2 Análisis y Diseño del Sistema.

De islas cuadradas flotantes y de cámaras movidas en Internet;
del empirismo al razonamiento, y del sueño a la práctica.

2.1 Análisis y Diseño de la Aplicación.

El análisis, diseño, y posteriormente, la construcción, comenzó a partir de recolectar los requerimientos del problema (manipular la videocámara en un ambiente Internet), y comenzar a buscar un lenguaje de programación, así como componentes físicos (para el circuito de control) con los que se pudiera construir una aplicación que pudiera obtener una imagen a voluntad de dicho dispositivo, cambiar la posición física de la videocámara, y obtener una nueva imagen en la nueva ubicación de la videocámara; también se requería guardar información, para consultarla posteriormente. También se buscaron herramientas ***CASE*** y de desarrollo de aplicaciones, a la vez que se buscó algún estándar de análisis, diseño y desarrollo de aplicaciones al cual apegarse.

Dentro de las metodologías para el desarrollo de sistemas de cómputo se encuentran el de desarrollo en cascada, y *RUP* (*Rational Unified Process*). El primero es para crear sistemas a partir de sistemas ya existentes o para automatizar procesos manuales; en ambos casos ya existen reglas claras de los procesos existentes, contando con documentación escrita acerca de los mismos. La otra metodología, *RUP*, fue desarrollada por *Rational Software*, para crear sistemas donde no existan tales reglas, y se tenga que actualizar el proceso de desarrollo y adaptarse a cambios; esta metodología no está desarrollada para un lenguaje de cómputo específico, pero recomienda el uso herramientas para automatizar el desarrollo de sistemas (sobre todo los productos de *Rational Software*). Para las fases de creación de este sistema de cómputo no se adquirieron aplicaciones para seguir esta metodología, sino que se siguen lo que llaman las mejores prácticas, que son reglas que surgieron al estudiar cómo se construyeron exitosamente sistemas de cómputo. Dichas reglas son:

1 Desarrollar los sistemas de cómputo (*software*) iterativamente.
2 Administrar requerimientos.
3 Utilizar arquitecturas que tengan base en componentes.
4 Desarrollar la aplicación con base en modelo visual.
5 Verificar la calidad del sistema.
6 Tener un control de cambios de los códigos fuente.

Para las fases de análisis, diseño, desarrollo y mantenimiento del sistema, se procuró seguir las reglas de *RUP*.

2.1.1 Análisis de la aplicación.

El análisis comenzó por buscar un lenguaje o lenguajes de programación con el que se pudiera programar y construir la aplicación, teniendo en cuenta que dicha aplicación o sistema tendría que unir elementos muy heterogéneos entre sí, tales como los puertos de comunicación de una computadora personal y la construcción de páginas de web; también se buscó que dicho lenguaje fuera orientado a objetos, para aprender desarrollar una aplicación con ese paradigma de programación (que ha reemplazado en gran medida a la programación estructurada), y aprovechar la experiencia adquirida en otros proyectos.

Otro factor que influyó en la selección del lenguaje fue el económico, descartándose aquellos lenguajes de programación que cobraran licencia por su uso, o por el servidor de aplicaciones web. Se analizaron algunos

lenguajes de programación, los cuales se comparan en la siguiente tabla:

Lenguaje	Ventajas:	Desventajas:
'*C*'.	Existen varios compiladores gratuitos; puede tener interacción de forma directa con la memoria reservada para periféricos, y permite hacer llamadas a rutinas en lenguaje ensamblador. Se le considera un lenguaje rápido y eficiente. Existen compiladores para diversos sistemas operativos, tanto gratuitos como de licencia.	Se pueden hacer páginas de web que llamen a código binario mediante *CGI*, pero por cada llamada a estas páginas se genera un proceso, relativamente "pesado", porque por cada llamada a librerías, incorpora el código binario de las mismas al ejecutable que se genere. No es un lenguaje orientado a objetos, por lo que no se podría re-usar código fuente. Los apuntadores mal empleados pueden causar graves problemas. Se evaluó que las librerías para video y para bases de datos se tendían que comprar y/o conseguir de terceros. El código fuente no es totalmente portable entre sistemas operativos; tiene más diferencias entre funciones y librerías mientras el código fuente se relacione con componentes físicos y/o llamadas al sistema operativo.
C++	Es la versión orientada a objetos de '*C*', lo que lo hace más poderoso. Puede manejar la memoria reservada a periféricos, lo que le permite tener interacción con los componentes físicos de la computadora, por lo que diversas compañías que construyen periféricos hacen aplicaciones en ese lenguaje, y algunas crean librerías de aplicaciones (generalmente para *Windows*). Existen algunos compiladores gratuitos, pero los ambientes gráficos de programación son de licencia.	Permite la herencia múltiple, lo cual puede hacer el código confuso. El uso descuidado o malintencionado de los apuntadores pueden hacer un sistema inestable, o afectarlo negativamente. También, por sólo incluir el nombre de librerías (aunque no se usen) hace que se incorpore su código binario al ejecutable que se genere. Se puede construir páginas web dinámicas mediante *CGI*, para que se invoque un ejecutable; por cada vez que se llame, se genera un proceso, el cual reserva cierta cantidad de recursos de la computadora para su uso exclusivo, pero múltiples llamadas pueden agotar los recursos de cómputo del servidor.
PHP	Es un lenguaje de ejecución de comandos de guiones, diseñado para crear páginas web dinámicas, el cual puede tener interacción con muchas bases de datos, y tiene soporte en varios sistemas operativos, por lo que el código desarrollado no necesita ser adaptado al cambiarse a otro sistema operativo. Es de código abierto.	No es un lenguaje que tenga todas las características de Orientación a Objetos. Permite el manejo de *CGI* para ejecutar código binario, pero por sí solo no tiene la capacidad para ejecutar código para el control de los periféricos.

ASP.	Es un lenguaje de ejecución de guiones (del lado del servidor) para crear páginas web dinámicas, creado por Microsoft para *Windows*. Usa los lenguajes de programación *Visual Basic*, *J#* y *C#*. *C#* es la versión de *C++* de Microsoft; *J#* es la versión Microsoft de Java - la cual no cumple con las especificaciones de *Sun Microsystems*, los creadores del lenguaje java. *Visual Basic* es un lenguaje para crear aplicaciones de escritorio en *Windows*. Antes de la plataforma *.NET*, cobraban licencia por su uso. Usa *ODBC* para usar bases de datos, pero varias librerías *ODBC* cobran licencia.	Está limitado a la plataforma *Windows*, pero se había contemplado que en un futuro este sistema, o algunas partes del mismo fueran migradas a *Linux* o *Unix*; esta tecnología se usa en el *IIS* (*Internet Information Server*), el cual cobraba licencia, por lo que se descartó para programar el sistema (actualmente es gratuito en los servidores *Windows* y en las versiones *Windows 2000* y *XP Professional*). Las aplicaciones en estos lenguajes frecuentemente tienen problemas de estabilidad, y no tienen el mismo rendimiento las aplicaciones programadas en lenguajes similares.
Dephi	Es un entorno gráfico de programación cuyo lenguaje era Pascal orientado a objetos, y actualmente tiene su propio lenguaje *Delphi Win32*; aunque fue ideado para crear aplicaciones de escritorio, Pascal también puede tener acceso a la memoria de los periféricos, y hacer rutinas de bajo nivel para tener interacción con ellos.	Permite crear páginas web dinámicas, usado *CGI*, que llama a librerías compartidas (*dlls*) para ejecutar el código, pero el tiempo de respuesta es muy grande. Cobra licencia por su uso, y se limita a la plataforma *Windows*.
Ruby	Es un lenguaje orientado a objetos gratuito y de código abierto. Permite incorporar extensiones programadas en '*C*' fácilmente. Las librerías que tiene son creadas y mantenidas por grupos de programadores.	Fue liberado en 1995, pero inicialmente no tuvo respaldo de grandes compañías de Tecnología de Información, y la primera documentación de este lenguaje estaba en japonés. Es a partir de 2006 cuando tiene mayor difusión.
Java	Es un lenguaje propiedad de *Sun Microsystems*, de uso gratuito. Es orientado a objetos y multi-plataforma, que ha sido respaldado por muchas empresas. Permite crear fácilmente páginas web dinámicas por medio de sus tecnologías de ***servlets*** y ***JSPs***. Puede usar muchas bases de datos en diversos sistemas operativos. Posee la tecnología ***Java Native Interface*** (***JNI***), con la cual puede llamar a librerías dinámicas. Tiene las ***APIs Java Communications***, ***Media Framework*** y ***JDBC***, para uso de puertos locales, multimedia y base de datos, respectivamente.	Es necesario que esté instalada la *Java Virtual Machine* en la computadora donde se ejecute alguna aplicación de este lenguaje, para que pueda interpretar los *bytecodes*, y ejecutar las instrucciones en el lenguaje máquina del sistema operativo; la interpretación de los *bytecodes* hace más lenta la ejecución del programa, en comparación con otros lenguajes que generan binarios (los cuales son sólo para un sistema operativo).

Tabla 11: Comparación de ventajas y desventajas de lenguajes de cómputo para programar una aplicación web.

Después de comparar ventajas y desventajas de diversos lenguajes de computación -excluyendo a *Ruby*, del cual no se tenía noticia-, se escogió el lenguaje java, ya que parecía que con un sólo lenguaje se podía construir el

sistema y tuviera interacción de forma armoniosa; incluso, *JNI* puede llamar a librerías dinámicas recuperando datos complejos (como resultado de las llamadas a las funciones nativas), si se tuviera que programar alguna parte del sistema en otro lenguaje de cómputo. Otros lenguajes permiten crear páginas web fácilmente, pero no pueden tener interacción con componentes físicos por sí solos -tales como la videocámara de estado sólido y los puertos de comunicación de la computadora; se tendría que usar otro lenguaje de cómputo que genere binarios con el lenguaje máquina, para poder manipular los dispositivos físicos, además de que la comunicación entre los lenguajes sería más limitada.

También influyó en la selección de este lenguaje la capacidad de la interfaz ***Java Media Framework*** (***JMF***); no se iba a usar software de desarrollo *Logitech*, porque limitaría al sistema a que funcionara con videocámaras de esta marca, mientras que ***JMF*** permite llamar a dispositivos multimedia, de distintas marcas, de una forma genérica.

Otro requerimiento del sistema es que se guarde información, para consultarla posteriormente; se decidió que la mejor manera de cumplir con ese objetivo es a través de un administrador de base de datos, ya que permite almacenar información (incluyendo datos binarios), consultarla y/o manipularla. Aunque el tipo principal de información a guardar son datos binarios de las imágenes obtenidas, también se planeó que se pudiera guardar información de otros tipos, incluyendo un registro de usuarios, para hacer un módulo de administración del sistema.

El paso siguiente fue la elección del sistema operativo; en este caso se carecía de acceso a un "*mainframe*", a la vez que tenía más acceso a computadoras con sistema operativo *Windows* instalado (por ser más populares que *Unix* y otros sistemas operativos). Aunque ya había aparecido el sistema operativo *Linux*, se carecía del conocimiento técnico para montar y administrar *Linux* en una computadora personal, por lo que se decidió crear el sistema en una plataforma *Windows*, sin perder de vista que dicho sistema de cómputo pudiera ser migrado (total o parcialmente) a otro sistema operativo, y o aprovechar las experiencias adquiridas en desarrollo del mismo para efectuar dicha migración. El equipo que se tenía para programar el sistema tenía *Windows Millenium* instalado, aunque algunos componentes java ya construidos se habían hecho en *Windows 98*.

Se procedió a escoger el administrador de base de datos. Ya se había escogido el lenguaje con el cual programar, java (que es multi-plataforma), y se tenía la restricción de que se construiría sobre el sistema operativo *Windows*, surgiendo los siguientes requerimientos para el administrador de base de datos:

- Que se pudiera instalar en *Windows*, pero que de preferencia tuviera versiones para otros sistemas operativos.
- Que tuviera librerías ***JDBC*** gratuitas (para poderla usarla desde java), y de preferencia que fueran del tipo 4, que son 100% java, y no requieren de instalación de otro software.
- Debería poder almacenar datos numéricos, cadenas, fechas e información binaria (de preferencia, campos *blob*).
- El administrador debería ser gratuito (por restricciones económicas).
- La base de datos debería correr en una computadora persona, dejando recursos disponibles para ejecutar algunas otras aplicaciones al mismo tiempo (tales como el sistema a desarrollar, procesador de palabras, etcétera).

Después de revisar los requerimientos que debía cumplir la base de datos, se efectuó una comparación entre los tipos de base de datos. El tipo más popular de base de datos son las relacionales -las cuales fueron el último punto de ***Conceptos Básicos***- en las que los datos se guardan de forma separada en columnas (del mismo tipo de dato). El otro tipo de base de datos, más reciente, es la base de datos orientada a objetos, en la cual se guardan colecciones de objetos (los cuales no se dividen); esto ahorra mucho esfuerzo de programación, al evitar que se separe la información y guardar dato por dato en columnas, lo que permite que se integre fácilmente con aplicaciones orientadas a objetos.

Se había encontrado que varios fabricantes de administradores de bases de datos orientadas a objetos cobran licencia por su uso, o permiten su uso (para evaluación) por un periodo de tiempo. Aunque ya existían algunos administradores gratuitos -tal como *db4o*, la versión gratuita de *Versant*-, también influyo el factor humano, ya que no todos conocen las bases de este tipo, por lo que no habría alguna persona que pudiera mantener dicha base si se legara la aplicación usando bases orientadas a objetos; por otra parte, ya existían proyectos utilizando bases de datos relacionales, con las cuales se tenía más familiaridad. Por esto último se decidió usar una base de datos relacional, procediendo tanto a escoger el administrador de base de datos relacional como a diseñar el modelo Entidad-Relación -el diseño se hizo de antemano, ya que se debía verificar que dicho administrador pudiera manejar los tipos de datos requeridos en el diseño.

