estándar SSL fue desarrollado por Netscape, junto con Mastercard, Bank of America, MCI y Silicon Graphics. Se basa en un proceso de cifrado de clave pública que garantiza la seguridad de los datos que se envían a través de Internet. Su principio consiste en el establecimiento de un canal de comunicación seguro (cifrado) entre dos equipos (el cliente y el servidor) después de una fase de autenticación (12).

[6] Secure Socket Layers o en español, Protocolo de Capa de Conexión Segura.

En el sistema de Video Vigilancia Suria en su versión Qt, la utilización de SSL Sockets hace posible la comunicación segura entre la aplicación Gestor y la Estación de Monitorización. Este proceso inicia con la creación de certificados de seguridad, estos son firmados e instalados en las estaciones de trabajo, de modo que se hace necesario contar con dichos certificados o la conexión no se puede establecer. El uso de este tipo de tecnología no implica que resulte más difícil la implementación de la mensajería entre aplicaciones, pues esto generalmente ocurre de forma transparente al usuario entre la capa de transporte y de aplicación.

1.4 CONCLUSIONES PARCIALES.

En este capítulo, luego de analizar las principales compañías que fabrican sistemas que incluyen una estación de monitoreo para el trabajo con los flujos de video, se concluyó que las compañías que comercializan este tipo de aplicaciones no tienen en cuenta más de una plataforma de despliegue, lo que restringe el uso de sus aplicaciones a un solo sistema operativo. Por otra parte se comprobó la existencia de estaciones de vigilancia basadas en la web, las cuales son por norma multiplataforma pero no poseen el rendimiento y prestaciones que pueden tener las aplicaciones de escritorio. Se tuvo en cuenta el uso de herramientas y tecnologías libres para reducir el costo de producción de la estación de monitorización.

CAPÍTULO 2: IMPLEMENTACIÓN Y PRUEBAS DE LA SOLUCIÓN.

2.1. Componentes de la solución.

- OpenSSL: Provee una serie de algoritmos de encriptación y cifrado usados por otros módulos en el sistema.
- QSslSocket: Utiliza las librerías de OpenSSL para transmitir información entre dos terminales de forma segura a través de sockets. Toda la información es cifrada con claves hasta de 1024 bits, por lo que hace imposible la recuperación de información en un canal inseguro.
- QtCore: Contiene todas las clases utilizadas por el framekork Qt que no están relacionadas con la interfaz gráfica, desde el espacio de nombres de trabajo multi-hilo, hasta los tipos definidos por Qt[7].
- QtNetwork: Posee las clases necesarias para el trabajo mediante la red, de forma fácil y portable.
- QtGUI: Este módulo extiende las posibilidades de QtCore, brindado un conjunto de controles que facilitan la interacción de la aplicación con el usuario.
- Libvlc: Las librerías de desarrollo libvlc proveen a la aplicación con una forma de representar los flujos de video provenientes de cámaras IP.

2.2 Patrones de diseño empleados (11).

- Patrón Visitador: Es usado para separar el código de la “visita” y el código del objeto visitado, el cual se encuentra en una colección.

```
QListIterator<Slot*> it(activeView->Slots());
while(it.hasNext())
{
    slt = it.next();
    rect = slotToRect(slt);
    c = highlightedSlot == slt ? QColor(79,79,79) : QColor(70, 70, 70);
    p.fillRect(rect.x(), rect.y(), rect.width(), rect.height(), c);
    p.drawRect(rect.x(), rect.y(), rect.width(), rect.height());
}
```

Figura 7 Código ejemplo del patrón Visitador.

[7] Ejemplo: QString, QChar, QByteArray, QDate, etc.

