Universidad de las Ciencias Informáticas

“Facultad 15”

Título: “Módulo Administrador de Tareas del Proyecto Sistema de Gestión Fiscal”

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

Autores: Rismary Cabrera Estévez
Leonardo Samada Salazar

Tutor: Ing. Alain Hernández López

Ciudad de La Habana

Junio del 2010

Nunca consideres el estudio como una obligación, sino como una oportunidad para penetrar en el bello y maravilloso mundo del saber.

Albert Einstein

DECLARACIÓN DE AUTORÍA

Declaramos ser los únicos autores de este trabajo y autorizamos a la Universidad de las Ciencias Informáticas a hacer uso exclusivo del mismo en su beneficio. Para que así conste firmamos la presente a los ____ días del mes de _____ del año _____.

_____ _____

Rismary Cabrera Estévez (Autora) Leonardo Samada Salazar (Autor)

_____

Ing. Alain Hernández López (Tutor)

DATOS DE CONTACTO

Ing. Alain Hernández López.

Graduado como Ingeniero en Ciencias Informáticas en el 2007, profesor de la Universidad de las Ciencias Informáticas con categoría docente de Instructor, con tres años de experiencia vinculado a la docencia y a la producción de software en el Proyecto Sistema de Gestión Fiscal perteneciente a la Facultad 15. Universidad de las Ciencias Informáticas, Ciudad de la Habana, Cuba.

Correo electrónico: ahernandezlop@uci.cu

DEDICATORIA

A mi papá, que aunque la vida lo quitó de mi lado, siempre lo llevo en mi mente y mi corazón, y sé que hoy estaría muy orgulloso de mí.

A mis abuelos.

A mi mamá, mi hermana y mi sobrinita,
Por ser las personas más grandes que tengo en este mundo.

Rismary

A Mama
Leo

AGRADECIMIENTOS

Rismary:

***A mi mamá**, por ser incondicional, por enseñarme a superar los golpes de la vida y darme fuerzas para seguir adelante, por confiar en mí, por brindarme su apoyo y amor en cada momento.*

***A mi hermana**, por pensar primero en mí antes que en ella, por hacer hasta lo imposible para que yo salga adelante.*

***A mi sobrinita**, por su dulzura, su amor y su cariño.*

***A mi papá y a mis abuelos**, que aunque ya no los tengo siempre estarán conmigo, por el amor y el cariño que me dieron siempre.*

*A mi segunda familia. **Mis segundos padres Nilda y Francisco**, por quererme como una hija más, por ayudarme en todo momento, por sus consejos y por estar siempre pendientes de mí. **A Maylén**, por ser como una hermana para mí.*

***A mi amiga Yaxe,** por ser tan buena amiga, mi mensajera de correos.*

***A mis tíos (Ramón, Osanita y Leysi) y a primos**, que me han acogido durante estos 5 años.*

***A mi tía Marcia y a Chicha**, por preocuparse siempre por mí.*

***A Yasel,** por estar dispuesto a ayudarme en todo.*

***A Leo,** por ser un compañero de tesis excepcional y muy buen compañero.*

***A nuestro tutor Alain**, por apoyarnos y haber depositado su confianza en nosotros.*

***A Maggie**, por estar siempre dispuesta a ayudarnos.*

***A Dashiel** por su entrega y el apoyo que me dio durante la carrera.*

*A mis amigos de la universidad. **A Noilsa, Aliuska, Juan, Barlia, Alexis David**, por compartir los buenos y malos momentos, por estar siempre que los he necesitado.*
*A **David** por su ayuda en las revisiones de la tesis y porque a pesar de conocernos en tan poco tiempo se queda en él un pedacito de mí.*

A todas esas personas que de una forma u otra, en el transcurso de estos 5 años, han vivido momentos tristes y alegres a mi lado. A los que nos ayudaron en la realización de este trabajo. Gracias a todos por haber contribuido en la formación de la persona que soy hoy.

Leo:

Agradezco a todas las personas que de una forma u otra aportaron su granito de arena para la realización de este trabajo.

*A **mi compañera de tesis Ris** por su paciencia y dedicación para el trabajo en equipo.*

*A **mi tutor Alain** por sus consejos y ayuda.*

*A **mis 5 padres (Mama, Tontón, Mima, Papín y Yuli)** por todo el apoyo y amor que me han dado y la confianza que han tenido en mi.*

*A mis hermanos **(Milton, Miky y Lili)** por ser mi inspiración y aliento en los momentos difíciles.*

*A **mis tíos y primos** por creer en mí.*

*A mi novia **Maggie** por los obstáculos que me ha ayudado a superar y estos lindos años de la carrera a mi lado.*

*A todos mis amigos de Holguín **(Dago, Iván, Lisan, José, Amado, Yonny, Yudith, Betty, Adislen...)** a los de la UCI **(Jorge, Irnel, Pavel, Street, Pichi, Lisandra, Karo, Noi...)** por soportarme y estar siempre ahí cuando los he necesitado.*

RESUMEN

El sector jurídico ha sido beneficiado como parte de la ardua tarea que lleva adelante el gobierno cubano de informatizar todos los sectores de la sociedad en el país. Formando parte de esta iniciativa se encuentra el proyecto Sistema de Gestión Fiscal (*SGF*) que se desarrolla actualmente en la Universidad de las Ciencias Informáticas, Facultad 15, con el objetivo de informatizar los procesos jurídicos, específicamente los de la Fiscalía General de la República, para agilizar el proceso de toma de decisiones, la eficiencia en el trabajo de los fiscales y elevar los niveles de conformidad por parte de los ciudadanos.

Diariamente en las fiscalías se recibe un gran número de casos a procesar, los mismos tienen términos que como dicta la ley no pueden vencerse. Controlar todos estos datos y manipularlos es un proceso muy complicado que requiere de una buena gestión de la información entre las partes involucradas y una gran responsabilidad en la misma. Este proceso se hace aún más complejo al irse acumulando diariamente nuevos casos. Todo esto provoca que sea difícil para los fiscales mantener el control requerido y que estén actualizados en todo momento de los casos que se estén desarrollando. Este trabajo tiene como objetivo fundamental desarrollar un módulo para el proyecto *SGF* que apoye en la solución a este problema, la misma permitirá administrar todas las tareas existentes en el sistema, mantener actualizados a los fiscales de todos los casos en proceso y del tiempo que tienen para realizarlos, así como en caso de existir incumplimiento se le notificará a los fiscales superiores.

Palabras claves: Tarea(s), tarea(s) automática(s), administrador de tareas, notificaciones.

ÍNDICE DE CONTENIDO

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

INTRODUCCIÓN

El progreso de las Tecnologías de la Información y las Comunicaciones (*TIC*) ha llevado consigo un cambio drástico a nivel mundial. Actualmente estas tecnologías están sufriendo un desarrollo vertiginoso que beneficia prácticamente a todas las esferas de nuestra sociedad, convirtiéndose en una herramienta fundamental en el desarrollo económico y social de cualquier nación.

Cuba no está exenta a estos cambios, hace algunos años, se comenzó en el país el proceso de informatización de la sociedad, con la incorporación de tales tecnologías en las diferentes esferas y sectores para el logro de una mejor eficiencia, eficacia y elevar la calidad de vida de los ciudadanos.

Entre los diferentes programas que ha llevado la Revolución Cubana para el desarrollo de esta ardua labor se encuentra la creación de la Universidad de las Ciencias Informáticas (*UCI*) en al año 2002, la cual tiene entre sus misiones la formación de personal capacitado y la producción de software, así como de servicios informáticos, siguiendo la modalidad de vincular el estudio con la producción e investigación.

Como parte de un acuerdo entre la *UCI* y la Fiscalía General de la República (*FGR*) surge la tarea de automatizar los procesos jurídicos del país, específicamente los de la *FGR*, ya que la misma contaba con un sistema que no cumplía las expectativas requeridas.