A continuación se muestra una tabla donde se comparan algunos administradores de base de datos -no se compararon todos los administradores de bases de datos más populares, por la gran cantidad de los mismos-:

Administrador de base de datos relacional.	**Ventajas:**	**Desventajas:**
Oracle.	Posee diversas herramientas y funciones que lo han convertido en referencia respecto a otras bases de datos. Tiene versiones para diversos sistemas operativos.	Requiere licencia y muchos recursos de cómputo; la versión gratuita, *Oracle Lite*, tiene menos herramientas pero aún así necesita muchos recursos de cómputo.
Access	Es relativamente popular. No se apega al *ANSI SQL*.	Es de licencia; no es eficiente para manejar concurrencia.
SQL Server	Maneja concurrencia adecuadamente.	Es de licencia; no proporciona librerías ***JDBC***, por lo que la comunicación con el lenguaje java es por medio del puente ***ODBC-JDBC***.
Paradox	Es una base de datos para aplicaciones de escritorio. Utiliza *QBE* (*Query by Example*) para facilitar las consultas a los usuarios, y posee conectores *SQL Links* nativos para comunicarse a otras bases de datos.	Es de licencia, y sólo para plataforma Windows; no provee librería ***JDBC***, por lo que una aplicación java tendría que usar el puente ***ODBC-JDBC***, a la vez que se crea un *SQL Link* a *ODBC*, o usar un ***JDBC*** de otro proveedor (pagando licencia).
DB2	Es una administrador confiable	Una computadora personal no le puede dar los recursos necesarios para que corra adecuadamente; brinda una librería ***JDBC***, pero varias de sus herramientas requieren el *Java Development Kit* de *IBM*. Es de licencia.
SQL Anywhere	Es confiable y tiene varias herramientas de administración y manipulación de datos.	Es de licencia; la versión de evaluación es por tiempo limitado.

Java DB	Es una base de datos relacional construida en el lenguaje java; usa *ANSI SQL*. Tiene versiones para varios sistemas operativos, y al ser una aplicación relativamente chica, consume pocos recursos de cómputo.	Es de reciente aparición; no existía al momento de iniciar el análisis, pero se menciona por las características que ofrece.
MySQL Community Edition	Es un administrador de bases de datos relacionales gratuito bajo la licencia *GNU* -la que permite modificar el código fuente, si los cambios hechos son publicados de forma gratuita. Existen muchas versiones para diversos sistemas operativos. Tiene la funcionalidad de cualquier administrador de base de datos comercial, pero no requiere tantos recursos de cómputo como las versiones *lite* o personales de productos similares de otros vendedores. Posee utilerías y librería ***JDBC*** gratuitas.	

Tabla 12: Comparación entre administradores de bases de datos relacionales.

Después de comparar diversos administradores de base de datos, se aprecian las ventajas de *MySQL Community Edition* (excluyendo a *Java DB*, que es reciente), sobre todo porque se puede usar gratuitamente, no utiliza muchos recursos a la vez que brinda la funcionalidad de administradores de bases de datos relacionales. Por otra parte, en el **Centro de Ciencias Aplicadas y Desarrollo Tecnológico** (**CCADET**, antiguo **Centro de Instrumentos**) ya se habían hecho o iniciado proyectos con dicho administrador, por lo que ya existe personal que conozca y administre.

La última aplicación esencial para poder crear el sistema es el servidor de aplicaciones web, o en su defecto, un contenedor de ***servlets*** y ***JSPs***. Los requisitos que aparecieron fueron los siguientes:

- Que fuera gratuito para su uso.
- Utilizara el lenguaje Java, y pudiera ejecutar el código de ***servlets*** y ***JSPs***.
- Que pudiera utilizar la librería ***JDBC*** de ***MySQL*** (de tipo 4, la cual no requiere instalaciones adicionales).
- Que pudiera llamar a interfases que invoquen a código nativo y/o usar la interfaz *Java Native Interface* (*JNI*), tanto para el control de los puertos locales como el video de la videocámara de estado sólido.

De forma simular, se compararon diversos servidores de aplicaciones, comparándolos en la siguiente tabla:

Servidor de aplicaciones web.	Ventajas.	Desventajas.
Apache-Tomcat.	Es una aplicación de código abierto que implanta los estándares de ***servlets*** y ***JSPs***, desarrollada y distribuida bajo la licencia de *Apache Software*. Algunos servidores de aplicaciones web lo siguen como referencia, o iniciaron con base en alguna de las versiones de *Tomcat*.	No es un servidor de aplicaciones web, sino un contenedor de ***servlets*** y ***JSPs***. No tenía respaldo de entornos de desarrollo integrados gratuitos hasta apenas unos cuantos años.
Resin.	Tiene varias versiones para diversos sistemas operativos, proveyendo de respaldo para ***servlets***, ***JSPs***, *EJBs*, *PHP* y otros estándares recientes para aplicaciones web. Ofrece versiones de código abierto y otras comerciales (de mayor rendimiento, pero son de licencia).	Cuando inició el proyecto no se tuvo acceso a la versión de código abierto, y las demás cobraban licencia.
Sun Glassfish Application Server.	Ofrece el respaldo para todas las tecnologías web de java que ofrece *Sun Microsystems*.	Actualmente es gratuito, pero las primeras versiones (que tenían otros nombres) eran de licencia. Requiere de más recursos de cómputo que otros servidores, debido a la cantidad de servicios que levanta para atender las diversas tecnologías que implanta.
JBoss	Es un servidor de aplicaciones empresariales web, que tiene base en ***Apache-Tomcat*** y en el *Apache Web Server*, por lo que también tiene contenedor de EJBs, e integra otras tecnologías.	Es de licencia; su consola de administración es confusa.
WebLogic	Es un servidor de aplicaciones web poderoso, el cual puede contener ***servlets***, ***JSPs***, *EJBs*, *webservices*, contando también con su propio entorno de desarrollo integrado.	La versión personal sólo permite un dominio en un equipo.
WebSphere	Es un servidor de aplicaciones web robusto, de *IBM*. La versión gratuita, *WebSphere Community Edition*, es una versión personal construida con base en productos de *Apache Software*.	Son de licencia. Las primeras versiones del mismo estaban más enfocadas para equipos y productos *IBM*; las versiones más recientes ya colaboran con otros productos java de terceros. Aunque es tecnología 100% java, este servidor utiliza configuraciones y librerías exclusivas de *IBM*; una de las últimas versiones tenía errores que impedían generar *webservices*.
Oracle Application Server (OAS).	Fue reemplazado por *WebLogic*, cuando *Oracle* adquirió *Bea Systems*.	Estaba más inclinado a integrarse con otras herramientas de *Oracle*.

Tabla 13: Comparación entre servidores de aplicaciones *web*.

Después de comparar sus características, se descartaron los servidores que requerían de pago de licencia;

después se descartaron los que requerían otras aplicaciones para funcionar y los de licencias personales (los que requieren un registro por cada instalación).

Se prefirió el ***Apache-Tomcat*** porque es gratuito, porque se puede redistribuir respetando su licencia (de *Apache Software*), porque sirve de referencia para construir otros servidores de aplicaciones (a los cuales se podría migrar la aplicación posteriormente), y porque no requiere de muchos recursos de cómputo. Aunque no es un servidor de aplicaciones web (porque no ofrece el respaldo de todas las tecnologías web) ni posee la capacidad administrativa de los servidores de aplicaciones, sí tiene la capacidad para ejecutar el código de los ***servlets*** y ***JSPs***, integrar otras librerías (como el ***JDBC*** de *MySQL* y las de ***Java Media Framework***), y de llamar a librerías dinámicas (nativas del sistema operativo) por medio de *JNI*.

Algunos servidores de aplicaciones ofrecían mejor rendimiento, pero se prefirió desarrollar la aplicación en un estándar como el *Apache-Tomcat*, y si fuera necesario cambiar a un servidor de aplicaciones, sería más fácil mover migrar la aplicación web, y después efectuar la configuración del nuevo servidor para la aplicación.

Las últimas herramientas escogidas (no esenciales) fueron un entorno de desarrollo integrado (*IDE*), alguna herramienta para diagramar circuitos electrónicos, y alguna de *CASE*.

Los entornos de desarrollo integrados no son considerados como absolutamente indispensables (pero más adelante fueron casi vitales), pero sí fueron de mucha ayuda para el proyecto, para poder vincular el editor con el compilador, ahorrando tiempo para detectar errores antes y al compilar la aplicación. Los códigos más simples (y varios de prueba) fueron hechos en el editor de notas y compilados en una ventana de consola. Más adelante se decidió aprovechas las bondades de los entornos gráficos, utilizando la versión personal del *JBuilder*, cambiando posteriormente al *JDeveloper* (el cual revisa la sintaxis de los ***JSPs***), y más recientemente, con ***NetBeans***, que permite la depuración de las aplicaciones web.

2.1.2 Diseño de la aplicación.

El diseño de la aplicación se dividió en dos partes, las cuales son independientes en esta fase de desarrollo del sistema:

- El diseño de la base de datos relacional.
- El diseño de la programación de la aplicación (*software*).

2.1.2.1 El diseño de la base de datos relacional.

El diseño de la base de datos relacional se hizo con la metodología ***CASE*Method*** (explicada en el punto ***1.6 de Conceptos Básicos***), prefiriéndose ésta sobre otras (como la de Chen), debido a que es más fácil modelar relaciones complejas, como la de muchos a muchos, y descomponerla con ayuda de una nueva entidad dependiente. A pesar de no contar con herramientas *CASE*, se siguió la metodología diagramando con una aplicación para graficar y hacer presentaciones.

El diseño debía permitir guardar datos del sistema, y de usuarios del mismo; debía poder guardar información binaria, y datos flotantes -para poder guardar otro tipo de mediciones, en la misma base de datos; también debía ser independiente de los lenguajes de programación, y de los administradores de bases de datos relacionales (el cual todavía se seguía escogiendo todavía). El diseño siguió los siguientes pasos:

1 Identificar las entidades y sus atributos.

2 Establecer las relaciones entre las entidades.
3 Pasar el diseño a la primera forma normal.
4 Llevar el diseño a segunda forma normal.
5 Concluir el diseño pasando a la tercera forma normal.

El modelo Entidad-Relación obtenido se encuentra en el anexo B. También se construyó la tabla de relaciones, que muestra las asociaciones de las entidades del sistema, la cual se muestra a continuación:

		USUARIO	PUESTO	INSTITUCIÓN	PAÍS	ESTADO	NIVEL_USUARIO	NIVEL	SESIÓN	MEDICIÓN	INSTRUMENTO	CONDICIÓN_INSTRUMENTO	AUTORIZACIÓN	CONDICIÓN_USUARIO	TELÉFONO	TIPO_TELÉFONO
USUARIO		asignado	afiliado	habitando	residiendo	jerarquizado		creando				otorgando		posey endo		
PUESTO	ejercido															
INSTITUCIÓN	afiliando															
PAÍS	habitado															
ESTADO	residido															
NIVEL_USUARIO																
NIVEL	jerarquizando															
SESIÓN	creada															
MEDICIÓN	obtenida								obteniendo							
INSTRUMENTO									calculando		habilitado					
CONDICIÓN_INSTRUMENTO										habilitando						
AUTORIZACIÓN	otorgada												calificando			
CONDICIÓN_USUARIO												calificando				
TELÉFONO	poseído															clasificado
TIPO_TELÉFONO														clasificando		

Tabla 14: Matriz de relaciones del modelo Entidad-Relación para la base de datos del Administrador de la Videocámara.

Para el caso de la entidad Usuario, los atributos Colonia, Municipio, Código Postal, Apartado y Población,

tendrían a su vez atributos, por lo que se convertirían en Entidades (para pasar a la primera forma normal ***NF1***) si el modelo se apegara fielmente a la organización de países en estados (o departamentos), los cuales contienen municipios (o división política análoga), que se dividen a su vez en colonias o barrios, los cuales tienen asignado uno o más códigos postales. Esto implica que también se tendría que modelar las relaciones y las dependencias de las nuevas entidades, lo cual aumentaría la complejidad del modelo.

También se tendría que capturar todos los estados, municipios, poblaciones, etcétera, de todos los países, lo cual es posible, pero consumiría mucho tiempo; esto se podría hacer para un sistema de mensajería, donde sí importa la asociación de la dirección con la división política. Para la forma de registro de Usuarios de este sistema, se considerará que los atributos mencionados anteriormente son atómicos (conteniendo sólo cadenas de caracteres o números), y que sólo dependen de la llave primaria de Usuario.

Para el sistema desarrollado para esta tesis no tiene tanta relevancia el estado o país donde residan los usuarios del sistema; lo más importante es comprobar que se puede manipular un dispositivo periférico a través de Internet, y obtener una respuesta del mismo. La base de datos se construyó para guardar información de la respuesta de dicho periférico, y para registrar algunos datos de usuarios; los cuales tienen diferentes permisos para borrar datos o bloquear el acceso al periférico.

El modelo de la base de datos fue modificado hasta llegar a la tercera forma normal (***3NF***), la cual es recomendada para la mayoría de los modelos de bases de datos relacionales. Existen otras formas normales más estrictas, pero no se revisó si el modelo Entidad-Relación cumple con esas formas normales, ya que se considera que dicho análisis está fuera del alcance de esta tesis.

El paso siguiente fue el diseño de las tablas, en donde cada entidad se convierte en una tabla; el nombre de la tabla es el plural del nombre de la entidad, y cada atributo de la entidad se convierte en una columna de la nueva tabla. La entidad en la cual la relación con otras entidades es obligatoria o tenga cardinalidad múltiple, su tabla correspondiente se construye agregando una columna por cada relación que tenga con otras entidades; este tipo de columnas se conocen como llaves foráneas.