- Patrón Serializador: Se crea un objeto destinado a leer y escribir otros objetos o flujos.

```
void ClientSocket::incomingData()
{
    QString dta;
    while(!atEnd())
    {
        dta.append(readAll());
    }
    QStringList lst = dta.split(":");
    emit messageArrived(lst.first().toInt(), lst.at(1));
}
```

Las instancias de la clase ClientSocket tienen la responsabilidad de leer los flujos de la red

Figura 8 Código ejemplo del patrón Serializador.

- Patrón MetaObject o Reflection: Es usado para describir características de un objeto, las cuales son encapsuladas en otro objeto llamado Meta Objeto. En el caso de Qt, QMetaObject.

- Patrón Fabricador: Se responsabiliza a una clase de la creación de objetos de otro tipo.

```
VisorWidgetInterface *CamManager::getVisorFromCamId(int resId)
{
    QString model;
    if(GestorManager::resId_resData.contains(resId))
    {
        ResData *cd = GestorManager::resId_resData[resId];
        model = cd->Data().value("model", "");
        ICam *cam = camObjects.value(model, 0);
        if(cam != NULL)
        {
            if(cam->externalRenderName() == "VLC")
            {
                cam->setExternalRenderer(new VlcRenderer);
            }
            VisorWidgetInterface *vwi = cam->getVisorWidget(cd->Data());
            if(vwi)
            {
                setId(vwi);
                visorId_resId[vwi->Id()] = resId;
                return vwi;
            }
        }
    }
    return NULL;
}
```

La función getVisorFromCamId "fabrica" instancias de tipo VisorWidgetInterface

Figura 9 Código ejemplo del patrón Fabricador.

- Patrón Singleton: Restringe que solo pueda haber una instancia de una clase dada.

```
ClientSocket * NetworkManager::getMainClient()
{
    if(mainClient)
        return mainClient;
    else
        return getNewClient();
}
```

Figura 10 Código ejemplo del patrón Singleton.

- Patrón Composición: Se usa para crear objetos complejos a partir de objetos más simples, generalmente a partir de la herencia.

- Patrón Observador: Es generalmente usado por programas manejados por eventos en el caso de Qt por el mecanismo de señales y slots.

```
class InfoPanel : public QFrame
{
    Q_OBJECT
public:
    explicit InfoPanel(QWidget *parent = 0);
    void setData(const QHash<QString, QString> &data);
signals:
    void modifyRequest(int resId);
public slots:
    void modify();
```

Figura 11 Código ejemplo del patrón Observador.

- Patrón Comando: Se utiliza para encapsular una acción en un objeto, lo que hace posible su adición a una cola de acciones o su registro.

2.3 Estándar de codificación.

2.3.1 Nombres

- Los nombres de las clases son sustantivos singulares.

- Los nombres de clases y objetos deben reflejar qué hacen y no cómo lo hacen.
- Escoger nombres lo suficientemente largos que expresen correctamente el sentido de lo que se quiere, pero evitando manejar nombres que dificulten la labor de implementación.
- Evitar nombres que permitan una interpretación subjetiva (evitar ambigüedad y asegurar la abstracción).
- Evitar redundancia no repitiendo nombres de clases en sus elementos.
- Dado que los nombres generalmente son el producto de concatenar varias palabras, se debe emplear la primera palabra en minúscula, mayúscula para denotar la letra de inicio de cada una de las palabras restantes por las que esté formado y minúscula para las letras intermedias en el caso de los nombres de métodos y funciones. Para el caso de los nombres de variables y atributos debe aplicarse la misma convención. (camel case).
- Las variables booleanas deben contener la palabra is en su nombre.
- Los nombres de constantes deben contener solo letras mayúsculas.
- Minimizar el uso de abreviaturas. En caso de ser requeridas, se debe ser consistente en su uso y cada abreviatura debe significar solo una cosa.
- Los nombres de los métodos son frases que incluyen verbos.
- Los nombres de los atributos y parámetros son frases con sustantivos.