La *FGR*, según establece la Constitución de la República de Cuba en su artículo 127 capítulo XIII, es el órgano del Estado al que corresponde, como objetivos fundamentales, el control y la preservación de la legalidad, sobre la base de la vigilancia del estricto cumplimiento de la Constitución, las leyes y demás disposiciones legales, por los organismos del Estado, entidades económicas y sociales y por los ciudadanos; y la promoción y el ejercicio de la acción penal pública en representación del Estado.

Sistema de Gestión Fiscal (*SGF*) es el proyecto actual que se está desarrollando en la *UCI* para la *FGR*, específicamente en la Facultad # 15, el mismo es una aplicación web que tiene como objetivo principal informatizar los procesos que se desarrollan en las fiscalías para agilizar la toma de decisiones y elevar el control de los procesos.

La asignación de trabajo en las fiscalías cubanas se realiza de jefe a subordinados. Por lo general cada jefe tiene un grupo numeroso de subordinados y estos a su vez un gran número de casos a procesar, debiendo estar pendiente en todo momento de cada uno de ellos, ya que tienen plazos definidos e inquebrantables para su cumplimiento o entrega. Esto requiere un alto volumen de información a tramitar, generando gran cantidad de documentos que precisan de un considerable empleo de tiempo y esfuerzo, lo que hace más complejo el control que debe llevarse sobre todos los documentos.

Toda esta carga de trabajo trae consigo que existan en las fiscalías retrasos, mala organización del trabajo y descontrol sobre los procesos cotidianos. Para ello se ha pedido que se automatice el trabajo de la administración y control de las tareas que se realizan a diario, las que serían generadas en el *SGF*, permitiendo a los fiscales tener conocimiento de lo que les corresponde hacer en cada momento, notificándoles constantemente sobre el estado de sus tareas y los plazos que tienen para realizarlas.

Por lo planteado anteriormente *SGF* requiere de un módulo que se encargue de administrar las diferentes tareas que se generan en el sistema; además de mantener al tanto a los fiscales sobre todas sus actividades en la aplicación y el estado de sus procesos.

Analizando lo expuesto anteriormente surge el siguiente **problema científico**:
¿Cómo controlar las actividades generadas en el Sistema de Gestión Fiscal para mejorar la calidad con que se ejecutan los procesos en el sistema?

Teniendo en cuenta el problema planteado se define como **objeto de estudio**:
El proceso de desarrollo del software.

Por lo que se especifica el siguiente **campo de acción**:
Sistemas de control, ejecución y administración de tareas.

Dadas estas condiciones se plantea lograr como **objetivo general**:
Desarrollar un módulo que garantice el control, ejecución y administración de las tareas generadas en el Sistema de Gestión Fiscal.

La investigación se basa en la siguiente **hipótesis**:

El desarrollo de un módulo para el control, ejecución y administración de las tareas generadas en el Sistema de Gestión Fiscal permitirá mejorar la calidad con que se ejecutan los procesos en el sistema.

Para dar cumplimiento al objetivo trazado se proponen las siguientes **tareas de la investigación:**

- Elaboración del marco teórico de la investigación.
- Levantamiento de requisitos.
- El diseño del módulo.
- La implementación de un módulo que satisfaga los requerimientos planteados por la Fiscalía General de la República a partir de los resultados obtenidos en el marco teórico.
- La validación del componente implementado.

Se utilizaron los **métodos científicos** siguientes:

Métodos Teóricos:

Entre los métodos teóricos el **Análisis Histórico Lógico** permite estudiar de forma analítica la trayectoria histórica real de los fenómenos, su evolución y desarrollo.

Este se aplica en la elaboración del estado del arte posibilitando realizar un análisis de la evolución de los sistemas y herramientas automáticas utilizadas para la gestión y administración de tareas.

Otro método teórico utilizado es el **Analítico – Sintético** que facilita el entendimiento del fenómeno en el que se trabaja, es más útil la división de este en diferentes fases, y de esta forma descubrir sus características generales, lo que ayuda a seguir una correcta investigación.

Se realizó una investigación y un estudio detallado de documentos y del sistema implementado (*SGF*) para así identificar el problema concreto existente en la *FGR*, sus causas y lograr un profundo análisis y diseño.

Métodos Empíricos:

Entre los métodos empíricos a utilizar se empleó **la Entrevista**.

Se realizaron entrevistas a los fiscales con el objetivo de profundizar sobre las distintas tareas que realizan los mismos. También se entrevistó al jefe de desarrollo del proyecto para conocer las tareas que son generadas por el sistema ya implementado.

El **aporte práctico** esperado del trabajo es:

- Planeación eficiente de las tareas generadas por los usuarios o por el sistema.

- Gestión de las tareas por parte de los usuarios.
- Ejecución automática de las tareas planificadas.
- Apoyo al trabajo diario de los fiscales en las tareas a realizar.
- Ampliación del Proyecto *SGF* con la aparición de este nuevo módulo.

El documento consta de cuatro capítulos:
En el **capítulo I "Fundamentación Teórica"**, se mencionan los principales conceptos de la investigación, se muestra el estado del arte de las diversas técnicas y tendencias del desarrollo de herramientas de administración de tareas existentes. Se realiza además, una descripción de las herramientas y metodologías de desarrollo utilizadas para alcanzar los objetivos propuestos.

En el **capítulo II "Caracterización del Sistema"**, se describe el sistema que se propone para dar solución a la problemática. En primer lugar se detallan los procesos que dan origen a esta situación y luego los procesos que serán automatizados (tareas). Se recogen las características básicas del sistema expresadas en funcionalidades y en requisitos no funcionales como apariencia, rendimiento, confiabilidad, entre otras.

En el **capítulo III "Diseño del Sistema",** se realiza el diseño del módulo. Esto incluye la definición del modelo de diseño describiendo las principales clases del diseño, especificando sus atributos y operaciones, definiendo los patrones y estilos arquitectónicos.

El **capítulo IV "Implementación y Prueba",** está dedicado a la implementación y prueba de la solución propuesta, en el cual se muestra el diagrama de despliegue, los diagramas de componentes y las pruebas de caja negra aplicadas al sistema.

CAPÍTULO I: FUNDAMENTACIÓN TEÓRICA

1.1. Introducción

En este capítulo se definen los principales conceptos relacionados con el dominio del problema. Se analiza el estado del arte de los sistemas existentes para la administración de tareas. Además se realiza un estudio sobre las posibles herramientas y metodologías a utilizar para el desarrollo del módulo, así como la selección de las idóneas para la solución del problema.

1.2. Conceptos asociados al dominio del problema

En el desarrollo del trabajo se utilizan algunos términos técnicos de los cuales a continuación se exponen sus definiciones para un mejor entendimiento y comprensión de los mismos.

1.2.1. Tarea

“Se le denomina tarea al conjunto de actividades específicas en las que es posible distribuir responsabilidades, en aras de una eficiente división del trabajo al interior del equipo responsable de la ejecución del proyecto y aprovechamiento del tiempo”. (1)

1.2.2. Tarea automática

Una tarea automática es una acción, cuya responsabilidad para ejecutarse se le atribuye a un sistema computacional, eléctrico o programa informático, que se debe realizar cada cierto tiempo o en un momento específico y propio de dicha tarea. Sirve para ejecutar automáticamente una acción a cierta hora, cierto día o periódicamente.

En el contexto de aplicaciones web, una tarea automática normalmente es un *script PHP* o *CGI* a ejecutar en un momento definido. Algunos sitios le llaman también "*Scripts* autoejecutables".

Ejemplo de tareas automáticas en *SGF*:

- Seguimiento y control de los procesos fiscales.
- Envío automático de correo.
- Notificaciones automáticas a través del sitio.
- Limpieza automática de la base de datos.
- Ejecución automática de archivos (Ej: script de réplica).