Por cada tabla se forma una matriz, en la que se indica por cada columna, si es obligatoria (No Nula, *NN*) u opcional (si debe tener dato o puede ser nulo), si es única (U), y además se indica si la columna es un tipo de llave -primaria (*PK*), secundaria o foránea (*FK*)-. Después de agregar las columnas, se pude agregar ejemplos de tipos de datos por columna, que corresponden de uno a tres registros que podría tener la tabla; estos datos sirven como referencia para la construcción física de las tablas. El diseño físico de las tablas se muestra a continuación:

Columna:	id_puesto	descripcion
Tipo llave:	PK	
No nulo/Único:	NN, U	NN
Ejemplos:	1	Investigador
	2	Profesor
	3	Alumno

Tabla 15.1: Diseño físico de la tabla *puestos* (*ocupaciones*).

Columna:	id_estado	nombre	nombre_internet	nombre_oficial	nombre_oficial_internet	abreviatura
Tipo llave:	PK					
No nulos Únicos:	NN, U					
Ejemplos:	9	Distrito Federal	Distrito Federal	Distrito Federal	Distrito Federal	D. F.
	16	Michoacán	Michoacá	Michoacán de Ocampo	Michoacá de Ocampo	Mich.
	17	Morelos	Morelos	Morelos	Morelos	Mor.

Tabla 15.2: Diseño físico de la tabla *estados*.

Columna:	id_institucion	nombre
Tipo llave:	PK	
No nulos Únicos:	NN, U	NN
Ejemplos:	1	U. N. A. M.
	2	Centro de Instrumentos
	3	Facultad de Ingeniería

Tabla 15.3: Diseño de la tabla *instituciones*.

Columna:	id_pais	nombre	nombre_internet
Tipo llave:	PK		
No nulos Únicos:	NN, U	NN	NN
Ejemplos:	2	Albania	Albania
	115	México	México
	143	República Checa	República Checa

Tabla 15.4: Diseño de la tabla *paises*.

Columna:	descripcion
Tipo llave:	PK
No nulos **Únicos:**	NN, U
Ejemplos:	Administrador
	Usuario

Tabla 15.5: Diseño de la tabla *niveles*.

Columna:	descripcion_nivel	nombre_usuario
Tipo llave:	FK	FK
No nulos **Únicos:**	NN	NN
Ejemplos:	SU	hrz
	Administrador	Benjas
	Usuario	Juan

Tabla 15.6: Diseño de la tabla *niveles_usuarios*.

Columna:	id_telefono	numero_telefonico	id_usuario	id_tipo_telefono
Tipo llave:	PK		FK	FK
No nulos **Únicos:**	NN, U	NN	NN	NN
Ejemplos:		56 22 86 02 Ext. 122	1	2
		56 76 02 54	2	1
		445551234567	1	3

Tabla 15.7: Diseño de la tabla *telefonos*.

Columna:	id	descripcion
Tipo llave:	PK	
No nulos **Únicos:**	NN, U	
Ejemplos:	1	Casa
	2	Oficina
	3	Celular

Tabla 15.8: Diseño de la tabla *tipos_telefonos*.

Columna:	id_condicion_usuario	descripcion
Tipo llave:	PK	
No nulos **Únicos:**	NN, U	NN
Ejemplos:	1	Activo
	2	Inactivo

Tabla 15.9: Diseño de la tabla *condiciones_usuarios*.

Columna:	fecha	hora	autorizado	autoriza	id_condicion_usuario
Tipo llave:	PK	PK	PK, FK	PK, FK	FK
No nulos **Únicos:**	NN	NN	NN	NN	NN
Ejemplos:	37248	0.64596	7	1	1
	37257	0.74333	14	1	2

Tabla 15.10: Diseño de la tabla *autorizaciones*.

Columna:	numero_sesion	cadena_sesion	fecha_acceso	hora_acceso	fecha_salida	hora_salida	ip_conexion	id_usuario
Tipo llave:	PK1	PK2						FK
No nulos Únicos:	NN, U	NN, U	NN	NN	NN	NN	NN	NN
Ejemplos:	3e+07	ACD15FB	37278	0.5	37279	0.785	18721411	1
	5e+07	BDSDF1A	37255	0.604	36891	0.09	127.0.0.1:8090	2

Tabla 15.11: Diseño de la tabla *sesiones*.

Columna	id_medicion	fecha	hora	cantidad	valor_binario	id_instrumento
Tipo llave:	PK					FK
No nulos Únicos:	NN, U	NN	NN			NN
Ejemplos:	1	37294	0.41763	112		1
	2	37294	0.41807		ABD$#%$#	1

Tabla 15.12: Diseño de la tabla *mediciones*.

Columna:	id_condición_instrumento	descripción
Tipo llave:	PK	
No nulos Únicos:	NN, U	NN
Ejemplos:	1	Habilitado
	2	Inhabilitado
	3	Descompuesto

Tabla 15.13: Diseño de la tabla *condiciones_instrumentos*.

Columna:	id_instrumento	nombre	id_condición_instrumento
Tipo llave:	PK		FK
No nulos Únicos:	NN, U	NN	NN
Ejemplos:	1	Videocámara	1
	2	eesdf	2

Tabla 15.14: Diseño de la tabla *instrumentos*.

Columna:	id_usuario curp nombre apellido_paterno apellido_materno fecha_nacimiento ... cp dirección apartado ... población correo_electrónico página_internet ... contraseña ... puesto estado ... instituto país [column names printed overlapping; not separable]
Tipo llave:	PK1 PK2 FK FK FK FK
Nulos / Únicos:	NN,U NN,U NN NN NN NN NN NN NN NN NN
Ejemplos:	1 RUZH7306... Hugo Ruiz Zepeda ... 26872 ... Portal 91 ... Cerca ... México ... @internet.com.mx http://www.hugo... ABC 3 9 2 1 15 [printed overlapping; partly illegible]
	41 PEHJ70021... Juan Pérez Hernández 25612 Carretera México-Puebla Km 50 Addsed Adjkjh juan jjiui 3 21 1 1 15 [printed overlapping; partly illegible]

Tabla 15.15: Diseño de la tabla *usuarios*.

Las tablas se construyen con base en sentencias *SQL* de tipo *DDL* (*Data Definition Language*), iniciando con la sentencia de creación de la base de datos, siguiendo con cada tabla, indicando el nombre de la tabla, el nombre y tipo de columna, y si puede ir nulo o no (dato no obligatorio u obligatorio, respectivamente). Al último se construyen otros elementos de la base de datos, tales como índices. La mayoría de los administradores de bases de datos relacionales aceptan las sentencias *DDL* del *ANSI 92*, aunque la construcción varía dependiendo de dicho administrador, según las mejorías que quiera implantar dicho administrador.

En este caso, ya se había escogido *MySQL* como administrador de base de datos, por lo que la base de datos y sus tablas se construyeron mediante sentencias *DDL* propias de *MySQL*, mapeando cada matriz, identificando el tipo de dato que corresponde a cada columna, indicando la opcionalidad, y si fuera el caso, señalando si es llave primaria. Todas estas instrucciones *DDL* se escribieron en un archivo *SQL* para ejecutarlo como un guión.

Finalmente, se hizo otro guión con instrucciones ***SQL*** de tipo *DML* (*Data Manipulation Language*), para insertar datos en la base de datos recién creada, creando un usuario por defecto, el super administrador (*SU*), y agregando información en las tablas que se conocen popularmente como catálogos (debido a que rara vez cambian y sirven como referencia).

2.1.2.2 Diseño del sistema.

El diseño del sistema gira en torno del lenguaje de programación java, ya que este último provee todos los elementos de presentación, control, adquisición de imágenes, de base de datos, etcétera, y además permite integrar dichos elementos de forma armoniosa.

Siguiendo la metodología de *RUP*, que aconseja el desarrollo iterativo, y apegarse a arquitecturas para poder administrar las clases de java a generar, se decidió dividir el diseño y desarrollo del sistema por módulos, probar los códigos de forma individual, y una vez resuelto los problemas técnicos más cruciales de un módulo, integrarlo en el ambiente de web, y probar y corregir de nuevo. Se evitó concentrar toda la programación en un archivo fuente, que una misma clase efectuara tareas de distinta naturaleza, prefiriendo que la programación se dividiera en clases, y a su vez, estas últimas se agruparían por el tipo de tareas que llevan a cabo. Siguiendo esta pauta, el diseño del sistema fue subdividido por el tipo de función que lleva a cabo:

- Circuito electrónico de control.
- Módulo de control del puerto paralelo
- Módulo de base de datos.
- Módulo de video (control de la videocámara *CCD* y obtención de imágenes fijas).
- Módulo de web (páginas *HTML*, ***servlets*** y ***JSPs***), el cual contiene elementos de presentación y otros que controlan e integran a los módulos anteriores.

Las primeras clases del lenguaje java programadas para esta aplicación tenían una arquitectura muy simple, siguiendo las convenciones del lenguaje java, dividiendo la funcionalidad de la clase en varios métodos. Se dejó la división de clases por módulo; más adelante, en algunas clases se comenzó a aplicar el concepto de herencia, para evitar volver a programar métodos ya hechos, y para evitar tener muchas versiones de clases similares.

La complejidad del problema a resolver impedía diseñar todo el sistema a detalle antes de programarlo; *RUP* recomienda el desarrollo iterativo, en la que se programa, se prueba, y se hacen los cambios o ajustes necesarios; para los módulos de control del puerto paralelo, de base de datos, y el de video, no se especificó una arquitectura más específica (aún), pero se hizo un ***Diagrama de Flujo de Datos*** sencillo del proceso donde el usuario solicita el cambio de posición de la videocámara, y se le responde con una página *HTML* con la imagen recién obtenida, y se decidió que era mejor que se pasara a la fase de desarrollo.

Sin embargo, antes de comenzar a programar ***servlets*** y otros componentes web, sí se compararon arquitecturas web, tanto para organizar el desarrollo de los componentes web como para facilitar la integración de los otros módulos (diseñados y construidos de forma independiente). Primero se compararon las arquitecturas enfocadas a la página o foliocentristas[49], en las cuales (después de recibir la petición) un ***JSP*** de presentación llama a otro *JSP* que tiene código de procesamiento y de presentación; esta arquitectura permite el desarrollo rápido de páginas web, pero si el código de procesamiento es complejo o cambiante, causaría que se tenga que modificar mucho el JSP que concentre el procesamiento. Debido a lo anterior, se descartó este tipo de arquitectura.

El siguiente tipo de arquitectura revisada fue el modelo de despachador o remisión, la cual se basa en el modelo ***MVC*** (*Model-View-Controller*, Modelo-Vista-Control)[49], cuyas partes se explican a continuación:

- Modelo, el cual es la plantilla de los datos; en esta aplicación son los *JavaBeans*.

- Vista, la cual se expone al usuario mostrando los datos del modelo. Los *JSPs* son los objetos vista en la aplicación web.
- Control, el cual recibe las peticiones del usuario (incluyendo datos de entrada), procesa información, llaman

a otras clases, invocan procesos, y en general, ejecutan diversas tareas; crea instancias de modelo, agregándoles los datos de salida (los que se van a mostrar al usuario) y se los envía a la Vista, a la cual le entregan el flujo del proceso. Los ***servlets*** son los objetos control del sistema desarrollado.

El modelo ***MVC*** es relativamente sencillo, pero a la vez poderoso, porque en un sistema cambiante, se modifican los métodos de las clases de control (y las auxiliares que use); las clases de modelo (*JavaBeans*) cambian poco, mientras que la vista (***JSPs***) no se altera o tiene cambios mínimos. También permite (a diferencia de la arquitectura foliocéntrica) que dos o más personas trabajen a la vez en un módulo, ya que un diseñador puede estar modificando los ***JSPs***, mientras que un programador cambia las reglas de negocio o el proceso que efectúa un servlet. Otra ventaja adicional es que la navegación por la aplicación web no se afecta mucho mientras se hacen los cambios.

El modelo anterior fue expandido en las arquitecturas web de despachador, y en específico, se implantó la arquitectura ***Servicios al Trabajador***[50], la cual hace una separación más clara de la presentación y del procesamiento; esta arquitectura se recomienda cuando se tiene que hacer mucho procesamiento de la petición, por la cantidad de información de una forma *HTML*, o en este caso, para recibir y procesar los parámetros para controlar la videocámara.

Con base en el modelo antes descrito, se desarrollaron las pantallas (***JSPs***), los cuales son vistos por el usuario, despliegan información y tienen interacción con el mismo. Las peticiones del usuario son enviadas a los ***servlets***, los cuales procesan las peticiones (junto con los parámetros), llaman a otras clases, tienen interacción con la base de datos, crean y alimentan de datos a objetos ***JavaBeans***, los cuales son enviados como atributos (de petición, sesión, o de aplicación), y re-direcciona la aplicación a otro ***JSP***, el cual recupera los ***JavaBeans***, y se despliega al usuario, mostrando los datos contenidos en los *JavaBeans*.

Cuando se comenzaron a integrar los módulos de control del puerto paralelo y de la videocámara con los de web, apareció otro problema distinto. Dichos módulos se habían desarrollado y probado como aplicaciones de consola (un sólo usuario), pero dentro del ambiente de web pasaban a un ambiente web multi-usuario, en la que peticiones simultáneas de usuarios podrían causar conflicto por el control de dichos periféricos. En este caso, ya se había desarrollado parte del sistema, y se regresó de nuevo a la fase de diseño, lo cual es permitido por *RUP*. Para resolver el problema, se implantó el modelo ***Singleton*** (un único elemento)[39] tanto para el control del puerto paralelo como de la videocámara en nuevas clases sincronizadas; al recibir peticiones simultáneas que involucren el puerto paralelo o la videocámara, se van atendiendo las peticiones una por una (comenzado por la primera que llegó), y hasta que se acabe de atender una petición, se atiende la siguiente.