2.3.2 Codificación

- Se establece un tamaño de indentación estándar de cuatro espacios, sin tabulaciones.
- Alinear secciones del código.
- Alinear verticalmente llaves de apertura y cierre.
- Usar espacios antes y después de los operadores que el lenguaje de programación permita.
- Emplear líneas en blanco para organizar el código, permitiendo crear párrafos de código para una mejor lectura.
- Evitar colocar más de una sentencia por línea.
- Emplear constantes en sustitución de números o cadenas de caracteres literales.
- Minimizar el alcance de las variables para evitar confusión y facilitar el mantenimiento.
- Emplear cada variable y rutina solo para un propósito.
- Evitar el uso de variables públicas, sustituirlas por variables privadas y métodos que provean el valor de tal variable, para mantener el encapsulamiento.
- Emplear las letras i, j, k, l, m, p, q, r para contadores en ciclos.

- Mantener la modularidad del código bajo el criterio de la lógica que encierra, no exagerar la modularidad.
- Emplear correctamente los tipos de ciclos: si es al menos una vez usar do-while, si es ninguna o más veces usar while-do, y si se conoce el número exacto de ciclos usar for.
- Usar la sentencia for en sustitución de foreach, a menos que dificulte la implementación.
- Inicializar todas las variables.
- Inicializar todos los punteros en 0 o NULL preferentemente.
- Emplear líneas en blanco para separar pasos lógicos (Ej.: declaraciones, lazos).

2.3.3 Estándar de documentación.

Los principales formatos utilizados para la documentación utilizando la herramienta Doxygen son mostrados a continuación.

- Estilos de bloques de documentación:

Se adopta el estilo de comentario de bloques de Qt, el cual agrega luego de la apertura de un bloque de comentarios estilo C un signo de exclamación (!). Opcionalmente se puede comenzar cada línea dentro del comentario con el símbolo *. Ejemplo:

```
/*!
*. . .texto. . .
*/
```

Para hacer una descripción breve se adopta el uso del comando \brief o @brief en el bloque de comentarios ya descrito. La acción de este comando termina al final de un párrafo, de tal manera que la descripción detallada sigue después de una línea vacía, tal como lo muestra el ejemplo.

```
/**
 * @brief descripción breve.
 * Continuación de la descripción breve.
 *
 * La descripción detallada comienza aquí, nótese
 * que se debe dejar una línea en blanco para lograr
 * tener las dos descripciones (breve y detallada)
 */
```

Si no se utiliza el comando @brief, Doxygen tomará la descripción hecha en el bloque como una descripción detallada y no habrá descripción breve.

- Descripción de argumentos y métodos.

A la hora de hacer una descripción de los argumentos de métodos y funciones ésta se hará en línea, es decir, luego de la declaración de cada uno de los argumentos se da una breve descripción de cada uno de ellos, en el siguiente ejemplo se muestra el formato a utilizar.

```
int multFunction ( int c , /**<Primer operando de la operación */
                   int d , /**<Segundo operando de la operación */
                   int e /**<Tercer operando de la operación */ );
```

- Documentación de tipos de datos.

Para describir la función y los elementos que componen los tipos de datos definidos se pueden utilizar los formatos especificados en los apartados anteriores. A continuación se muestra un ejemplo:

```
/*!
 * @brief Enumerado de los tipos de datos admitidos
 *
 * Descripción más detallada de la función de este tipo de dato.
 */
enum DataTypes{ INTEGER, /**<Puede ser un valor de tipo entero */
                DOUBLE, /**<Puede ser un valor de tipo double */
                STRING /**<Puede ser un valor de tipo string */ };
```

Observamos que se genera una descripción breve seguida de una descripción más detallada para la función, luego se explica brevemente el valor de retorno de la misma y la descripción de los parámetros usando el formato de documentación en línea.

- Comandos @autor y @date.