1.2.3. Administrador de tareas o *cron*

Un Administrador de Tareas es un programa informático diseñado para ejecutar procesos definidos a intervalos regulares (por ejemplo, sólo una vez, a cada minuto, día, semana o mes). Los procesos que deben ejecutarse se especifican en una base de datos o fichero. Además proporcionan información sobre el estado de dichos procesos permitiendo la modificación de los mismos.

Un Administrador de Tareas de un sitio web o cron es una aplicación informática que se encarga de la ejecución, gestión y control de los procesos que puede generar el sitio o un usuario del mismo.

"Un *Cron Job* es una tarea (*job*) guardada en un fichero que el *Cron* verifica cada minuto para ver si existe alguna tarea a realizarse, todo esto se efectúa en segundo plano". (2)

1.3. Valoración del estado del arte

1.3.1. Historia de los administradores de tareas

Todo sistema operativo gestiona los programas mediante el concepto de proceso. En un instante dado, en el ordenador pueden existir diversos procesos listos para ser ejecutados. Sin embargo, solamente uno de ellos puede ser ejecutado (en cada microprocesador). De ahí la necesidad de que una parte del sistema operativo gestione, de una manera equitativa, qué proceso debe ejecutarse en cada momento. Es por ello que surgen los primeros administradores de tareas, que con su evolución permitieron administrar otras tareas que no fuesen propias del sistema.

Con el transcurso del tiempo se crean nuevas herramientas para gestionar y/o administrar procesos propios de diversos programas con una independencia parcial de la que brinda el sistema operativo. Aparecieron disímiles *software* para satisfacer requerimientos de otros programas, permitiendo además, que usuarios interactuaran con ellos y ejercieran un rol para gestionar los diferentes procesos. Con la aparición de complejas aplicaciones web para gestionar el negocio concerniente a empresas, surge la necesidad de que existieran sistemas de administración para sus tareas, que garantizaran el correcto funcionamiento de estas y permitieran su monitoreo y gestión.

En la actualidad es frecuente que la responsabilidad de la ejecución de los crones (tareas a ejecutar) de las aplicaciones web sea el *Cron* del sistema operativo, ejecutando los *scripts* que satisfacen las necesidades del sitio web. Algunos *CMS* (Sistema de Administración de Contenido, por sus siglas en inglés) traen unido su propio sistema para estas gestiones, como es el caso de *Joomla* y *Drupal*.

1.3.2. Características de los administradores de tareas

Es conveniente en el desarrollo de una aplicación analizar sus características. Los sistemas de administración automáticos de tareas se caracterizan por:

- Interfaz liviana, de fácil accesibilidad y con algoritmos que permitan su actualización automática y constante, permitiendo mostrar el contenido real en cada momento.
- Se construyen según la necesidad y los requerimientos.
- Deben permanecer corriendo de forma permanente para posibilitar la ejecución de sus trabajos en tiempo y forma.
- Suelen incluir la coordinación de tareas humanas con tareas automáticas.
- Suponen un constante monitoreo del estado de los procesos que tramitan, así como la gestión de los mismos.

1.3.3. Ventajas y desventajas

Dentro de las ventajas de los sistemas para administrar tareas se encuentran:

- Automatización de tareas.
- Sistema de ejecución de procesos.
- Centralización de la información de los procesos.
- Administración humana de tareas maquinales.
- Reducción de costos en cuanto a recursos humanos destinados con anterioridad para estas funciones.
- Garantía de reportes fiables de los procesos gestionados.

Al igual que cualquier programa informático, los administradores de tareas cuentan con desventajas a pesar de las disímiles ventajas que presentan, entre ellas se encuentran:

- En caso de falta de energía eléctrica pueden quedar procesos sin ejecutarse en el momento señalado.
- Un excesivo número de procedimientos a ejecutarse automáticamente puede colgar el ordenador, ya que su trabajo se basa en hilos del sistema.

1.3.4. Análisis de sistemas existentes.

A continuación se enumeran algunos de los sistemas existentes en el mundo para la administración automática de tareas y gestión de crones:

1.3.4.1. *Quartz*

Este es un *framework* (herramienta) de *scheduler* (planificador) de nivel empresarial, en base *Java J2EE*, que posee dentro de sus características principales, niveles de administración de clúster, y tolerancia a fallas, manejo transaccional y lo mejor de todo es una solución *Open Source* (código abierto, por sus siglas en inglés), basada en licencia de tipo *Apache 2.0*. *Quartz* opera embebido en una aplicación *Java*, en un servidor de aplicaciones y de otras formas similares, calendarizando, ejecutando y notificando los procesos que se implementen con el mismo.

1.3.4.2. *QueIt*

Es una poderosa utilidad de planificación que permite planificar programas, tareas o incluso ficheros *batch MS-DOS* para cargar automáticamente a las horas específicas que se elijan. Con *QueIt* se puede planificar un programa para cargar diariamente, semanalmente, mensualmente o a cada hora. Admite incluso ser configurado para que ejecute una tarea o programa cada minuto. *QueIt* también soporta ejecuciones condicionales (por ejemplo, sólo ejecutar un programa si un fichero existe). Permite enviar también argumentos mediante líneas de comandos al programa que esté lanzando. Incluso consigue enviar un correo electrónico después que una tarea ha comenzado o se ha completado. Es capaz de ejecutar sus tareas planificadas en ordenadores remotos (por ejemplo en un ordenador diferente al que se instaló *QueIt*).

1.3.4.3. *Cron Jobs* del *CPanel*

CPanel (acrónimo de Panel de Control) es una herramienta de administración basada en tecnologías web para administrar sitios que incluye el módulo *Cron Jobs*. Los trabajos de *cron* permiten automatizar ciertos comandos y *scripts* en un sitio y básicamente ejecutan tareas programadas por el usuario. Puede configurar un comando o *script* para correr a un tiempo específico cada día, semana, etc. Por ejemplo, consigue poner un trabajo de *cron* que borre archivos temporales para que el espacio de disco no sea usado por esos archivos. Es una herramienta no libre y actualmente solo está disponible para sistemas operativos basados en *GNU - Linux*.

Los *Cron Jobs* pueden utilizarse para automatizar casi cualquier elemento en una cuenta de *hosting* (alojamiento web), desde realizar respaldos de una base de datos en *MySQL* (Servidor de Base de Datos, por sus siglas en inglés) o ejecutar un *script* en el servidor.

1.4. Diferencias de los sistemas estudiados

Después de haber realizado un estudio de sistemas de administración de tareas con características similares al que se necesita, se llegó a la conclusión de manera general que aunque sirven de apoyo para la solución propuesta no pueden ser utilizados ya que:

- En el caso de *Quartz*, requiere la máquina virtual de java para poder funcionar, además las tareas que es capaz de planificar se reducen a notificaciones y ejecución de algunos scripts.
- Por su parte la herramienta *QueIt* no permite una integración directa con ninguna base de datos, imposibilitando el manejo de la información almacenada en ellas y comunicación con la aplicación web, hasta el momento *QueIt* solo es funcional para plataformas *Windows*.
- *CronJobs* es de las tres la más similar a la que se propone, sin embargo actualmente solo existen versiones disponibles para *Linux* y no permite su extensión ya que no es patentizado bajo licencia *Open Source*. Es una herramienta no libre suponiendo esto un gasto adicional para el proyecto.

En resumen todas tienen la desventaja de depender completamente del *Cron* (para Linux) o el *Sheduler* (para *Windows*) lo que traería consigo una dependencia total de un servicio del sistema operativo atentando contra la portabilidad del sistema requerido. Además se requiere de un módulo capaz de administrar tareas muy propias de *SGF* y estos sistemas sólo proporcionarían pequeñas funcionalidades dentro del proyecto.

1.5. Lenguajes, herramientas y tecnologías

1.5.1. Lenguajes de programación

“Un lenguaje de programación es un conjunto de símbolos y reglas sintácticas y semánticas que definen su estructura y el significado de sus elementos y expresiones. Es utilizado para controlar el comportamiento físico y lógico de una máquina”. (3)

Es un conjunto de símbolos, caracteres y reglas (programas) que les permiten a las personas comunicarse con la computadora. Los lenguajes de programación tienen un conjunto de instrucciones que permiten realizar operaciones de entrada/salida, cálculo, manipulación de textos, lógica/comparación y almacenamiento/recuperación.