Al comenzar a integrar el módulo de base de datos con la aplicación web, también se regresó de la implantación (construcción) al diseño. Aunque al principio de la construcción del sistema web ya se tenía código que tenía interacción con la base de datos, se le dio mayor preferencia al desarrollo e integración de los módulos del puerto paralelo y de la videocámara, para concluir el módulo de base de datos al último; *RUP* también recomienda dar mayor prioridad a las partes más complejas del proyecto. Para concluir e integrar el módulo de base de datos con el resto de la aplicación web, se amplió la arquitectura ya existente.

Para poder almacenar información en la base de datos e integrar los demás módulos, se implantó el modelo ***DAO*** (*Data Access Model*, Modelo de Acceso a Datos)[48], el cual consta de los siguientes componentes.

1. ***DAO***, que son clases sencillas que contienen la lógica de manipulación de datos. El intercambio de información se hace con un objeto *Data Source* (fuente de datos); cada ***DAO*** contiene los métodos de consulta, inserción, modificación y/o borrado relacionados con una entidad (tabla de la base de datos).
2. ***Business Objects*** (objetos de negocio), los cuales crean e invocan los métodos de los objetos ***DAO***.

En este sistema, los ***servlets*** se convierten en ***Business Objects***, sin tener que reemplazarlos; los ***servlets*** ya no hacen consultas directas a la base de datos, removiendo las sentencias *SQL* de los ***servlets*** y pasándolas a un objeto ***DAO***. Los ***servlets*** siguen controlando el procesamiento de peticiones y flujo de la aplicación.

3. ***Data Source*** (fuente de datos), es una clase que encapsula el origen de los datos; le da un nombre y forma de acceso único al resto de la aplicación. Se creó un ***Data Source*** en ***Apache-Tomcat***, aprovechando las nuevas características de ***Tomcat*** versión 5 en adelante, reemplazando a la clase *PozoConexiones.java*, que creaba y administraba varias conexiones; los nuevos objetos ***DAO*** buscan la fuente de datos, y le solicitan una conexión a la base de datos.
4. ***Transient Objects*** (objetos de transferencia), los cuales llevan datos de los ***Business Objects*** a los ***DAO***, y viceversa. En este sistema, los *JavaBeans* son usados como ***Transient Objects***; no se cambia la estructura de los *JavaBeans*, sino la forma de utilizarlos. El mismo tipo de *JavaBean* (modelo) con que se pasan datos de un ***servlet*** (controlador) a un ***JSP*** (vista) en la arquitectura ***MVC***, ahora sirve como objeto de transición, reusando código y ampliando el papel de estos componentes.

Estos elementos se crearon para modelar las tablas de la base de datos relacional; cada tabla se representa por una clase de tipo objeto de transferencia, y cada columna se convierte en un atributo de la clase. No se usó ninguna herramienta automatizada para crear estos objetos a partir de la base de datos.

La última parte del diseño fue la de seguridad; en esta aplicación web ya se había construido un módulo de administración, pero no todos los usuarios lo deberían usar, sólo los operadores registrados podrían detener la videocámara o cambiar permisos a otros usuarios. Por otra parte, tampoco se quería restringir el acceso a toda la aplicación, ni rehacer el código. Se modificó la base de datos, cambiando la tabla ***niveles***, para indicar los papeles o funciones de los usuarios del sistema; se crea otra llamada ***niveles_usuario***, la cual relaciona a un usuario con un nivel (función o papel)[43].

En la configuración del ***Apache-Tomcat*** en *web.xml* se indican los mismos papeles o funciones contenidos en la tabla niveles, y se delega a ***Tomcat*** la autenticación de usuarios.

2.2 Análisis y diseño de los Componentes Físicos.

2.2.1 Análisis del circuito de control de movimiento de la videocámara.

Originalmente el circuito empleado para las pruebas de desarrollo era un circuito formado por *buffers* de salidas de tres estados SN74LS241N (para aumentar la señal del puerto paralelo), y circuitos de tipo *latch* SN74LS373N (para retener el valor del bit proveniente del *buffer*), cuya salida se desplegaba en *leds* -con unas resistencias de por medio, para reducir la corriente.

Más adelante, cuando se hicieron varias pruebas satisfactorias de enviar valores y secuencias por el puerto paralelo desde una página web (encendiendo y apagando leds del circuito mencionado anteriormente), se procedió a diseñar un circuito electrónico de control.

Por una parte, los datos de salida del puerto paralelo debían conservarse temporalmente, antes de que se recibiera el siguiente byte por el mismo puerto, dando tiempo a que el circuito electrónico produjera una respuesta ante tal valor de entrada, pero se descartó el uso de circuitos electrónicos sofisticados, como los procesadores digitales de señales (*DSPs*, *Digital Signal Processors*).

Por otra parte, se requería que la camara se moviera con precisión y que se desplazara uniformemente, por lo que se optó por usar motores de paso, los cuales siempre rotan la misma cantidad de radianes en cada paso. También se seleccionó que dieran un paso completo para transferir el movimiento directamente a la videocámara, y no de medio paso -que regresa a su posición original, transfiriendo el movimiento a través de un sistema de engranes.

La configuración del alambrado de control le permite al motor de pasos dar un paso completo en un sentido, al enviar una secuencia de cuatro señales por los cuatro bits de datos (provenientes del puerto paralelo), alterando el orden en que se alimentan las bobinas del motor, y por efecto del juego de atracción y repulsión electromagnética se lleva a cabo el movimiento de motor. El orden en que se activan y desactivan las bobinas se muestra a continuación:

Paso	Bobina 1	Bobina 2	Bobina 3	Bobina 4
1	Alto	Bajo	Bajo	Bajo
2	Bajo	Alto	Bajo	Bajo
3	Bajo	Bajo	Alto	Bajo
4	Bajo	Bajo	Bajo	Alto

Tabla 17: Secuencias de pasos para el motor bipolar.

La lógica de programación del Módulo de Control del puerto paralelo indica la secuencia indicada para girar en uno u otro sentido; ya que se usa dicho puerto para el control de dos motores (movimientos horizontal y vertical) simultáneamente, la lógica de programación también indica que se envíen los mismos datos de los cuatro bits correspondientes al motor que no se mueve (evitando que se altere la polaridad de las bobinas), tal y como se indica en ***Análisis y Diseño de la Aplicación***.

2.2.2 Diseño del circuito de control de movimiento.

El circuito de control se diseñó como un circuito de control abierto, el cual efectúa el cambio de posición (horizontal o vertical) a voluntad del usuario, sin una retroalimentación del cambio de posición. Por lo anterior, se decidió usar piezas relativamente simples y de bajo costo.

Para poder mover la videocámara, se escogió un motor de pasos (uno para el movimiento horizontal y otro para el vertical), ya que se puede controlar con precisión los movimientos de cada motor, independientemente de la velocidad del microprocesador del equipo servidor.

Cabe mencionar que se pudiera construir un circuito con motores de corriente directa que se movieran más rápido que uno de pasos, pero el problema es que para que se muevan uniformente los motores, el lapso entre que se inicia y termina el movimiento tendría que ser constante; esto podría resolverse mediante programación, al tener un contador que inicie, se verifique a intervalos si el lapso de permitir movimiento no ha sido excedido, y si es así, detener el movimiento. Sin embargo, si el equipo donde se ejecuta ese programa tiene mucha carga de trabajo, pudiera no mandar la orden de detener movimiento a tiempo.

También se podrían hacer circuitos con contadores y motores de corriente directa, pero este tipo de circuitos tendría que ser calibrado uno por uno (si se hacen varios de ellos) o usar componentes eléctricos de precisión. En ambos caso aumentaría el costo de los circuitos, en comparación con el circuito con motores de paso.

Por lo anterior se prefirió el circuito con motores de paso, y para crearlo se alambró un circuito con una etapa de potencia que alimenta a un motor de pasos PM55L-048-HPG9; el circuito integrado empleado (por motor) es un ULN2003AN al cual le llegan cuatro bits de datos del puerto paralelo; estos circuitos consisten en arreglos de diodos *darlington*, los cuales permiten acoplar circuitos de lógica *TTL* (de 5 [V]) con la fuente de 12 [V], 300 [mA], que alimenta a los motores.

El motor de pasos PM55L-048 (HPG9 es su serie) es un motor bipolar, lo que significa que puede girar en ambos sentidos al alternar la alimentación de las bobinas. Tiene 5.5 [cm] de diámetro, y requiere de 48 pasos para una revolución completa (gira 7.5º por cada paso).

El circuito electrónico empleado es el siguiente:

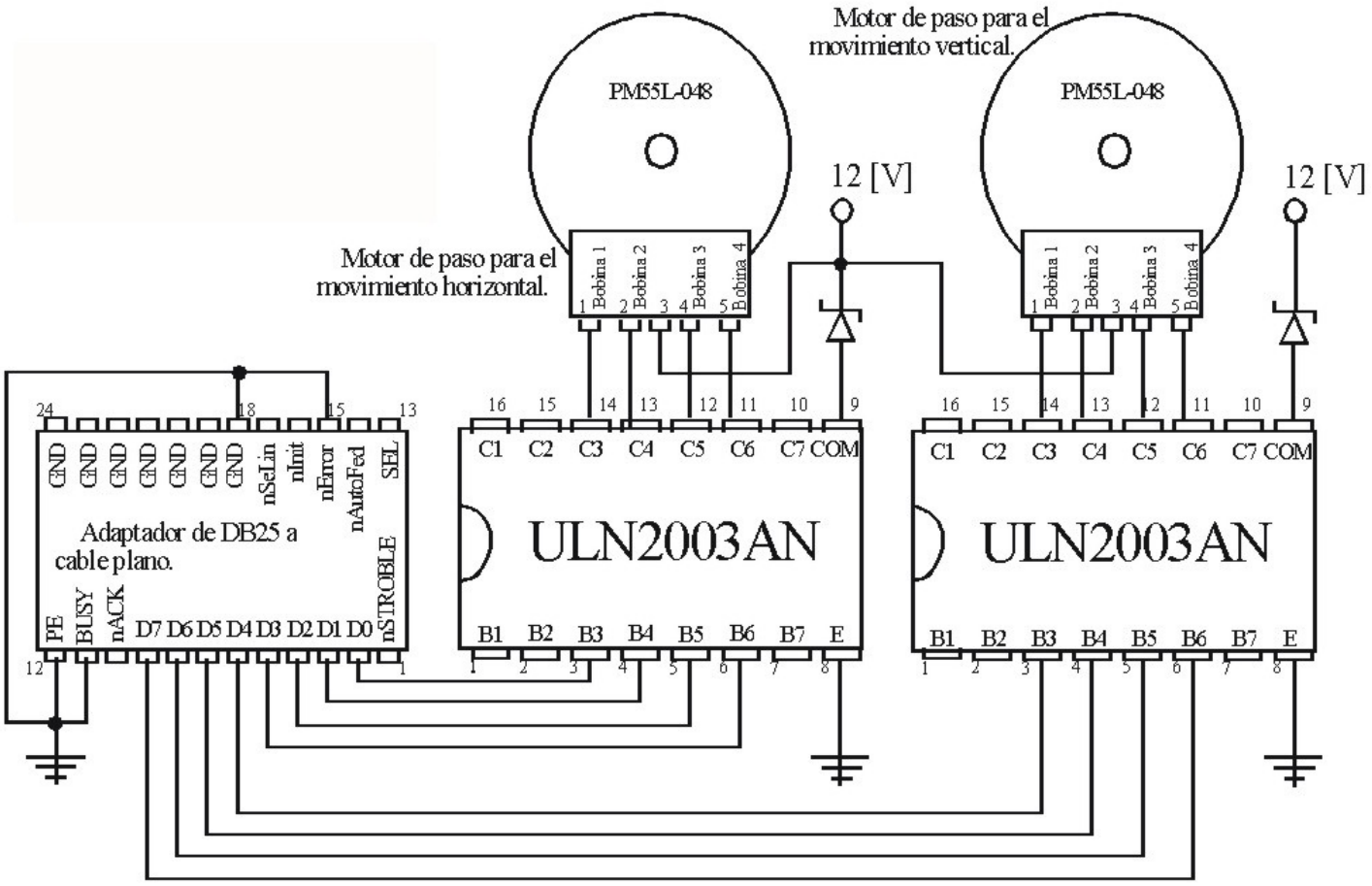

Figura 24*: Diagrama del circuito electrónico de control horizontal y vertical de la videocámara web.*

En el circuito mostrado, se utilizaron las entradas B3 a la B6 (y sus correspondientes salidas C3 a C7) en el *ULN2003AN* sólo por cuestión de espacio en la tableta *protoboard*, ya que originalmente se iba a alambrar otro circuito. Se utiliza un diodo *zener* para conectar de común a la fuente de 12 volts, ya que esto se recomienda para evitar cualquier corriente inversa que pudiera venir de la fuente.

2.3 Implantación y construcción de la aplicación.

2.3.1 Implantación de la aplicación.

La implantación (construcción) de la aplicación comenzó por la instalación de las aplicaciones con las que se construiría el sistema:

- *Java Development Kit* (*JDK*), el cual provee el compilador del lenguaje java, y otras herramientas de programación.
- *Java Runtime Enviroment* (*JRE*), el cual le da soporte para la ejecución de aplicaciones java; al principio no se usaba para desarrollo, pero más adelante las últimas versiones de ***Apache-Tomcat*** pueden ejecutarse con el *JRE* y ya no exclusivamente con el *JDK*.
- Administrador de base de datos *MySQL*.
- Contenedor de ***servlets Apache-Tomcat***.
- Interfaz *Java Media Framework 2.1.1e* con *Performance Pack* para *Windows*.
- Interfaz *Java Communications 2.0* para *Windows* (con librería dinámica de 32 bits).
- *JDBC Connector/J* para *MySQL*.
- Entorno de desarrollo integrado (*IDE*, *Integrated Development Enviroment*).