Es importante que se especifique el nombre del autor y la fecha de creación de cualquier estructura en un código, para ello se utilizan los comandos @author y @date para el nombre del autor y la fecha respectivamente, en el siguiente ejemplo se puede ver la acción de estos comandos:

```
/**@brief Descripción breve
 *
 * Aquí comienza la descripción detallada
 * @author Rayner Pupo rpgomez@estudiantes.uci.cu
 * @date 07-02-2011
 */
```

- Comando @see.

Existe otro comando útil que permite hacer referencias a otras clases o métodos cuando esto sea necesario, es importante usar este comando en la documentación a la hora de implantar los métodos o funciones propias de alguna clase, este comando es @see y su uso se muestra en el siguiente ejemplo.

```
/**
* Constructor de la clase
* @see Test::Test()
*/
Test1();
```

Aquí se generará en la documentación para el constructor de la clase de prueba Test1 una referencia hacia la documentación existente para el constructor de la clase de prueba Test.

Si se quiere hacer una referencia desde cualquier estructura hacia la documentación completa de una clase ya documentada, solo se debe utilizar el comando @see seguido del nombre de la clase de esta forma:

```
/**
* Constructor de la clase
* @see Clase
*/
Test1();
```

(13)

2.4 Pruebas de la solución.

2.4.1 Descripción de los casos de prueba.

La realización de los casos de pruebas es la actividad en la cual un sistema o componente es ejecutado bajo ciertas condiciones o requerimientos específicos, cuyos resultados son observados, registrados y evaluados. Después de la implementación de la solución se realizaron las pruebas de caja negra, éstas permiten al ingeniero de software, dado un conjunto de condiciones de entrada, probar exhaustivamente los requisitos del sistema.

Al realizar la primera iteración de las pruebas de caja negra se arroja como resultado un total de 12 no conformidades de las cuales 5 se clasifican como significativas, relacionadas problemas de funcionalidad de la aplicación. Las restantes 7 no conformidades son de tipo no significativas y son detectadas principalmente por problemas ortográficos o de redacción.

Luego de un plazo de corrección de 10 días, con una segunda iteración de pruebas se reducen las no conformidades a un total de 4, siendo una de estas de tipo no significativa relacionada con problemas de ortografía y 3 significativas pertenecientes a funcionalidades del sistema.

Durante una tercera iteración de pruebas, tras un plazo de 7 días no son detectadas no conformidades.

Se probaron 19 requisitos funcionales, los que representan un total del cien por ciento del total. Para cada uno se tuvo en cuenta los distintos escenarios que representan las posibles formas en que el usuario puede interactuar con el sistema.

Los resultados de las pruebas de caja negra realizados a la aplicación pueden ser consultados en el Anexo 2.

2.5 Conclusiones parciales.

En este capítulo se muestran en contexto los principales patrones de diseño que fueron utilizados para estructurar la implementación de una forma más entendible y reutilizable. Los usos de un mismo patrón se repiten en más de una ocasión durante el desarrollo de la aplicación pero la forma en que se aplican varía considerablemente.

Los estilos de codificación y documentación explicados tienen como objetivo lograr una estructura de nombres y formatos uniformes, lo que permite una mejor comprensión del código fuente, esto reduce el tiempo empleado en dar mantenimiento e incorporar nuevas funcionalidades al sistema.

Por último se describen las pruebas realizadas al sistema, con el principal objetivo de encontrar errores o posibles vulnerabilidades en el software. De un total de veintiséis pruebas realizadas a la aplicación veintiséis resultaron satisfactorias. Se destaca que los resultados finales obtenidos en las pruebas son resultado de un proceso iterativo de solución de errores encontrados previamente.

CONCLUSIONES GENERALES.

Con la implementación de la Estación de Monitorización Suria versión Qt, el centro de desarrollo GEYSED cuenta con una herramienta dedicada a la Video Vigilancia, la cual posee la mayoría de las funcionalidades típicas de esta clase de aplicación[8]. La principal característica que distingue al sistema es su posibilidad de ejecución en múltiples plataformas[9], sin sufrir ningún cambio en cuanto a la interfaz gráfica o la usabilidad, lo que puede ampliar considerablemente el espectro de clientes interesados en el producto.