1.5.1.1. Lenguaje *HTML*

HTML, siglas de *Hyper Text Markup Language* (Lenguaje de Marcas de Hipertexto, por sus siglas en inglés), es el lenguaje de marcado predominante para la construcción de páginas web. Es usado para describir la estructura y el contenido de las páginas en forma de texto, así como para complementar el texto con objetos tales como imágenes y tablas. *HTML* se escribe en forma de "etiquetas", rodeadas por corchetes angulares (<,>). También puede describir, hasta cierto punto, la apariencia de un documento, y puede incluir un *script* (por ejemplo *Javascript*), el cual puede afectar el comportamiento de navegadores web y otros procesadores de *HTML*.

1.5.1.2. Lenguaje *JavaScript*

JavaScript es un lenguaje de programación utilizado para crear pequeños programas, encargados de realizar acciones dentro del ámbito de una página web. Se trata de un lenguaje de programación del lado del cliente, porque es el navegador el que soporta la carga de procesamiento. Su uso se basa, fundamentalmente, en la creación de efectos especiales en las páginas y la definición de interactividades con el usuario. Es un lenguaje muy eficaz para operaciones rápidas y con requerimientos gráficos que no dependan totalmente del servidor. Las sentencias escritas en *JavaScript* se encapsulan entre las etiquetas *<script>* y *</script>.*

"Javascript es el siguiente paso, después del *XHTML*, que puede dar un programador de la web que decida mejorar sus páginas y la potencia de sus proyectos. Es un lenguaje de programación bastante sencillo y pensado para realizar sistemas con rapidez". (4)

1.5.1.3. Lenguaje *PHP* v5.x

PHP (*Hypertext Pre-processor*) es un lenguaje de programación libre para la creación de páginas web dinámicas. Permite la creación de aplicaciones con interfaz gráfica, conexión a servidores de base de datos (*Oracle, MySQL, PostgreSQL*) y puede ser ejecutado en sistemas *Unix, Windows, Linux y Mac OSX*.

PHP es un lenguaje de alto nivel. Es procedente del nombre *PHP Tools o Personal Home Page Tools,* sirve principalmente para proporcionar características dinámicas a una página web. Entre sus características más importantes se encuentra la capacidad de poder fusionarse con el lenguaje *HTML*. Se ejecuta e interpreta directamente en el servidor en el que está el sitio web.

La versión 5.x cuenta con innumerables mejoras que consolidan su éxito, como por ejemplo la implementación de un sistema de manejo de excepciones que versiones anteriores no poseían. Ofrece la posibilidad de hacer programas orientados a objetos, lectura de archivos *XML* de forma sencilla, utilización de la base de datos ligera *SQLite* o la implementación de servicios web, además de las características generales de este importante lenguaje de programación que a continuación se presentan:

- Es un lenguaje multiplataforma.
- Completamente orientado a la web.
- Capacidad de conexión con la mayoría de los motores de base de datos que se utilizan en la actualidad, destaca su conectividad con *MySQL y PostgreSQL*.
- Integración con diversas bibliotecas externas, permite generar documentos en *PDF* hasta analizar código *XML*.
- Facilidad de actualización.
- Soportado por una gran comunidad de desarrolladores, permitiendo que los fallos de funcionamiento se encuentren y reparen rápidamente.
- El código se pone al día continuamente con mejoras y extensiones de lenguaje para ampliar las capacidades de *PHP*.
- Facilidad de aprendizaje.

Entre las ventajas de *PHP* se pueden destacar que este lenguaje es completamente expandible, permite las técnicas de Programación Orientada a Objetos y biblioteca nativa de funciones sumamente amplia e incluida.

1.5.1.4. Lenguaje *Python* 2.5

Python es un lenguaje interpretado, orientado a objetos de propósito general. Permite mantener de forma sencilla interacción con el sistema operativo, y resulta muy adecuado para manipular archivos de texto. En la actualidad se desarrolla como un proyecto de código abierto, administrado por la *Python Software Foundation*. El principal objetivo que persigue este lenguaje es la facilidad, tanto de lectura, como de diseño.

"*Python* permite dividir el programa en módulos reutilizables desde otros programas escritos en *Python*. Viene con una gran colección de módulos estándar que se pueden utilizar como base de los programas (o como ejemplos para empezar a aprender *Python*). También hay módulos incluidos que

proporcionan entrada y salida de ficheros, llamadas al sistema, *sockets* y hasta interfaces a *GUI* (Interfaz Gráfica de Usuario, según sus siglas en inglés) como *Tk, GTK, Qt* entre otros". (5)

Ventajas de Python:

- *Python* se utiliza como lenguaje de programación interpretado, lo que ahorra un tiempo considerable en el desarrollo del programa, pues no es necesario compilar ni enlazar. El intérprete se puede utilizar de modo interactivo, lo que facilita experimentar con características del lenguaje, escribir programas desechables o probar funciones durante el desarrollo del programa.
- Es un lenguaje multiplataforma, pues el mismo código funciona en cualquier arquitectura, la única condición es que disponga del intérprete del lenguaje. No es necesario compilar el código una vez para cada arquitectura.
- Es muy útil en el desarrollo de aplicaciones que trabajen con hilos del sistema o demonios.

1.5.1.5. Lenguaje *C++*

Es un lenguaje de programación diseñado a mediados de los años 1980 por *Bjarne Stroustrup*. La intención de su creación fue el extender al exitoso lenguaje de programación *C* con mecanismos que permitieran la manipulación de objetos. En ese sentido, desde el punto de vista de los lenguajes orientados a objetos, el *C++* es un lenguaje híbrido.

Posteriormente se añadieron facilidades de programación genérica, que se sumó a los otros dos paradigmas que ya estaban admitidos como la programación estructurada y la programación orientada a objetos. Por esto se suele decir que el *C++* es un lenguaje multiparadigma. Una particularidad del *C++* es la posibilidad de redefinir los operadores o lo que se conoce como sobrecarga de operadores, y de poder crear nuevos tipos que se comporten como tipos fundamentales. Permite trabajar tanto a alto como a bajo nivel.

1.5.2. Lenguaje de modelado

1.5.2.1. Lenguaje Unificado de Modelado (*UML*)

Lenguaje Unificado de Modelado (*UML*, por sus siglas en inglés, *Unified Modeling Language*) es el lenguaje de modelado de sistemas de software más conocido y utilizado en la actualidad. Es un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema. El *UML* está

compuesto por diversos elementos gráficos que se combinan para conformar diagramas. Debido a que es un lenguaje, cuenta con reglas para combinar tales elementos.

"UML ofrece un estándar para describir un modelo del sistema, incluyendo aspectos conceptuales tales como procesos de negocio y funciones del sistema, y aspectos concretos como expresiones de lenguajes de programación, esquemas de bases de datos y componentes reutilizables". (6)

UML no es una guía para realizar el análisis y diseño orientado a objetos, es decir, no es un proceso. *UML* es un lenguaje que permite la modelación de sistemas con tecnología orientada a objetos que describe lo que supuestamente hará un sistema, pero no dice cómo implementar dicho sistema.