Durante el desarrollo del sistema se cambiaron las versiones de las aplicaciones mencionadas anteriormente; se cambió el entorno de desarrollo integrado para poder utilizar la depuración de código en línea y de otras ventajas que ofrecían, mas no el código fuente ya desarrollado previamente. La construcción del sistema inició sobre *Windows Millenium*, debido a que era más accesible una computadora personal con dicho sistema operativo.

Se decidió hacer, al principio, clases de prueba por cada módulo, para verificar que se podía hacer uso del puerto paralelo, que se podía conectar y manipular información de la base de datos, hacer ***servlets*** y que respondieran generando *HTML*, y más adelante, se comenzaron a hacer las clases para manipular la multimedia de la videocámara. Cada módulo se comenzó a desarrollar individualmente como si fueran aplicaciones de consola (excepto la de web) creando clases del lenguaje java con la funcionalidad más básica de su módulo, y siguiendo las convenciones de nombres de clase y de variable que recomienda *Sun Microsystems*.

Después de hacer clases que efectuaran las acciones más básicas de su módulo, resolviendo los problemas técnicos del mismo, se hicieron otras versiones en las que se incluía la funcionalidad ya probada, agregando métodos nuevos, y eliminado el uso de rutas fijas y otras configuraciones del código, buscando que estas versiones tuvieran la función más general, mientras que para tareas más específicas se crearon otras clases, que importan las clases más genéricas (composición); en algunas se comenzó a aplicar el concepto de herencia, para no re-escribir todo el código, usar los métodos heredados como propios a la vez de agregar sus métodos y atributos propios. Estas clases fueron probadas como aplicaciones de consola, por módulo, e independientes entre sí. De forma más específica, el desarrollo por módulo fue el siguiente:

- Módulo del puerto paralelo: Fue el primer módulo en diseñarse y construirse; se crearon clases que identificaban los puertos locales de la computadora, se abría el puerto paralelo construyendo flujos para enviar un dato numérico, y más adelante, se enviaban secuencias, verificando en un circuito *protoboard* con memorias *buffers* -las cuales retienen el valor recibido del puerto paralelo hasta que reciban otro dato- y *leds* (diodos emisores de luz), que estos últimos se prendieran y apagaran para mostrar en binario el número enviado. Una vez que se consiguió dicha funcionalidad, se creó la clase **Paralelo16.java**, en la cual se colocan los métodos relacionados con el uso del puerto paralelo, tales como identificar y abrir el puerto paralelo, abrir un flujo de datos hacia el mismo, enviar de un dato numérico, y cerrar el puerto. Se delega

a otras clases el envío de datos específicos y/o secuencias, las cuales tienen que crear una instancia de **Paralelo16.java**, y hacer usos de sus métodos. Cabe mencionar que estos códigos se desarrollaron y probaron en *Windows 9x* y *Millenium*, funcionando correctamente sobre estas plataformas.

- Módulo de base de datos: Se comenzó más adelante, con clases de prueba que cargaban el controlador ***JDBC***, se abría la conexión a la base de datos con una *URL* fija en el código, y se hacían consultas simples, revisando que se pudiera recuperar información de las consultas; después se hicieron otras clases de prueba para comprobar que pudiera efectuar inserciones, borrados y ejecutar *stored procedures* hacia *MySQL*. Al resolver los problemas de configuración de la *URL* de conexión, se comenzó a crear otras clases. Para conectarse a *MySQL* se había creado la clase **ConexionGeneralBD.java**, la cual carga un archivo de propiedades, construye la *URL* de conexión, y se encarga de conectarse y desconectarse a la base de datos. Otra de ellas era **PozoConexiones.java**, la cual cargaba el controlador ***JDBC***, leía un archivo de propiedades, creaba y almacenaba varias conexiones a la base de datos, para que posteriormente proporcionara una conexión cuando se le solicitara, con el fin de administrar las conexiones activas a la base de datos; más adelante fue reemplazada al configurarse una fuente de datos en ***Apache-Tomcat*** que tenía la misma funcionalidad, pero que permite ser llamada desde cualquier parte de la aplicación web, ahorrando el esfuerzo de crear o llamara a instancias de **PozoConexiones.java** para obtener una conexión a *MySQL*. Como este módulo no era tan crítico, se pospuso esta parte del desarrollo, para darle mayor prioridad al módulo de video y a su integración en web, tal y como aconseja *RUP*. Se hizo la clase **OperacionesGenericasMySQL.java**, la cual contiene métodos genéricos que reciben cadenas y efectúan operaciones ***DML*** (*Data Manipulation Language*) en la base de datos *MySQL*, regresando valores *booleanos*, enteros u objetos *ResultSet*.
- Módulo de video: Este es el módulo más complejo, y que requirió más tiempo y esfuerzo, a pesar que la interfaz ***Java Media Framework*** prometía ser fácil de usar. Comenzó por buscar el medio con el que se controla el video de la videocámara de estado sólido, utilizando la interfaz ***JMF***. Al igual que los otros módulos, se comenzó por desarrollar clases que se utilizaran como aplicaciones de consola, las cuales primeramente, buscaban conectarse a la videocámara para activarla y apagarla.
- Módulo de web: Se comenzó por construir ***servlets*** y ***JSPs*** de prueba que generaran una respuesta ***HTML*** sencilla, los cuales eran llamados desde *Internet Explorer* y *Netscape*. Después se inició con la construcción de otras páginas con base en la arquitectura ***MVC*** (*Model-View-Controller*, Modelo-Vista-Controlador), creando ***servlets***, ***JavaBeans*** y ***JSPs*** para la presentación. También se había iniciado la construcción de una pantalla de autenticación de usuarios con *Base64* y un archivo de usuarios y contraseñas, sobre ***Apache-Tomcat 3x***. En las versiones de ***Tomcat 3x*** y ***4x*** se configuraba *server.xml* para indicar cuál era la aplicación web (**videocamara**, en un directorio del mismo nombre), mientras que los archivos *class* de los ***servlets*** se colocaban manualmente en un sub-directorio de *WEB-INF*; los ***JSPs*** se colocaban también manualmente en un sub-directorio de **videocamara**. Este módulo fue concluido hasta el último, ya que los módulos de control del puerto paralelo y de video tenían mayor prioridad.

El código de cada módulo se comenzó a organizar en paquetes. *RUP* permite el desarrollo iterativo, lo que significa que se puede regresar a fases anteriores, para terminar de detallar o redefinir partes de los módulos; también señala que hay que darle mayor prioridad a partes más críticas del proyecto, dejando al último las tareas más sencillas o que consumen más tiempo. Algunas clases de este sistema se crearon con editor de texto, y se compilaban y probaban en una ventana de consola, pero para poder acelerar el desarrollo y depuración de código se comenzaron a usar ambientes de desarrollo integrado, el primero era *JBuilder*, y más adelante se cambió por *JDeveloper*.

El desarrollo del módulo del puerto paralelo, utilizándolo como aplicación de consola en esta etapa, no tuvo sobresaltos, ya que se enviaban datos numéricos por dicho puerto de forma exitosa, requiriendo pausar el proceso entre cada envío de dato, para garantizar que el circuito electrónico respondiera adecuadamente ante cada dato recibido; se utilizó el método *Thread.sleep(int tiempo)* para detener el proceso a intervalos regulares, y no contadores

con decremento, debido a que el lapso de pausa varía de acuerdo con la velocidad del microprocesador. La forma en que trabaja la clase **Paralalelo16.java** para el control y comunicación con el puerto paralelo de la computadora es la siguiente:

1. Se crea una instancia de **Paralelo16.java**.

2. Se llama a *public synchronized void abrePuerto()*.

   2.1 Obtiene los puertos locales disponibles.

   2.2. Busca dentro de los puertos encontrados los que sean de tipo paralelo, y cuyo nombre sea "**LPT1**".

   2.3 Abre el puerto paralelo "**LPT1**", registrando a "**Paralelo16**" como la aplicación que usa el puerto.

   2.4 Abre un flujo de salida (*OutputStream*) del puerto.

3. Llama al método enviar (tantas veces como se desee), el cual transmite un entero por el flujo de salida.

4. Finalmente, se cierran los flujos de puerto, y el puerto mismo.

Como se había indicado, *RUP* aconseja darle mayor prioridad a las partes más críticas. El módulo de video fue el que requirió mayor esfuerzo y diversas pruebas para controlar el dispositivo, y luego obtener una imagen fija. Al instalar la interfaz ***Java Media Framework***, también se instaló una herramienta llamada *Java Media Player*, la cual se utilizó como herramienta de diagnóstico para verificar que la videocámara y la interfaz JMF estuvieran instaladas correctamente, pero no se usó como parte del sistema, ni se aplicó ingeniería inversa para revisar el código de la misma. Se revisaron ejemplos del uso de ***JMF*** pero ningún ejemplo resolvía el problema de obtener una imagen fija, por lo que se revisó la documentación y tutoriales con los cuales se entendió cómo trabaja la interfaz.

Después de hacer varios códigos de prueba para activar la videocámara, obtener video, detenerla y cerrar el medio, quedaba el problema de obtener una imagen (correspondiente a un cuadro de video) en un formato conocido que pudiera desplegarse en una página ***HTML***. En las pruebas iniciales se había encontrado que el sistema operativo (*Windows Millenium*) usaba *Video for Windows*, y la videocámara utiliza el formato ***AVI***. Se hicieron unas clases para transmisión y recepción de video en formato ***RTP*** (***Real Transfer Protocol***), y de ahí obtener una imagen fija en un formato conocido; este formato tiene menor resolución, y se encontró el mismo problema de que no se podía obtener una sola imagen con los métodos ya existentes.

El problema es que se podía iniciar y detener el proceso de toma de video por parte de la videocámara, y una vez iniciado el proceso de grabación (previamente configurado), no se tiene el control directo para manipular el video obtenido; considerando que el video es una secuencia de cuadros de imágenes, se buscó la forma de obtener uno de ellos del video de forma programática. La videocámara y la tarjeta de video usan Video para *Windows* (***VfW***, ***Video for Windows***) para obtener imágenes; la interfaz ***JMF*** puede usar el formato ***API***, por lo que se buscó la manera de que buscara una fuente de video de tipo Video para Windows, y que el video obtenido lo convirtiera a formato ***AVI*** (que es un formato contenedor), y de ese formato extraer la pista de video.

Una vez extraído el video, se implantan ***códecs*** propios que reciben un bloque de datos binarios sin formato (crudos) del flujo de video, que corresponden a un cuadro de video, se les da el formato ***JPEG***, y se guardan en un archivo en disco duro; debido a que no se podía calcular cuándo se tenía una imagen capturada y guardada, ni un

control preciso sobre el proceso de captura de imágenes una vez iniciado, se implantó la generación de un evento y su auditor correspondiente, para avisar que se tiene una nueva imagen capturada y guardada, y detener el proceso de grabación del video, para evitar sobre-escribir la imagen ya obtenida.

El código del módulo de video que implanta la funcionalidad descrita en el párrafo anterior, la efectúa la clase **CuadroVideo05.java**, la cual lleva a cabo los pasos básicos para utilizar un dispositivo multimedia con la interfaz ***Java Media Framework***. Dichos pasos son los siguientes:

1. Busca un dispositivo de multimedia que corresponda con el indicado en el archivo de propiedades - uno que maneje *Video for Windows*.
2. Se obtienen los formatos que maneja el dispositivo multimedia.
3. Obtiene una instancia de *MediaLocator*, el cual es equivalente a una *URL*, y sirve como referencia para tener acceso a la fuente multimedia.
4. Llama al método ***creaProcesador***, el cual utiliza a la instancia de ***MediaLocator*** para invocar al administrador(***Manager***) y crear una fuente de datos (***DataSource***); el administrador es el componente que se encarga de llamar a las librerías nativas para comunicarse con el sistema operativo, y tomar el control del hardware de un dispositivo multimedia. El objetivo de este método es crear un procesador y no un reproductor porque es más difícil desplegar el reproductor (que tiene interfaz gráfica propia) en una página web, y que tenga interacción de forma remota con el dispositivo multimedia; por otra parte, un procesador tiene los controles para manipular al dispositivo multimedia (invocados desde el ***servlet***), y si bien carece de interfaz gráfica, el ***JSP*** (creado posteriormente) le proporciona una.
5. A partir de la fuente de datos, siguiendo dentro de ***creaProcesador***, se crea un procesador, el cual se encarga de obtener los datos multimedia del dispositivo, y darles formato.
6. El procesador pasa del estado **Sin Realizar** (estado inicial) al de **Configurado**, pasando por el de **Configurando**.
7. Se obtienen las pistas multimedia del procesador; para la videocámara se obtienen dos pistas, una de audio y otra de video.
8. De las pistas obtenidas, se revisa cuál de ellas corresponde a Video, y se procede a darle formato, el cual tiene base en un formato de ***JPEG*** al cual se le indica la velocidad de encuadre (cuántos cuadros de video toma por segundo), y el alto y ancho de la imagen a obtener (tomando en cuenta las limitaciones del formato ***AVI***).
9. Se crea una instancia de **AccesoPosteriorCodec**, la cual es una clase interna que extiende **AccesoPrevioCodec**, la cual implanta la interfaz *códec*. El objetivo de **AccesoPosteriorCodec** es recuperar el flujo de datos que corresponde a un cuadro de video recién tomado de la videocámara, descomprimir los datos dándoles el formato ***JPEG***, guardar esos datos formateados en un archivo con extensión ***JPEG***, y generar un evento de tipo **EventoImagenGenerada** para avisar que se obtuvo un cuadro de video de la videocámara y que se guardó en disco duro; ese evento y la implantación de su correspondiente auditor (**Listener**) fueron diseñados específicamente para este caso, ya que la videocámara, una vez que inicia, sigue un proceso independiente que toma imágenes continuamente.
10. Se especifican los *códecs* de entrada y salida de datos (con base en **AccesoPosteriorCodec**, creado en el paso anterior), y se encadena a con los otros *códecs* que maneja la pista de video.
11. Se le da la descripción del tipo de media al procesador -en este caso, ***CONTENT_UNKNOWN***, porque no se puede saber con certeza que formatos de video maneje la pista de video.
12. Llama al método ***esperaEstado***, y se espera hasta el que procesador pase al estado de **Realizado**.
13. Especifica un factor de calidad ***JPEG*** con el que se generarán las imágenes.
14. Obtiene una instancia de **DataSource** del procesador ya realizado.
15. Sale del método ***creaProcesador*** regresando verdadero.
16. Llama al método continuar, el cual revisa que el procesador haya sido construido correctamente, y lo arranca para que comience a tomar imágenes.