De manera general se cumplieron los objetivos trazados al inicio de la investigación y se alcanzaron los resultados esperados.

[8] Las estaciones de monitorización generalmente poseen funcionalidades tales como: listar cámaras disponibles, visualizar y manipular cámaras (PTZ y controles de imagen).

[9] Actualmente pueden ser construidos binarios para Windows, Linux y Mac OS X.

RECOMENDACIONES

A continuación se realizan algunas recomendaciones para la continuación de la implementación del sistema.

- Mejorar el manejo de visores dentro del área de vistas.
- Agregar nuevos renderizadores[10] al sistema.
- Incorporar una ayuda a la aplicación.
- Agregar soporte para una mayor cantidad de cámaras.
- Incorporar funcionalidades como la detección de movimiento.

[10] Actualmente solo se ha desarrollado el render para libvlc.

BIBLIOGRAFÍA CONSULTADA

1. **Edmis Deivis Semanat Aldana, Leonor Verdecia Four.** *Sistema de Video Vigilancia.* La Habana : s.n., 2009.

2. All-Seeing Eye La Historia de Video Vigilancia.htm. [En línea] 2010. [Citado el: 20 de 10 de 2010.] htto://www.All-Seeing Eye La Historia de Video Vigilancia.htm.

3. **NexoTech.** Nexo-Tech. [En línea] [Citado el: 14 de diciembre de 2010.] http://www.nexo-tech.com/srv_ip.php?menu=2&submenu=2.

4. D_ Link Software de gestión de Cámaras IP. [En línea] [Citado el: 8 de 11 de 2010.] http://www.dlink.es/.

5. **Datys.** *Datys. [Online] DATYS le brinda un Sistema de Video Vigilancia profesional.* 2010.

6. CodeGoogle. [En línea] [Citado el: 15 de diciembre de 2010.] http://code.google.com/p/idvr/.

7. Grupo de soluciones GSInnova. [En línea] [Citado el: 10 de diciembre de 2010.] http://www.rational.com.ar/herramientas/rup.html.

8. Scribd. [En línea] [Citado el: 15 de diciembre de 2010.] http://www.scribd.com/doc/47886536/generalidades.

9. *qt.nokia.com.* [En línea] [Citado el: 15 de diciembre de 2010.] http://www.qt.nokia.com.

10. kdevelop. [En línea] [Citado el: 16 de diciembre de 2010.] www.kdevelop.org.

11. Doxygen. [En línea] [Citado el: 16 de enero de 2011.] http://www.stack.nl/~dimitri/doxygen/.

12. Kioskea.Net. [En línea] 16 de Octubre de 2008. [Citado el: 14 de Enero de 2011.] http://es.kioskea.net/contents/crypto/ssl.php3.

13. **Erich Gamma, Richard Helm , Ralph Johnson , and John Vlissides.** *Design Patterns.* 0-201-63361-2.

14. **Sorio, Rosalina Puerto.** *Sistema de Vigilancia por Cámaras Cancerbero.* 2010.

15. **DATYS.** Sistemas, D. T. (s.f.). [En línea] [Citado el: 17 de 10 de 2010.] http://www.datys.cu/.

16. XYMA SAVE VISION SISTEMA DE VIDEO VIGILANCIA IP. [En línea] [Citado el: 8 de 11 de 2010.] http://www.datys.cu/docs/Documentación%20Productos/Xyma.

17. Scribd. [En línea] 2010. [Citado el: 9 de febrero de 2011.] http://www.scribd.com/doc/3062020/Capitulo-I-HERRAMIENTAS-CASE.