1.5.3. Herramientas y tecnologías

1.5.3.1. *Visual Paradigm* 6.3

"*Visual Paradigm* para *UML* es una de las herramientas *UML CASE* más completa y fácil de usar, con soporte multiplataforma y que proporciona excelentes facilidades de interoperabilidad con otras aplicaciones. Fue creada para el ciclo vital completo del desarrollo del software que lo automatiza y acelera, permitiendo la captura de requisitos, análisis, diseño e implementación. *Visual Paradigm-UML* también proporciona características tales como generación de código, ingeniería inversa y generación de informes. Tiene la capacidad de crear el esquema de clases a partir de una base de datos y crear la definición de base de datos a partir del esquema de clases. Está diseñada para usuarios interesados en sistemas de software de gran escala con el uso del acercamiento orientado a objeto, además apoya los estándares más recientes de las notaciones de *Java* y de *UML*. Incorpora el soporte para trabajo en equipo, que permite que varios desarrolladores trabajen a la vez en el mismo diagrama y vean en tiempo real los cambios hechos por sus compañeros". (7)

1.5.3.2. Eclipse *PDT*

Eclipse es una plataforma de programación utilizada para crear entornos integrados de desarrollo (del inglés *IDE*). Es como una armazón sobre la que se pueden montar herramientas de desarrollo para cualquier lenguaje, mediante la implementación de los *plugins* adecuados.

"La arquitectura de *plugins* de *Eclipse* permite, además de integrar diversos lenguajes sobre un mismo *IDE*, introducir otras aplicaciones accesorias que pueden resultar útiles durante el proceso de desarrollo como: herramientas *UML*, editores visuales de interfaces y ayuda en línea para librerías.

Eclipse dispone de un editor de texto con resaltado de sintaxis. La compilación es en tiempo real. Tiene pruebas unitarias con *JUnit*, control de versiones con *CVS*, integración con *Ant*, asistentes (wizards) para creación de proyectos, clases, tests y refactorización". (8)

1.5.3.3. *Ajax*

"Ajax, acrónimo de *Asynchronous JavaScript And XML* (*JavaScript* asíncrono y *XML*, por sus siglas en inglés), es una técnica de desarrollo web para crear aplicaciones interactivas o *RIA* (Aplicación de Internet Exquisita). No es una tecnología en sí misma. En realidad, se trata de la unión de varias tecnologías que se desarrollan de forma autónoma y que se unen de formas nuevas y sorprendentes". (9)

Se dice que *Ajax* es una tecnología asíncrona, ya que los datos adicionales se requieren al servidor y se cargan en segundo plano sin interferir con la visualización ni el comportamiento de la página. *JavaScript* es el lenguaje interpretado *(scripting language)* en el que normalmente se efectúan las funciones de llamada de *Ajax* mientras que el acceso a los datos se realiza mediante *XMLHttpRequest*, objeto disponible en los navegadores actuales. En cualquier caso, no es necesario que el contenido asíncrono esté formateado en *XML*.

Ajax es una técnica válida para múltiples plataformas y utilizable en muchos sistemas operativos y navegadores, dado que está basado en estándares abiertos como *JavaScript* y *Document Object Model* (*DOM*).

Entre las librerías que incluye *Ajax* se puede encontrar:

- **Librería *Prototype***

"Prototype es una librería de *JavaScript* muy completa que amplía las posibilidades del lenguaje de programación, añade todas esas funciones que faltaban y con las que los programadores soñaban y ofrece nuevos mecanismos para la manipulación de los elementos *DOM*". (10)
Los archivos de *Prototype* se incluyen con el framework *Symfony* y son accesibles en cualquier nuevo proyecto, en la carpeta *web/sf/prototype/.*

"Como los mucho de los *helpers* de *Ajax* de *Symfony* dependen de *Prototype*, la librería *Prototype* se incluye automáticamente cuando se utiliza cualquiera de ellos. Por tanto, no es necesario añadir los

archivos *JavaScript* de *Prototype* a la respuesta si la plantilla hace uso de cualquier *helper* cuyo nombre acaba en _remote. Una vez que la librería *Prototype* se ha cargado, se pueden utilizar todas las funciones nuevas que añade al lenguaje *JavaScript*". (11)

1.5.3.4. *Python Interpreter* 2.5

"El intérprete de *Python* estándar incluye un modo interactivo, en el cual se escriben las instrucciones en una especie de *shell*: las expresiones pueden ser introducidas una a una, pudiendo verse el resultado de su evaluación inmediatamente. Esto resulta útil tanto para las personas que se están familiarizando con el lenguaje como también para los programadores más avanzados: se pueden probar porciones de código en el modo interactivo antes de integrarlo como parte de un programa". (12)

1.5.3.5. *PyScripter* 2.3

PyScripter es un *IDE* (Entorno de Desarrollo Integrado, por sus siglas en inglés) de *Python.* Es una herramienta libre para el diseño de aplicaciones en el lenguaje *Python*. Tiene la ventaja de ser bastante ágil con respecto a otros *IDES* de *Python* y proporciona una amplia mezcla de características que lo convierten en un entorno de desarrollo productivo:

- Alto grado de completamiento de código.
- Ayuda sensitiva del contexto de las palabras claves de *Python*.
- Plantillas de parámetros de código.
- Notificación de cambio de archivo.
- Conversión de las roturas de línea (*Windows*, *UNIX*).
- Ventana de relojes.
- Ventana de puntos de ruptura.
- Editor de ventanas.
- Documentación en formato *HTML* (|*pydoc*|).
- Explorador de código.
- Explorador de archivo de fácil configuración.
- Acceso a los manuales de *Python* por el menú Ayuda.

1.5.3.6. *Py2exe*

Creado por *Thomas Heller*. *Py2exe* es una extensión que convierte los scripts de *Python* en programas ejecutables para *Windows* incluyendo en su interior las librerías necesarias para la ejecución del

distribuible, de manera que puedan ejecutarse sin necesidad de instalar el intérprete de *Python*. Ofrece la ventaja de la distribución al usuario final sólo del ejecutable de la aplicación desarrollada, siendo transparente para el lenguaje en el cual fue desarrollado, librerías y técnicas utilizadas para su desarrollo.

1.5.3.7. *PyQt GPL* v4.4.3

"*Qt* es una biblioteca multiplataforma para desarrollar interfaces gráficas de usuario y también para el desarrollo de programas sin interfaz gráfica como herramientas de la consola y servidores". (13)

"PyQt es un agregado de la biblioteca gráfica *Qt* para el lenguaje de programación *Python*. La biblioteca está desarrollada por la firma británica *Riverbank Computing* y está disponible para *Windows, GNU/Linux y Mac OS X* bajo diferentes licencias". (14)

PyQt GPL v4.4.3 es una versión estable de una biblioteca de desarrollo multiplataforma que está patentizada bajo licencia *GPL* y contiene todas las librerías y utilidades de *QT*, entre sus diferentes ventajas se encuentran:

- Completo conjunto de elementos gráficos (listados, árboles, grillas.)
- Flexible y potente control del comportamiento de la interface.
- Posee un mecanismo de conexión de señales y eventos simple.
- Se puede definir los eventos más sencillos en diseñador de *GUI's* (ejemplo: al pulsar un botón, borrar un campo de texto) y en el código *Python*, definir las acciones más avanzadas.
- Rápido y de apariencia nativa (las últimas versiones utilizan funciones nativas en windows).
- Se puede separar el diseño de la interface, pero usa un "compilador" *pyuic* para crear las clases *python*.
- Arquitectura opcional para Modelo/Vista para las tablas, listas y árboles.

Se le atribuye la desventaja de no venir pre-instalado con Python, es decir debe ser instalado por separados, además de ser relativamente más complejo de aprender.

1.6. Servidores

1.6.1. *Apache Web Server* 2.2.4

Es un servidor de aplicaciones web, entre sus características se destacan:

- Multiplataforma.

- Es un servidor de web conforme al protocolo *HTTP*/1.1.
- Modular: Puede ser adaptado a diferentes entornos y necesidades, con los diferentes módulos de apoyo que proporciona, y con la *API* de programación de módulos, para el desarrollo de módulos específicos.
- Basado en hebras en la versión 2.0.
- Incentiva la realimentación de los usuarios, obteniendo nuevas ideas, informes de fallos y parches para la solución de los mismos.
- Se desarrolla de forma abierta.
- Extensible: gracias a ser modular se han desarrollado diversas extensiones entre las que destaca *PHP*, un lenguaje de programación del lado del servidor.
- La licencia *Apache* es una descendiente de la licencias *BSD*, no es *GPL*.