Por otra parte, la clase **CuadroVideo05.java** cuenta con varios atributos booleanos que indican el estado del procesador, de la fuente de datos y si ha conseguido una nueva imagen; cuenta con un atributo de tipo *File*, el cual indica ruta y nombre del archivo generado, en el cual se guardó una nueva imagen. Así mismo, cuenta con varios métodos con los cuales se manipula a la videocámara, los cuales se explican a continuación:

- *public void controllerUpdate(ControllerEvent ce)*, este método realiza la interfaz ***ControllerUpdateListener***, sincronizando el código de la aplicación con los procesos de las librerías nativas que controlan directamente los dispositivos multimedia, asegurando que el procesador pase de un estado a otro correctamente.

- *public boolean esperaEstado(int estado)*, similar al anterior, pero la diferencia consiste en que el método anterior responde a eventos, arrojados por el mismo procesador, mientras que este responde a una solicitud de cambio de estado (identificado por un entero).

- *public synchronized boolean detener()*, el cual termina la obtención de imágenes del procesador.

- *public synchronized boolean continuar()*, el cual reinicia con la obtención de imágenes de la videocámara.

- *void setCalidadJPEG(Player p, float valor)*, el cual asigna un factor de calidad que esté en el intervalo (0, 1), tomando en cuenta cuáles formatos de salida son del tipo ***JPEG***.

- *private synchronized boolean esperaEstado(Processor p, int estado)*, similar a los mencionados anteriormente, pero recibe como parámetros la instancia de procesador, y el estado al cual pasa; ayuda a la transición de estados, sincronizando las tareas del procesador con el código de esta clase.

- *public void imagenCreada(EventoImagenCreada recibeEvento)*, el cual implanta la interfaz **ImagenCreadaListener**, con la cual se auditan (o esperan) los eventos de tipo **EventoImagenCreada** que arroja la clase **AccesoPosteriorCodec** (que se ejecuta en un proceso independiente al de **CuadroVideo05**).

- *public synchronized void cerrar()*, el cual detiene a la videocámara, cerrando los flujos del mismo, y liberando los recursos multimedia del dispositivo.

Los módulos de control del puerto paralelo y de video se desarrollaron (en una fase intermedia) y probaron como si fueran aplicaciones de consola (mono-usuario), pero para integrarse en el ambiente de web (multi-usuario) se encontró el problema de concurrencia hacia recursos únicos, en este caso el puerto paralelo y la videocámara de estado sólido. Se regresó a la etapa de diseño, para resolver este problema; una posible solución era sincronizar los métodos que utilizaran el puerto paralelo o la videocámara. La otra solución era el modelo o patrón **Singleton**[39, 49], el cual garantiza que existe una sola instancia de dicha clase (mediante un constructor privado) en la máquina virtual de java, lo que permite representar con mayor exactitud al puerto paralelo y a la videocámara (ambos son objetos físicos y únicos en la computadora personal); al crear una sola instancia permite que inicie sólo una vez, sin tener que configurar nuevamente la clase, y también permite que se pueda invocar desde cualquier parte de la aplicación web.

Para no re-escribir código también se aplica el concepto de herencia, para crear las clases **PuertoParalelo.java** y **VideoCamara.java**, las cuales extienden a las clases **Paralelo16.java** y **CuadroVideo05.java**, y los métodos nuevos o los sobre-escritos están sincronizados para resolver el problema de concurrencia. Estas clases se probaron nuevamente como aplicaciones de consola, trabajando correctamente, por

lo que se procedió a construir los elementos de web del modelo ***MVC*** para invocar a ***PuertoParalelo*** y a ***VideoCamara***.

Se creó el ***servlet*** **ControlPuertoParalelo.java**, su correspondiente ***JSP*** y unos *JavaBeans* para guardar datos de la posición relativa de la videocámara, sobre *Apache-Tomcat 4.x*, en la cual sólo se indicaba el nombre de la aplicación web, mientras que los archivos compilador (*class*) se colocaban manualmente en una ruta de la aplicación web (sub-directorio de *WEB-INF*), mientras que el *JSP* se colocaba en otro sub-directorio de la misma aplicación. Se comenzó a tener problemas entre ***Apache-Tomcat*** y la interfaz ***Java Communications***, hasta que las librerías de este última y su archivo de propiedades se colocaron dentro de las librerías externas del ***JRE*** (***Java Runtime Enviroment***), probando que se controlara el envío de datos por el puerto paralelo desde un ambiente de web.

La incorporación del módulo a la aplicación web presentó problemas adicionales, ya que si bien la clase **VideoCamara.java** trabajaba correctamente como aplicación de consola, dentro del ambiente de web no funcionaba correctamente al inicio. Las librerías de la interfaz ***JMF*** se habían colocado en el sub-directorio *lib* de ***Tomcat***, en *JAVA_HOME/lib*, *JRE/lib* (directorios de instalación del compilador y del ambiente de java, respectivamente) y en el directorio *WEB-INF/lib* de la aplicación web **videocamara**, se activaba y desactivaba la cámara, pero no se obtenía ninguna imagen; se agregaron los archivos de propiedades y de configuración de ***JMF*** al directorio *WEB-INF\lib* de la aplicación web, y se comenzó a obtener imágenes apareciendo otros problemas de sincronización de la clase *VideoCamara.java* con el ***servlet*** **controlvideocamara.java**, debido a que este último esperaba indefinidamente, o respondía inmediatamente enviando un *JavaBean* sin datos al ***JSP*** **controlImagen.jsp**.

Mediante prueba y error se terminó de modificar la clase **VideoCamara.java**, para intentar hasta tres veces la captura de la imagen, y en caso de no obtenerla, el ***servlet*** **controlvideocamara.java** presentaría otro ***JSP*** con un mensaje de error. Se decidió juntar en un sólo ***JSP*** la presentación de la imagen capturada, datos de la misma y los controles de cambio de posición de la videocámara, para que los usuarios puedan manipular ese dispositivo, recibiendo la misma página pero con la nueva imagen en la nueva posición y con datos actualizados. La funcionalidad del control se concentró en **ControlDireccionPuerto01.java**, ya que en cada cambio de posición se debe capturar y presentar una nueva imagen; se siguieron utilizando los *JavaBeans* **Punto.java**, **Coordenada.java**, **Direccion01.java** y **DatosImagen.java**, ya que representan los cambios de posición, la posición actual y los datos de la imagen recién capturada.

Estos ***servlets*** y ***JSPs*** se probaron y ejecutaron en ***Apache-Tomcat 4x***, colocando los archivos compilados (***servlets***, *JavaBeans* y archivos auxiliares) en sub-directorios de *WEB-INF*, mientras que los ***JSPs*** se colocaban en otros sub-directorios de la aplicación web. En la arquitectura de esta versión de ***Tomcat*** los ***servlets*** se actualizan de forma automática al cambiar el archivo ***class***, mientras que las clases estáticas se cambian al reiniciar el contenedor de ***servlets***. Mientras que la aplicación se desarrollaba y crecía, apareció el problema de tener actualizadas las clases presentes en ***Tomcat***, por lo que se creó un archivo *WAR* (*Web ARchive*) para copiar todas las clases actualizadas, ***JSPs***, archivos de propiedades, *HTMLs* e imágenes fijas, del ambiente de desarrollo a ***Tomcat***. También permitió copiar las librerías y archivos de propiedades de las interfases externas utilizadas (***Java Communications***, ***Java Media Framework*** y *J/Connector*) al sub-directorio *WEB-INF\lib*, para que en el caso de tener que actualizar alguna de estas librerías, se volviera a generar el archivo *WAR* y no tener que cambiar la configuración de ***Tomcat***. Esta tarea se efectuó de forma automática con el entorno de desarrollo integrado (*JDeveloper* y/o *NetBeans*), el cual crea un archivo con el mismo tipo de compresión que los archivos *ZIP*, con la estructura de la aplicación web.

El código desarrollado y probado hasta ese entonces se había hecho en ***Apache-Tomcat 4x***, sobre el sistema operativo *Windows Millenium*, pero por diverso problemas con dicho sistema operativo se tuvo que cambiar a *Windows XP*, el cual se presentó como una plataforma más robusta y segura que las versiones anteriores de *Windows*.

Cuando se migró la aplicación de *Windows Millenium* a *Windows XP* con *Service Pack 1*, se cambió el nombre con que se identifica al Video para *Windows*, por lo que cambió el nombre y se re-compiló la aplicación web, funcionando, hasta ese momento, como lo hacía anteriormente.

También se cambió la versión de ***Apache-Tomcat*** a la 5 (y más adelante a la 6), para aprovechar las mejoras de seguridad, por lo que se cambió la forma de conexión a base de datos creando una fuente de datos reemplazando a las clases **ConexionGeneralBD.java** y **PozoConexiones.java**; esta fuente de datos se utilizaría para la aplicación web como para implantar un reino de seguridad.

Para que el sistema tuviera interacción con la base de datos se regresó a la etapa de diseño, buscando cómo guardar y recuperar datos más fácilmente; ya se había probado que el sistema web podía efectuar operaciones ***DML*** con *MySQL* a través de ***JDBC***, pero se tendría que programar mucho código relacionado con base de datos en ***servlets***, y pasarle datos a los ***JSPs***. Se encontró que la arquitectura ***DAO*** (***Data Access Object***)[48] podría ayudar.

Se crearon objetos ***DAO***, los cuales son clases simples del lenguaje java, pero contienen las sentencias ***DML*** (consultas, borrados, inserciones, modificaciones) para *MySQL*; se movió el código de algunos métodos de los ***servlets*** a los objetos ***DAO***, con el objetivo de que ningún ***servlet*** efectuara consultas a la base de datos directamente, sino que enviaran o recibieran objetos de transferencia a o desde de un objeto ***DAO***, y este último se encargue de efectuar la consulta. Los ***JSPs*** tampoco harían consultas a la base de datos, sino que recibirían *JavaBeans* o vectores con *JavaBeans*, para desplegar información. Los *JavaBeans* ya existentes y los nuevos se utilizaron como objetos de transferencia. Se modificó la clase **OperacionesGenericasMySQL.java**, para encontrar la fuente de datos y obtener de ella una conexión hacia *MySQL*; los objetos ***DAO*** prepararan las sentencias *SQL*, incluyendo sentencias almacenadas (*Stored Procedures*), e invocan a los métodos de **OperacionesGenericasMySQL.java**, los cuales efectúan las consultas, borrados, modificaciones e inserciones, incluyendo los procedimientos almacenados (para sentencias *SQL* repetitivas).

Los objetos de transferencia, que son *JavaBeans*, se construyeron manualmente creando uno por cada tabla de la base de datos; cada uno tiene un atributo por cada columna de la tabla; existen herramientas que efectúan esta tarea de forma automática. Algunas de estas clases se emplean como *JavaBeans* en el modelo ***MVC***[49] implantado (o extendiendo el modelo hacia ***Servicios al Trabajador***[50]).

Para guardar o recuperar imágenes de la base de datos *MySQL* se utiliza un procedimiento almacenado que maneja un arreglo de bytes para almacenar o extraer información de un campo de tipos *blob*; se encontró que otras sentencias *SQL*, que no estuvieran almacenadas, no guardan ni recuperan información binaria correctamente del campo *blob* de *MySQL*.

La clase **OperacionesGenericasMySQL.java** tiene métodos invocados por los objetos de tipo ***DAO***, recibiendo objetos de transferencia (con los datos), los tipos de datos (en un arreglo) y una cadena con el procedimiento almacenado a efectuar; se crea el procedimiento almacenado, los datos se extraen de los objetos de transferencia y junto con el tipo de dato indicado, se efectúa la sustitución de parámetros. Después de revisar todos los objetos de transferencia se efectúa el procedimiento almacenado.

Conceptualmente, la parte más difícil del módulo de base de datos, fue crear métodos genéricos para que pudiera manipular cualquier conjunto de objetos de transferencia recibidos. La lógica implantada en los objetos ***DAO*** es relativamente sencilla, pero consumió mucho tiempo, debido a que se tiene que concatenar cadenas para preparar las sentencias ***DML***, o simplemente para asignar o recuperar datos de los atributos de los objetos de transferencia. Este módulo se hubiera podido hacer más rápida y fácilmente si se hubiera hecho con una base orientada a objetos, ya que en este última se guardan directamente los objetos, sin tener que concatenar cadenas y/o manipular los atributos de los objetos.

La última parte construida fue la implantación de la seguridad y de un filtro de peticiones -para permitir o desviar solicitudes a la página de control de la videocámara. Se regresó a la fase de diseño, para cambiar el modelo Entidad-Relación, modificando en *MySQL* las tablas **Usuarios** y **Niveles**, y creando otra tabla dependiente de las anteriores llamada **Usuarios-Niveles**, con el fin de implantar un reino de seguridad con autenticación con base en ***JDBC***; se modificaron los guiones de creación de tablas de la base de datos, y en el guión de inserción de datos se creó un usuario por defecto, el "Super-Usuario" ***SU***, con el cual entrar al sistema por primera vez, y autorizar a los nuevos usuarios.