BIBLIOGRAFÍA CONSULTADA

1. C++ GUI Programing whit Qt4 1st Edition.
2. Design Patterns in C++ with Qt4.
3. Foundations of Qt Development.
4. The C++ Programming Language Special 3rd Edition.
5. The Art of Building Qt Applications.
6. Advanced Qt Programming Creating Great Software with C and Qt 4.

GLOSARIO DE TÉRMINOS.

A

- API: Interfaz de programación de aplicaciones, es el conjunto de funciones y procedimientos que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción.
- Aplicación: Terminología técnica para referirse a un programa de software.
- Árbol de Cámaras: Componente visual usado para representar la estructura organizativa de las cámaras.
- Área de vistas: Componente visual que permite la distribución ordenada de uno o más visores en un área.
- Autenticación: Verificación de la identidad de una persona o de un proceso para acceder a un recurso o poder realizar determinada actividad.

C

- Cámara IP: Es una cámara que emite las imágenes directamente a la red (Intranet o internet) sin necesidad de un ordenador.
- Circuito cerrado de televisión: Es una tecnología de vídeo vigilancia diseñada para supervisar una diversidad de ambientes y actividades de un lugar.

E

- Estación de monitorización: Componente que permite la supervisión de varios entornos, en el caso de la Video Vigilancia es aplicado al control y visión de cámaras.

F

- Flujos de video: Señal enviada desde cámaras tanto por un medio analógico como digital.

I

- IDE: Entorno Integrado de Desarrollo.

L

- Librería: Es un conjunto de subprogramas utilizados para desarrollar software.

M

- Meta Objeto: Objeto que encapsula información acerca de otro objeto.
- Mpeg4: Es el nombre de un grupo de estándares de codificación de audio y video así como su tecnología relacionada normalizada por el grupo MPEG (Moving Picture Experts Group)
- Multicast: En español Multidifusión, es el envío de la información en una red a múltiples destinos simultáneamente.

- Multiplataforma: Es un término usado para referirse a los programas, sistemas operativos, lenguajes de programación, u otra clase de software, que puedan funcionar en diversas plataformas.
- Multiplexores: Se utiliza como dispositivo que puede recibir varias entradas y transmitirlas por un medio de transmisión compartido. Para ello lo que hace es dividir el medio de transmisión en múltiples canales, para que varios nodos puedan comunicarse al mismo tiempo.

P

- Plugin: Es un módulo de hardware o software que añade una característica o un servicio específico a un sistema más grande.
- PTZ: Es un acrónimo de pan-tilt-zoom, las cámaras PTZ pueden moverse horizontalmente, verticalmente y acercarse o alejarse de un área o un objeto de forma manual o automática.

R

- RJ45: (registered jack 45) es una interfaz física comúnmente usada para conectar redes de cableado estructurado, (categorías 4, 5, 5e, 6 y 6a).

S

- Sistemas operativos embebidos: Un sistema embebido o empotrado es un sistema de computación diseñado para realizar una o algunas pocas funciones dedicadas. Frecuentemente en un sistema de computación en tiempo real.
- Socket: Designa un concepto abstracto por el cual dos programas (posiblemente situados en computadoras distintas) pueden intercambiar cualquier flujo de datos, generalmente de manera fiable y ordenada.
- Software Libre: Es la denominación del software que respeta la libertad de los usuarios sobre su producto adquirido y, por tanto, una vez obtenido puede ser usado, copiado, estudiado, modificado y redistribuido libremente.

U

- Unicast: Es el envío de información desde un único emisor a un único receptor.

V

- Video Vigilancia: Mecanismo de supervisión mediante el uso de cámaras.
- Video Digital: Es un tipo de sistema de grabación de video que funciona usando una representación digital de la señal de vídeo, en vez de analógica.
- Visor: Componente gráfico que representa la señal emitida por una cámara mediante el uso de un render determinado.

X

- X11: X es el encargado de mostrar la información gráfica de forma totalmente independiente del sistema operativo en sistemas Unix. 11 se refiere a la versión actualmente en uso.