1.6.2. *PostgreSQL* 8.4

PostgreSQL es un sistema de gestión de base de datos relacional orientada a objetos y libre, publicado bajo la licencia *BSD*.

Algunas de sus principales características son:

- Alta concurrencia.
- Amplia variedad de tipos nativos.
- Consola de gestión de la base de datos *Windows*.
- Acceso *ODBC* desde cualquier aplicación *Windows* o compatible con el protocolo *ODBC*, por ejemplo, *Microsoft Access*.
- Acceso *JDBC* para conectar desde aplicaciones *Java*, *applets*, etc.
- Crea una amplia funcionalidad a través de su sistema de activación de disparadores (*triggers*).
- Vistas.
- Integridad transaccional.
- Herencia de tablas.
- Tipos de datos y operaciones geométricas.
- Soporte para transacciones distribuidas.

1.7. *Framework*

1.7.1. *Symfony Framework* 1.3

Symfony es un completo *framework* diseñado para optimizar el desarrollo de las aplicaciones web mediante algunas de sus principales características. Entre sus principales características se encuentra que separa la lógica de negocio, la lógica de servidor y la presentación de la aplicación web. Proporciona además varias herramientas y clases encaminadas a reducir el tiempo de desarrollo de una aplicación web compleja. Automatiza las tareas más comunes, permitiendo al desarrollador dedicarse por completo a los aspectos específicos de cada aplicación.

Symfony está desarrollado completamente con *PHP* 5. Es compatible con la mayoría de gestores de bases de datos, como *MySQL, PostgreSQL, Oracle y Microsoft SQL Server*. Se puede ejecutar tanto en plataformas **nix* (*Unix, Linux*, etc.) como en plataformas *Windows*.

"*Symfony* se diseñó para que se ajustara a los siguientes requisitos:

- Fácil de instalar y configurar en la mayoría de plataformas (y con la garantía de que funciona correctamente en los sistemas *Windows* y **nix* estándares).
- Independiente del sistema gestor de bases de datos. Su capa de abstracción y el uso de *Propel*, permiten cambiar con facilidad de *SGBD* en cualquier fase del proyecto.
- Utiliza programación orientada a objetos, de ahí que sea imprescindible *PHP* 5.
- Sencillo de usar en la mayoría de casos, está recomendado para grandes aplicaciones web.
- Aunque utiliza *MVC* (Modelo Vista Controlador), tiene su propia forma de trabajo en este punto, con variantes del *MVC* clásico como la capa de abstracción de base de datos, el controlador frontal y las acciones.
- Basado en la premisa de "convenir en vez de configurar", en la que el desarrollador solo debe configurar aquello que no es convencional.
- Sigue la mayoría de mejores prácticas y patrones de diseño para la web.
- Preparado para aplicaciones empresariales, y adaptable a las políticas y arquitecturas propias de cada empresa, además de ser lo suficientemente estable como para desarrollar aplicaciones a largo plazo.
- Fácil de extender, lo que permite su integración con las bibliotecas de otros fabricantes.
- Tiene una potente línea de comandos que facilitan generación de código ahorrando tiempo de trabajo". (11)

1.8. Metodologías

El desarrollo de software es sin dudas una tarea difícil y más aún si se trata de un software complejo. Es por ello que surgen las llamadas Metodologías de Desarrollo de *Software*, dada la necesidad de crear métodos que permitieran guiar, controlar y documentar el desarrollo de los mismos.

"La Metodología, (del griego *metà* "más allá", *odòs* "camino" y *logos* "estudio"), hace referencia al conjunto de procedimientos basados en principios lógicos, utilizados para alcanzar una gama de objetivos que rigen en una investigación científica o en una exposición doctrinal". (15)

Hoy en día existen numerosas propuestas metodológicas que inciden en distintas dimensiones del proceso de desarrollo. Al usar una metodología de software acorde con el proyecto que se va a desarrollar se obtienen las siguientes ventajas:

- Proporciona una guía para ordenar las actividades de un equipo.
- Dirige las tareas de cada desarrollador por separado y del equipo como un todo.
- Especifica los artefactos que deben desarrollarse.
- Ofrece criterios para el control y la medición de los productos y actividades del proyecto.

1.8.1. Proceso Unificado de Modelado (*RUP*)

El Proceso Unificado es un proceso de desarrollo de *software*, es decir el conjunto de actividades necesarias para transformar los requisitos de un usuario en un *software*. Sin embargo, el Proceso Unificado es más que un simple proceso; es un marco de trabajo genérico que puede especializarse para una gran variedad de sistemas de *software*, para diferentes áreas de aplicación, diferentes tipos de organización, diferentes niveles de aptitud y diferentes tamaños de proyecto. El Proceso Unificado está basado en componentes, lo cual quiere decir que el sistema de software en construcción está formado por componentes *software* interconectados a través de interfaces bien definidas.

RUP está basado en una notación gráfica, la cual permite especificar, construir, visualizar y documentar los artefactos de un sistema de *software* orientado a objetos. Sus principales características son:

- Guiado por casos de uso: Los casos de uso son el instrumento para describir el comportamiento del software y extraer los casos de prueba con los que se valida el sistema.
- Centrado en la arquitectura: Los modelos son proyecciones del análisis y el diseño, describe la arquitectura del producto a desarrollar.

- Iterativo e incremental: Durante todo el proceso de desarrollo se producen versiones superiores.

Se divide en ciclos de trabajo, teniendo un producto superior como resultado de cada ciclo. Estos se componen en su interior por varias fases, en la cuales se llevan a cabo un conjunto de flujos para el desarrollo de todo el proyecto.

"Una particularidad de esta metodología es que, en cada ciclo de iteración, se hace exigente el uso de artefactos, siendo por este motivo, una de las metodologías más importantes para alcanzar un grado de certificación en el desarrollo del software". (16)

1.9. Fundamentación de las herramientas seleccionadas

Luego de haberse realizado un estudio de las principales características de las tecnologías actuales y herramientas existentes, se propone utilizar:

- *IDE* para el desarrollo web *Eclipse*
- Desarrollo del lado servidor el lenguaje *PHP* v5 combinado con *HTML.*
- Lenguaje del lado cliente *JavaScript.*
- Framework *Symfony* en su versión 1.3.
- Servidor web *Apache Web Server* 2.2.4.
- Servidor de base de datos *PostgreSQL* 8.4.
- *PgAdmin* III para la gestión de la base de datos
- Metodología de desarrollo *RUP.*
- Herramienta de modelado *Visual Paradigm* 6.3.

Las propuestas hechas con anterioridad se basan en las características de las mismas que corresponden con las exigencias del proyecto.

Por su parte la aplicación encargada de verificar y ejecutar las tareas automáticas será desarrollada en:

- Lenguaje *Python* por ser rápido para la implementación y excelente para el trabajo con hilos del sistema, además de ser multiplataforma.

- *IDE* de desarrollo *PyScripter* 2.3 por sus amplias ventajas en cuanto a completamiento de código y rapidez para el desarrollo de aplicaciones pequeñas.
- Para la construcción de la interfaz de usuario en el lado servidor se propone la utilización de *PyQT GPL* v4.4.3 ya que incluye todas las librerías de *QT* para interfaces universales y estar patentizado bajo licencia *GPL*.
- Para la distribución de la solución desarrollada se utilizará la herramienta *Py2exe* por brindar la posibilidad de distribuir ejecutables en varias plataformas, dando la posibilidad de situar esta herramienta sobre cualquier sistema operativo, incluso con diferencias de plataformas entre el servidor web y la aplicación.

1.10. Conclusiones del capítulo

Se concluye como resultado de la investigación realizada durante este capítulo que ninguno de los sistemas estudiados, relacionados con la administración automática de tareas, responden a las características del sistema requerido. Por lo que se hace necesario desarrollar un nuevo sistema que dé solución al problema planteado mediante la utilización de las herramientas de desarrollo seleccionadas.