La tabla **Usuarios** conserva los campos *nombre_usuario* y *contraseña*, pero se le quita el campo *id_nivel*, para que en la tabla *usuarios-niveles* se aprecie cuáles usuarios tienen un nivel, y cuáles no. La tabla **niveles** se cambia, para tener una sola columna con la descripción de los niveles existentes para los usuarios, "SU" de super-usuario, "Administrador" y "Usuario", con permisos más reducidos.

Una vez modificada la base de datos, se configuró ***Apache-Tomcat*** para crear el reino de seguridad con base en ***JDBC***, delegando a ***Tomcat*** el control de acceso a las partes privadas de la aplicación; el contenedor de ***servlets*** también permite indicar cuáles páginas pueden ser mostradas sin pedir usuario y contraseña, y cuales sí requieren que se autentifique al usuario. A diferencia del resto del código de la aplicación web -que pude funcionar en otros servidores de aplicaciones que le den respaldo a la tecnología java- estas configuraciones son específicas para el contendedor ***Apache-Tomcat***, por lo que si se requiere tener control de acceso en un servidor de aplicaciones, se tendría que estudiar la configuración propia de dicho servidor, la cual puede ser igual (para los servidores con base en ***Tomcat***) o similar.

La configuración del reino de seguridad [43, 49] en ***Tomcat*** requiere lo siguiente:

1 El archivo *JAR* de *Connector/J* en *<directorio_tomcat>/lib*, para que el contenedor (y no sólo la aplicación web) lo pueda encontrar.
2 Indicar en *web.xml* los nombres de las tablas de usuarios y papeles.
3 Se creó ***acceso.jsp***, el cual tiene una forma *HTML* con nombre especial...
4 Se creó la página de error ***error.jsp***, la cual es mostrada cuando falla la autenticación del usuario.
5 Se configuró *web.xml* para indicar cuáles recursos web requieren de control de acceso.

Para autorizar a un nuevo usuario o administrador, se verifica las credenciales del usuario en sesión, y si su nivel es mayor que el solicitado para el nuevo usuario, continúa con las validaciones de los demás datos.

Se creó un filtro de peticiones hacia la página de la videocámara, para desviar las peticiones, hacia dicha página cuando se requiera mantenimiento o calibración sin tener que detener toda la aplicación. Se modificó en *web.xml* lo siguiente:

```
<filter>
   <filter-name>RedirigirVideoCamara</filter-name>
   <filter-class>filtro.AuthorizationFilter</filter-class>
</filter>
<filter-mapping>
     <filter-name>RedirigirVideoCamara</filter-name>
     <url-pattern>/controlDireccionPuerto/*</url-pattern>
</filter-mapping>
<filter-mapping>
     <filter-name>RedirigirVideoCamara</filter-name>
     <url-pattern>/jsp/controlDireccion.jsp</url-pattern>
```

</filter-mapping>

La etiqueta *filter* indica se define un filtro de peticiones hacia las *URLs* definidas dentro de *filter-mapping* en las etiquetas *url-pattern*. La clase **AuthorizationFilter** (siempre debe tener dicho nombre) revisa las peticiones, para permitir que se muestre la pantalla del control de dirección, o re-dirigir la petición *HTTP*.

Algunos ***servlets*** y ***JSPs*** utilizan la interfaz *Log4j* para reportar a bitácora; esta interfaz permite reportar distintas acciones en un archivo de forma uniforme, y según la jerarquía de la acción, evento o excepción a reportar, desde nivel de depuración hasta errores severos. En las aplicaciones web es recomendable tener una bitácora, para guardar la información de los hechos relevantes ocurridos (transacciones, errores, inicio y fin de sesiones); sin embargo, para esta aplicación web fue más útil la depuración en tiempo real con *NetBeans*.

Para facilitar la colocación de la aplicación web en ***Tomcat***, se crea un archivo *WAR* (*Web ARchive*), generado en *JDeveloper* o en *NetBeans*, con lo que se evita pasar elemento por elemento (clases, ***JSPs***, archivos *JAR* o *HTMLs*). El archivo *WAR* se genera de forma automática, y aunque existen otras herramientas para generarlo (como *ANT*), no se explicará cómo se crea; basta mencionar que es un archivo comprimido, creado con el mismo algoritmo de compresión que los archivos *ZIP*. Este archivo se copia en el sub-directorio *webapps* de ***Apache-Tomcat***, el cual se encarga de descomprimirlo en el directorio de la aplicación web.

Durante el desarrollo y al hacer pruebas, se definieron y/ modificaron las siguientes variables de ambiente, de las cuales también se muestra su valor correspondiente:

- **JAVA_HOME:** *C:\Java\jdk1-6-0_17*
- **PATH:** *%SystemRoot%\system32;%SystemRoot%;%SystemRoot%\System32\Wbem;%JAVA_HOME%\bin;C:\MySQL\MySQL_Server_5-1\bin*
- **CLASSPATH:** *.;C:\Java\JavaAPI\commapi\comm.jar*

La variable de ambiente **JAVA_HOME** señala dónde está instado el ***JDK*** del lenguaje java. La variable **PATH** indica las rutas a los ejecutables de las aplicaciones, en este caso, del ***JDK*** de java, y del administrador de *MySQL*.

La configuración mínima para ejecutar la aplicación web es la siguiente:

- *Windows 95*, *98*, *Millenium*, o *Windows XP*.
- Computadora con puerto paralelo ***IEEE 1284***.
- *Java 1.5* o superior; los módulos de control de puerto paralelo y de video trabajan con *Java 1.2,* pero el código del módulo de administración y de base de datos incorporan el concepto de *genéricos* [15].
- ***Apache-Tomcat 5.x*** en adelante, en conjunto con un ***Java Runtime Enviroment*** versión 1.5
- Administrador de base de datos *MySQL 3* en adelante.
- Interfaz ***Java Communications 2.0*** para *Windows*, instalada en el ***JRE*** o en el ***JDK*** que use ***Apache-Tomcat***.
- Interfaz ***Java Media Framework 2.1.1.e*** con *Performance Pack* para *Windows*.
- Interfaz *J/Connector 4* en adelante.
- Interfaz *Log4j 1.2* en adelante.

2.3.2 Construcción del circuito de control.

La construcción del circuito de control comenzó una vez que se tenía el diseño del circuito electrónico y se habían conseguido las piezas del mismo; debido a que se alambran componentes físicos, se tuvo que seguir el diagrama del circuito.

Originalmente el circuito empleado para las pruebas de desarrollo era un circuito formado por *buffers* de salidas de tres estados SN74LS241N (para aumentar la señal del puerto paralelo), y circuitos de tipo *latch* SN74LS373N (para retener el valor del bit proveniente del *buffer*), cuya salida se despliega en diodos emisores de luz *(leds*) -con unas resistencias de por medio, para reducir la corriente. Este circuito usaba lógica *TTL* de 5 [V], alimentado por una pila de 6 [V].

Una vez que se probó exitosamente que se podían enviar datos por el puerto paralelo a través de la interfaz web, se comenzó a diseñar un circuito electrónico que pudiera mover motores de paso, para mover la videocámara. Se probó primero un circuito de configuración de medio paso, encontrándose que no era adecuado para el movimiento deseado, por lo que se descartó esta configuración.

Sin embargo, con base en la experiencia anterior, se diseñó otro circuito de paso completo utilizando los mismos componentes, el cual se muestra en la figura 24; al terminar de diseñar el nuevo circuito con la configuración de paso completo, se alambraron los componentes, siguiendo el nuevo diseño. Este circuito permite acoplar la lógica *TTL* (de 5[V]) del puerto paralelo de la computadora personal con la parte de potencia que mueve los motores de paso (de 12 [V] y de mayor corriente). La etapa de potencia de este circuito se alimenta con un eliminador de 12 [V] y 1000 [mA].

2.4 Pruebas de la Aplicación.

Las pruebas de la aplicación se hicieron, primero por módulo, y a lo largo de la fase de construcción; se evitó programar grandes bloques de código y luego probarlos, sino construir elementos más pequeños, y luego probarlos uno por uno. Luego, estos elementos (clases) se probaron junto con el resto del módulo, y más adelante se probó la aplicación al integrar los módulos.

Los módulos del puerto paralelo y los de video se desarrollaron y probaron de forma independiente, como si fueran aplicaciones de consola, enviando datos por el puerto paralelo y obteniendo una imagen de la videocámara, respectivamente, probándolos en el sistema operativo *Windows Millenium*.

Al integrar ambos módulos en web, se requirió de regresar a la fase de diseño, a desarrollar nuevos códigos, y por lo tanto, de efectuar nuevas pruebas en el ambiente de web. Como resultado de las mismas se ajustaron algunos valores usados como constantes, así como otros de configuración, y se hicieron algunos ajustes menores a la aplicación. Estas pruebas se efectuaron usando ***Apache-Tomcat***, también en ***Windows Millenium***.

Debido a problemas con el sistema operativo *Windows Millenium*, se cambió el sistema operativo por *Windows XP* con *Service Pack 1*; después de instalar las aplicaciones y rehacer el ambiente de desarrollo de esta aplicación, se efectuaron pruebas nuevamente, encontrando que el nombre del video variaba ligeramente de *Windows Millenium* a *Windows XP*, por lo se cambió el nombre de la fuente de video a utilizar. Después de dicho cambio se efectuaron pruebas para verificar a través de web que siguiera obteniendo imágenes de la videocámara y siguiera enviando datos por el puerto paralelo, lo cual lo hacía exitosamente hasta ese momento.

El resto de la aplicación, el módulo de administración, el cual integra base de datos y web, se terminó de desarrollar sobre *Windows XP*; conforme se desarrollaba este módulo, también se aplicaban "parches" de seguridad

a *Windows XP*, y más adelante se instalaron el "*Service Pack 2*" y el "*Service Pack 3*" en *Windows XP*, debido a que prometían mejoras de seguridad de utilidad del sistema operativo, pero no mencionaban explícitamente que cambian la forma en que *Windows XP* administra los periféricos.

Estos cambios no se apreciaron de inmediato, debido a que se estaba construyendo código relacionado con base de datos, y además no se tenía todo el tiempo el circuito de control conectado y energizado. Cuando se cambió la versión de ***Tomcat*** 4 a la 5, se probó nuevamente el código de la aplicación, encontrando que variaba el comportamiento del puerto paralelo. Se había creído que dicho comportamiento era causado por datos que se habían quedado en la memoria reservada para el puerto paralelo después de haber hecho uso de la impresora; más adelante se apreció que se debía a la instalación de uno o varios parches para *Windows XP*, y no por datos que se hayan quedado en memoria, ni por el modo del puerto paralelo elegido en la configuración de *BIOS*. Se verificó que en el *BIOS* estuviera seleccionado el modo *ECP* (*Extended Capability Port*) para el puerto paralelo, el mismo modo con que *Windows XP* trabaja al puerto paralelo (aunque lo llama puerto de impresión).

Durante este periodo de pruebas también se estaba diseñando y probando el circuito de control de movimiento, por lo que se volvieron a ejecutar programas de java que se había hecho para el control del puerto paralelo; se revisó que todos los elementos de la interfaz *Java Communications* estuvieran en las rutas correctas y la variable de ambiente **CLASSPATH** incluyera la ruta hacia *comm.jar* . Se encontró que para funcionara correctamente esta interfaz (como cuando se usaba en *Windows Millenium* o *Windows XP* con *Service Pack 1*), se tenía que efectuar lo siguiente:

1 Inhabilitar el puerto paralelo en el Administrador de Dispositivos del Sistema de *Windows XP*.
2 Conectar y energizar el circuito de control.
3 Reiniciar *Windows XP*.
4 Habilitar el puerto paralelo.

Mediante los pasos anteriores se comprobó que se podía utilizar la interfaz *Java Communications* para enviar datos por el puerto paralelo tal y como se había hecho en *Windows 95*, *98* y *Millenium*. También se trató de obtener datos del puerto paralelo, utilizando el circuito de pruebas que usa circuitos tipo *latch*, pero se encontró que esta interfaz lanza la excepción *Unsupported operation. Output only mode* al tratar de cambiar el modo del puerto paralelo de *SPP* a *ECP* o a *EPP*, por lo que tampoco se pudo crear el flujo de entrada, ni recibir datos, a pesar de que la documentación de esta interfaz indica que sí se puede hacer.

Se efectuaron otras pruebas para verificar si la interfaz *Java Communications* respondía mejor en *Windows XP*, tal como habilitar el "*Plug and Play*" heredado en el controlador del puerto paralelo, pero al tratar de usar el puerto la excepción "*PortOwnershipException*" era arrojada, aunque no hubiera otra aplicación usando el puerto paralelo, por lo que no se podía hacer uso del puerto.

Se habilitó la opción "Usar cualquier interrupción asignada al puerto" en "Método de filtrado de recursos" de la configuración de puerto; al cambiar la configuración del puerto paralelo, la interfaz *Java Communications* vuelve a enviar datos correctamente. Si se conserva la configuración del puerto paralelo, y se reinicia el equipo, se tiene que inhabilitar y habilitar el puerto paralelo en el "Administrador de dispositivos", para que *Java Communications* envíe datos correctamente por dicho puerto.

También se trató de registrar la librería dinámica *win32com.dll* en *Windows XP* ejecutando el comando *regsvr32*, ejecutándolo de la siguiente forma:

regsvr32 /n /i: win32com.dll

Sin embargo, el sistema operativo envió el mensaje de advertencia "Se descargó *win32comm.dll*, pero no se encontró el punto de entrada *DllInstall*. No se puede registrar este archivo". Esto se debe a que *Windows XP* no lo reconoce como una librería dinámica válida, o porque *win32com.dll* requiere de otra librería dinámica (que no indica en ninguna parte de la documentación).

Se efectuaron pruebas secuenciales y recurrentes al ***servlet*** que controla la posición y proporciona la página web con la imagen solicitada de la videocámara; estas pruebas se efectuaron solicitando dicho ***servlet***, tal y como lo haría *Internet Explorer* o *Firefox*. Las primeras pruebas fueron las secuenciales, de las cuales se tomó el tiempo de inicio de la petición, el tiempo de fin, calculándose la duración de la misma; los resultados de la misma se muestran sus resultados a continuación:

Pruebas de tiempo de respuesta a solicitudes de URL.	**Datos de las pruebas.**
Línea de comando con los parámetros de ejecución.	C:\Java\jdk1-6-0_14\bin\javaw.exe -client -classpath C:\Proyectos\Java\PruebasAccesosSimultaneos\PeticionesURL\PeticionesSimultaneas\classes peticionessimultaneas.PruebaSolicitudesDistintas
	URL: http://192.168.0.10/videocamara/controlDireccionPuerto?direccionActual=1 Tiempo inicio solicitud: Tue Sep 29 22:45:49 CDT 2009 Tiempo inicio solicitud: Tue Sep 29 22:46:15 CDT 2009 Lapso solicitud: 25782 [ms]
	URL: http://192.168.0.10/videocamara/controlDireccionPuerto?direccionActual=1 Tiempo inicio solicitud: Tue Sep 29 22:46:15 CDT 2009 Tiempo inicio solicitud: Tue Sep 29 22:46:21 CDT 2009 Lapso solicitud: 5516 [ms]
	URL: http://192.168.0.10/videocamara/controlDireccionPuerto?direccionActual=2 Tiempo inicio solicitud: Tue Sep 29 22:46:21 CDT 2009 Tiempo inicio solicitud: Tue Sep 29 22:46:26 CDT 2009 Lapso solicitud: 5734 [ms]
	URL: http://192.168.0.10/videocamara/controlDireccionPuerto?direccionActual=3 Tiempo inicio solicitud: Tue Sep 29 22:46:26 CDT 2009 Tiempo inicio solicitud: Tue Sep 29 22:46:33 CDT 2009 Lapso solicitud: 6547 [ms]
	URL: http://192.168.0.10/videocamara/controlDireccionPuerto?direccionActual=4 Tiempo inicio solicitud: Tue Sep 29 22:46:33 CDT 2009 Tiempo inicio solicitud: Tue Sep 29 22:46:39 CDT 2009 Lapso solicitud: 6359 [ms]

	URL: http://192.168.0.10/videocamara/controlDireccionPuerto?direccionActual=2 Tiempo inicio solicitud: Tue Sep 29 22:46:39 CDT 2009 Tiempo inicio solicitud: Tue Sep 29 22:46:45 CDT 2009 Lapso solicitud: 5625 [ms]
	URL: http://192.168.0.10/videocamara/controlDireccionPuerto?direccionActual=3 Tiempo inicio solicitud: Tue Sep 29 22:46:45 CDT 2009 Tiempo inicio solicitud: Tue Sep 29 22:46:51 CDT 2009 Lapso solicitud: 6250 [ms]
	URL: http://192.168.0.10/videocamara/controlDireccionPuerto?direccionActual=1 Tiempo inicio solicitud: Tue Sep 29 22:46:51 CDT 2009 Tiempo inicio solicitud: Tue Sep 29 22:46:57 CDT 2009 Lapso solicitud: 5625 [ms]
	URL: http://192.168.0.10/videocamara/controlDireccionPuerto?direccionActual=4 Tiempo inicio solicitud: Tue Sep 29 22:46:57 CDT 2009 Tiempo inicio solicitud: Tue Sep 29 22:47:03 CDT 2009 Lapso solicitud: 5844 [ms]
	URL: http://192.168.0.10/videocamara/controlDireccionPuerto?direccionActual=1 Tiempo inicio solicitud: Tue Sep 29 22:47:03 CDT 2009 Tiempo inicio solicitud: Tue Sep 29 22:47:08 CDT 2009 Lapso solicitud: 5328 [ms]
Tiempos inicial y final de las pruebas.	Terminó las pruebas. Tiempo de inicio de las pruebas: Tue Sep 29 22:45:49 CDT 2009 Tiempo de fin de las pruebas: Tue Sep 29 22:47:08 CDT 2009 Tiempo total de pruebas: 78640[ms] Process exited with exit code 0.

Tabla 18: Pruebas secuenciales de la página de la videocámara.

De los datos de las pruebas anteriores se puede apreciar que sólo se inició la prueba subsecuente hasta que hubiera terminado la prueba anterior; en cada prueba incluye el inicio de la petición *HTTP*, el lapso en el cual efectúa el movimiento de los motores de paso, la obtención de la nueva imagen, la generación y recepción de la respuesta *HTTP*. También se puede observar que el tiempo de la primera prueba es aproximadamente 4.5 veces mayor que el tiempo de las demás respuestas; esto se debe a que en la primera prueba se activa la videocámara, se abre el puerto paralelo y se pre-compila por primera vez (desde que arrancó ***Apache-Tomcat***) el ***JSP*** con el cual proporciona la respuesta ***HTTP***. Para las pruebas subsecuentes, la videocámara ya ha sido activada, y sólo se reinicia la toma de video y se detiene; el puerto paralelo ya ha sido abierto y disponible para la aplicación, y respecto al *JSP*, ya se había iniciado su ciclo de vida al pre-compilarse, por lo se vuelve a compilar con mínimos cambios respondiendo mucho más rápido.

Se efectuaron otras pruebas, de forma secuencial, con el fin de verificar que atienda las solicitudes de forma sincronizada; la primera solicitud que llegue deberá ser totalmente satisfecha (cambiar la posición de la videocámara

y obtener una imagen), antes de atender solicitudes subsecuentes, para evitar cambios de posición correspondientes a solicitudes más recientes, cuando apenas está obteniendo la imagen correspondiente a la posición solicitada en la primera solicitud. Se hizo una pequeña aplicación de prueba (también con el lenguaje de programación java), la cual crea varios hilos, y cada uno de ellos efectúa una petición distinta de los otros -cambiando los parámetros que corresponden a la nueva posición solicitada-, los cuales inician prácticamente al mismo tiempo; la prueba termina cuando ya no hay hilos activos (cada hilo cumplió con su ciclo de vida). Los resultados de las pruebas se muestran en la siguiente tabla:

Pruebas de tiempo de respuesta a solicitudes simultáneas de URL.	**Datos de las pruebas.**
Línea de comando con los parámetros de ejecución.	C:\Java\jdk1_6_0-16\bin\javaw.exe -client -classpath D:\Proyectos\Java\PruebasAccesosSimultaneos\PeticionesURL\PeticionesSimultaneas\classes -Dhttp.proxyHost=http://99.90.56.56/ -Dhttp.proxyPort=80 -Dhttp.nonProxyHosts= -Dhttps.proxyHost=http://99.90.56.56/ -Dhttps.proxyPort=80 -Dhttps.nonProxyHosts= peticionessimultaneas.PruebaSolicitudesDistintas
	URL: http://192.168.0.12/videocamara/controlDireccionPuerto?direccionActual=1 Tiempo de inicio prueba número 1: Wed Oct 28 22:05:06 CST 2009 Tiempo de fin prueba número 1: Wed Oct 28 22:05:57 CST 2009 Tiempo duración prueba número 1: 50953 [ms].
	URL: http://192.168.0.12/videocamara/controlDireccionPuerto?direccionActual=1 Tiempo de inicio prueba número 2: Wed Oct 28 22:05:06 CST 2009 Tiempo de fin prueba número 2: Wed Oct 28 22:06:00 CST 2009 Tiempo duración prueba número 2: 53922 [ms].
	URL: http://192.168.0.12/videocamara/controlDireccionPuerto?direccionActual=2 Tiempo de inicio prueba número 3: Wed Oct 28 22:05:06 CST 2009 Tiempo de fin prueba número 3: Wed Oct 28 22:06:03 CST 2009 Tiempo duración prueba número 3: 56578 [ms].
	URL: http://192.168.0.12/videocamara/controlDireccionPuerto?direccionActual=3 Tiempo de inicio prueba número 4: Wed Oct 28 22:05:06 CST 2009 Tiempo de fin prueba número 4: Wed Oct 28 22:06:11 CST 2009 Tiempo duración prueba número 4: 64781 [ms].
	URL: http://192.168.0.12/videocamara/controlDireccionPuerto?direccionActual=4 Tiempo de inicio prueba número 5: Wed Oct 28 22:05:06 CST 2009 Tiempo de fin prueba número 5: Wed Oct 28 22:05:46 CST 2009 Tiempo duración prueba número 5: 40218 [ms].

	URL: http://192.168.0.12/videocamara/controlDireccionPuerto?direccionActual=2 Tiempo de inicio prueba número 6: Wed Oct 28 22:05:06 CST 2009 Tiempo de fin prueba número 6: Wed Oct 28 22:06:08 CST 2009 Tiempo duración prueba número 6: 62218 [ms].
	URL: http://192.168.0.12/videocamara/controlDireccionPuerto?direccionActual=3 Tiempo de inicio prueba número 7: Wed Oct 28 22:05:06 CST 2009 Tiempo de fin prueba número 7: Wed Oct 28 22:06:05 CST 2009 Tiempo duración prueba número 7: 59250 [ms].
	URL: http://192.168.0.12/videocamara/controlDireccionPuerto?direccionActual=1 Tiempo de inicio prueba número 8: Wed Oct 28 22:05:06 CST 2009 Tiempo de fin prueba número 8: Wed Oct 28 22:05:51 CST 2009 Tiempo duración prueba número 8: 45312 [ms].
	URL: http://192.168.0.12/videocamara/controlDireccionPuerto?direccionActual=4 Tiempo de inicio prueba número 9: Wed Oct 28 22:05:06 CST 2009 Tiempo de fin prueba número 9: Wed Oct 28 22:05:54 CST 2009 Tiempo duración prueba número 9: 48390 [ms].
	URL: http://192.168.0.12/videocamara/controlDireccionPuerto?direccionActual=1 Tiempo de inicio prueba número 10: Wed Oct 28 22:05:06 CST 2009 Tiempo de fin prueba número 10: Wed Oct 28 22:05:49 CST 2009 Tiempo duración prueba número 10: 42656 [ms].
Tiempos inicial y final de las pruebas.	Terminó las pruebas. Tiempo de inicio de las pruebas: Wed Oct 28 22:05:06 CST 2009 Tiempo de fin de las pruebas: Wed Oct 28 22:06:11 CST 2009 Tiempo total de pruebas: 65000 [ms] Process exited with exit code 0.

Tabla 19: Resultados de las pruebas de solicitudes simultáneas a la página de la videocámara.

De los resultados de la tabla anterior, se puede observar que el inicio de todas las peticiones fue prácticamente el mismo, pero dichas peticiones no llegaron al servidor en el mismo orden en que iniciaron; al llegar la primera petición, el ***servlet ControlDireccionPuerto.java*** abrió el puerto paralelo e inició la videocámara para obtener una imagen; las demás peticiones tuvieron que esperar a que concluyera y se enviara la respuesta a la petición que llegó primero, para que atendiera la segunda, y así sucesivamente. El tiempo final de cada prueba varía, y la duración de cada prueba no corresponde con el orden en que iniciaron las solicitudes; dichos tiempos son relativamente grandes para el tiempo de respuesta de ***servlets*** o ***JSPs***, pero esto se debe a que la primera petición a **ControlDireccionPuerto.java** consume más tiempo en lo que se inicia el ***servlet*** y abre los periféricos; las otras peticiones incluyen el tiempo de la primera solicitud, más el tiempo de las solicitudes que lo anteceden.

También se puede apreciar que la sincronización del ***servlet*** evita conflictos entre peticiones, a costa de la rapidez de respuesta a las solicitudes hechas.

2.5 Mantenimiento.

El primer mantenimiento del sistema fue cuando se cambió el sistema operativo de *Windows Millenium* a *Windows XP*; el sistema no estaba concluido y faltaba de programar el módulo de base de datos, pero se le debe considerar un mantenimiento ya que se revisó la funcionalidad del sistema, haciendo correcciones menores para que siguiera funcionando el sistema y no programarlo todo de nuevo. En este cambio de sistema operativo, se cambió el nombre con que se identifica a la fuente de video.

Posteriormente a esta migración de sistema operativo, hubo otra fase de desarrollo, relacionada con base de datos. Más adelante al cambiar la versión de ***Tomcat*** 4 a la 5, para aprovechar las ventajas de seguridad y aquellas relacionadas con base de datos de la última, se encontró que ***Apache-Tomcat*** requiere que se registren todos los ***servlets*** de la aplicación web en *web.xml*, de lo contrario ignora las peticiones a los ***servlets*** y no ejecuta el código de los mismos; en este caso se tuvo que registrar el nombre calificado de las clases de los ***servlets*** y registrarlos en *web.xml*, pero no se cambió el código de ningún ***servlet***.

Se cambió el administrador de la base de datos *MySQL* de la versión 4.1 a la 5.1, por lo que se hicieron ajustes menores a los guiones de creación de las tablas de base de datos, eliminando el tipo de tabla a crear, pero se mantuvieron iguales los nombres de tablas, columnas y tipos de datos; la versión 5 de *MySQL* permite configurarlo para indicar el tipo de aplicación que va a usar la base de datos, así como el número de conexiones concurrentes estimadas.

Se ha cambiado varias veces la versión del ***Java Development Kit*** y del ***Java Runtime Enviroment***, por lo que se ha tendido que cambiar los valores de las variables de ambiente, y volver a colocar los elementos de la interfaz ***Java Communications*** en las rutas de las nuevas versiones de java.

El último mantenimiento que se le dio a esta aplicación fue cambiar algunos datos usados como constantes y rutas de archivos, dentro del código de la aplicación, por parámetros de inicio de la aplicación web.