CAPÍTULO II: CARACTERIZACIÓN DEL SISTEMA

2.1. Introducción

En este capítulo se hace la descripción de la propuesta de solución al problema actual de administración de tareas en el Sistema de Gestión Fiscal. Se realiza un estudio detallado de los procesos que dan origen a la situación y los procesos que serán automatizados. Además, se enumeran los requisitos funcionales y no funcionales que debe cumplir el sistema que se propone y se identifican mediante un Diagrama de Casos de Uso del Sistema, las relaciones de los actores con las funcionalidades que brinda el módulo.

2.2. Descripción del objeto de estudio

2.2.1. Situación problemática

A partir de la década de los años 90 se produce en el país un incremento de las manifestaciones de corrupción, condicionadas por la apertura y transformaciones económicas que el Estado Cubano se vio obligado a adoptar. El aumento de los hechos ilícitos hace que exista un incremento de la aplicación de medidas contra aquellos que incumplan con lo establecido por las leyes, trayendo consigo que se tenga que procesar un alto volumen de información y se utilice gran cantidad de tiempo para generar los documentos pertinentes.

Además de lo descrito anteriormente, el déficit de fiscales; que trae consigo que cada uno atienda decenas de casos a la vez y que esté pendiente a su vez de los mismos y de los plazos de vencimiento establecidos por la ley, unido a la mayor complejidad de los procesos tramitados, provoca menor control y supervisión en los procesos tanto de la persona que esté llevando un caso como de los niveles superiores. Los niveles de la estructura en la Fiscalía parten de cada municipio del país hasta llegar al nivel superior en la *FGR*.

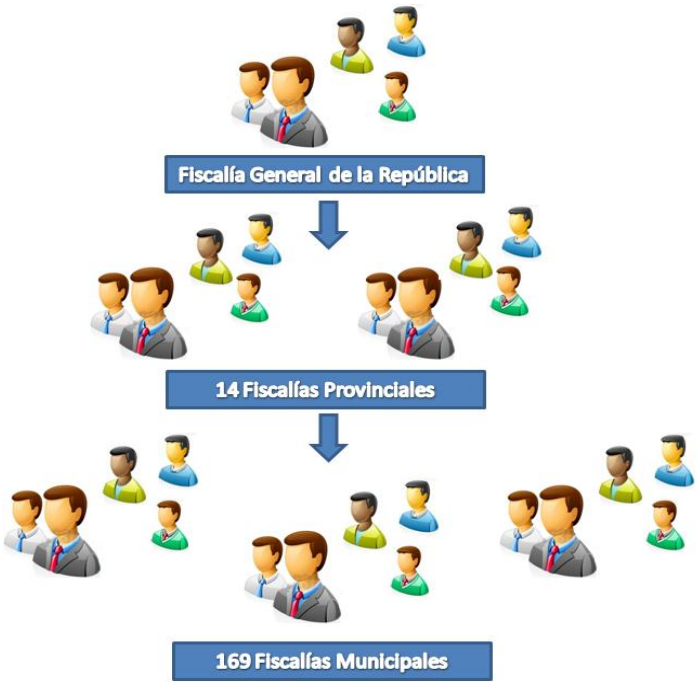


Figura 1. Estructura por niveles de la *FGR*.

El control de las tareas debe realizarse actualmente de la siguiente manera:

- Autocontrol: El Fiscal que está llevando un caso debe ser capaz de mantener un control del mismo, estar bien actualizado en todo momento con los límites de tiempo que dispone para terminar el proceso.
- Seguimiento: Luego de la asignación de un caso por un Fiscal Superior a alguno de sus subordinados el mismo debe dar seguimiento a dicho caso, mantenerse actualizado en todo lo referente a la situación procesal del mismo.

Actualmente estos procesos se realizan de forma manual y a conciencia de cada cual. Con el uso de la aplicación web (*SGF*) los fiscales manejarán todos sus procesos y actividades mediante el sistema según la esfera de trabajo a la cual pertenece. Luego de darle alta en el sistema al expediente el Fiscal apunta en su agenda los tiempos que dispone para desarrollar el proceso y debe consultarla con frecuencia para obtener dicha información. Existe el caso en que esta responsabilidad es asignada a la Secretaria la cual debe estar pendiente e informarle verbalmente a su Jefe Inmediato cuanto tiempo le queda para culminar determinada tarea y en caso de acercarse la fecha de vencimiento debe mantenerlo alerta.

En caso de que un Fiscal Jefe necesite obtener este tipo de información debe consultar de forma personal con la Secretaria o con el Fiscal correspondiente, lo que puede traer consigo que en un momento determinado dichas personas no estén disponibles para ofrecer la información necesaria.

Todo este proceso protocolar, unido a la carga de trabajo, repercute en el control, la organización, los atrasos en los procesos, la demora al consultar los estados de los procesos que se estén llevando a cabo, afectando además la toma rápida y eficiente de decisiones.

2.2.2. Información que se maneja

El sistema *SGF* proporcionará entre otras funcionalidades la automatización de los procesos fundamentales en los que interviene la Fiscalía General de la República:

- Procesos Penales (*PP*)
- Verificación Fiscal (*VF*)
- Protección a los Derechos Ciudadanos (*PDC*)
- Gestión de Cuadros y Personal de Apoyo (*GCPA*)
- Relaciones Internacionales (*RI*)
- Herramientas Comunes a Todas las Áreas (*HCTA*)
- Control de la Legalidad en Establecimientos Penitenciarios (*CLEP*)
- Dirección General de Control (*DGC*)

Según el marco de trabajo en que resida cada fiscal podrá generar las actividades y tareas en el sistema dependiendo de la toma de decisiones, de las cuales es la necesidad de llevar una correcta administración, control y ejecución.

2.2.3. Descripción de la solución propuesta

Con el desarrollo del capítulo 1 se concluyó que ninguna de las herramientas utilizadas para la administración de tareas se adecuaban para el desarrollo del módulo que se necesita, por lo era necesario una nueva propuesta capaz de llevar a cabo el proceso de la administración de las tareas en el sistema *SGF*, mantener actualizados a los fiscales en todo momento del estado de los procesos así como el tiempo que disponen y mantenerlos alertas ante cualquier situación anormal con dichas tareas.

El módulo propuesto permitirá generar tareas de forma automática, y tareas de forma manual.

- Tareas generadas de forma automática (por la aplicación existente *SGF*).
  - Tarea de tipo notificación por vía del sistema.
  - Tarea de tipo notificación por vía correo.
  - Tarea de tipo seguimiento, control y notificación de casos en procesos.
- Tareas generadas de forma manual (por los usuarios autorizados).
  - Tarea de tipo notificación por vía del sistema.
  - Tarea de tipo notificación por vía correo.
  - Tarea de tipo limpieza automática de la base de datos.
  - Tarea de tipo ejecución de archivo.
  - Ejecutar script de réplica.

Además permitirá al usuario crear y gestionar todas las tareas existentes en el sistema ya sean generadas por él o por el sistema, así como ejecutar y monitorear cada una de ellas.

La solución planteada se compone de cuatro partes fundamentales, un módulo de gestión web, una base de datos, un módulo desktop y un módulo de avisos al usuario. En la figura 2 se muestra un diagrama donde se relacionan los módulos antes mencionados.

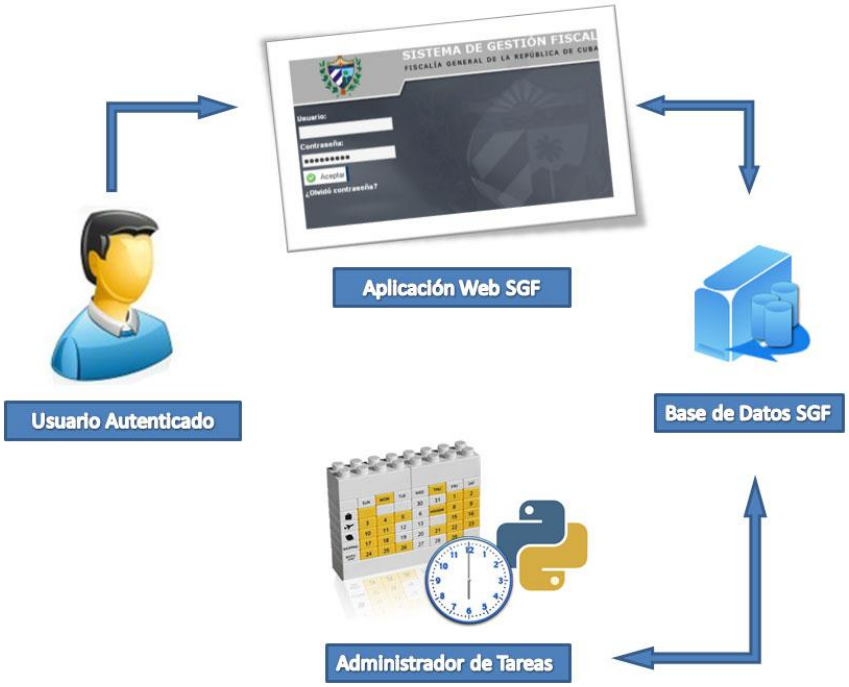


Figura 2. Propuesta de Solución.

Módulo de Gestión Web

En la web de *SGF* se encontrará un módulo solo accesible por los usuarios con permiso para estas acciones, desde donde estos usuarios podrán crear nuevas tareas o gestionar todas las tareas existentes, sin importar el tipo de creador (el sistema o el mismo usuario). Las acciones realizadas en las tareas se almacenan en la base de datos del sistema.

Base de Datos

Almacena toda la información que necesitan las tareas de *SGF* para ser ejecutadas incluyendo a las propias tareas. Sirve como medio de comunicación entre el módulo web y el módulo desktop ya que este último se nutre de esta información.

Módulo *Desktop*

Constituye una aplicación del lado del servidor, activa en todo momento que mediante hilos del sistema operativo chequea y consume tareas de la base de datos a intervalos regulares y definidos por el administrador del sistema. Todas las tareas generadas en *SGF* serán ejecutadas y monitoreadas por este módulo. Admitirá configuraciones para el envío de correos, reportes y toma de decisiones en caso de fallas.

Módulo de Aviso al Usuario

SGF debe proporcionar un espacio donde cada usuario podrá encontrar sus notificaciones. El módulo debe avisarle al usuario de forma amigable y constante en caso de una nueva notificación y darle la posibilidad de gestionar cualquiera de ellas. Debe mantenerse actualizado constantemente para proporcionar información real al usuario.

A continuación se muestra un flujo de eventos para el caso de una tarea automática generada por *SGF*.

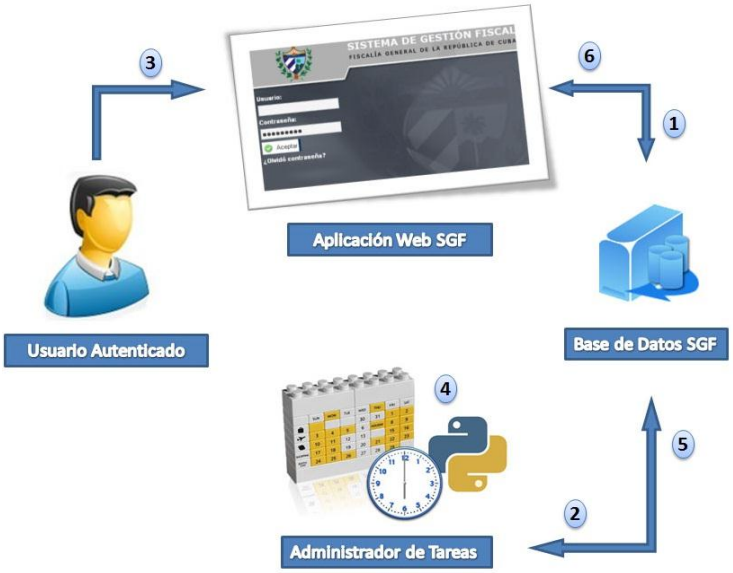


Figura 3. Flujo de eventos para una determinada tarea.

Flujo 1: El sistema (*SGF*) genera una tarea para ser ejecutada a una hora determinada y la almacena en la base de datos. (Ejemplo: hora de ejecución de la tarea 5-enero-2010 - 05:00 PM).

Flujo 2: El módulo *desktop* (Administrador de Tareas) chequea constantemente la base de datos en busca de tareas cuya hora de ejecución coincida con la hora actual del servidor para ejecutarla. (Ejemplo: hora del servidor 5-enero-2010 - 02:00 PM).

Flujo 3: El usuario autorizado modifica la tarea generada mediante una interfaz web si y solo si la tarea no ha sido ejecutada aun. (Ejemplo: Hora de modificación de la tarea 5-enero-2010 - 04:00 PM).

Flujo 4: El módulo *desktop* detecta que existe una tarea que tiene que ejecutarse en ese momento (Ejemplo: hora del servidor 5-enero-2010 - 05:00 PM = hora de ejecución de la tarea 5-enero-2010 - 05:00 PM). La tarea es ejecutada.

Flujo 5: Se actualiza la información en la base de datos con respecto a la tarea en cuestión.

Flujo 6: La tarea ejecutada produce algún efecto en *SGF*. La aplicación web no permite modificar la tarea ya que la misma está ejecutada.

2.2.4. Modelo de Dominio

Existen varias alternativas para llevar a cabo el modelamiento de un sistema, *RUP*, como metodología de desarrollo propone la realización de un modelo de negocio para el caso en el que los procesos dentro del entorno estén claramente definidos, y la realización de un modelo de dominio para los escenarios en los que no puedan identificarse tales procesos del negocio.

Después de haber realizado un estudio de los procesos que se van a efectuar, se llegó a la conclusión, que el negocio estudiado tiene muy bajo nivel de estructuración, donde los flujos de información se encuentran difusos, y cuando se desea realizar una actividad, intervienen en la misma diferentes personas de forma confusa, lo que implica un solapamiento de responsabilidades, además es difícil establecer las reglas de funcionamiento, por lo que se propone realizar un modelo de dominio.

¿Qué es el modelo de dominio?

"El modelo del dominio es un artefacto construido en la fase de concepción. Representa conceptos de la realidad física. Se utilizan para capturar y expresar el entendimiento ganando área bajo análisis antes de diseñar un sistema. Es utilizado por el analista para comprender el sector, ya sea industrial o de negocio, al que el sistema va a servir. Permite de manera visual mostrar al usuario los principales conceptos que se manejan en el dominio del sistema en desarrollo. Esto ayuda a los usuarios, clientes, desarrolladores e interesados a utilizar un vocabulario común para poder entender el contexto en que se enmarca el sistema". (17)

2.2.4.1. Conceptos del modelo de dominio

Es necesario tener un vasto conocimiento de cómo debe funcionar el proceso en cuestión, para poder capturar correctamente los requisitos y así poder construir un sistema con las características que el cliente desee. Este modelo va a contribuir posteriormente a identificar algunas clases que se utilizarán en el sistema. Primeramente se identificarán todos los conceptos que se utilizarán en el diagrama:

Concepto	Descripción
Fiscal	Usuario del sistema encargado de velar por la legalidad y el ejercicio de la acción penal pública.
Aplicación	Aplicación Web *SGF* donde se automatizan los procesos de la *FGR*.

Tareas	Tareas automáticas generadas en el sistema ante determinada acción del usuario y/o creadas directamente por él.
Administrador de Tareas	Aplicación encargada de velar por la ejecución y control de todas las tareas del sistema.

Tabla 2.1. Conceptos del Modelo de Dominio.

A continuación se muestra el Modelo de Dominio correspondiente:

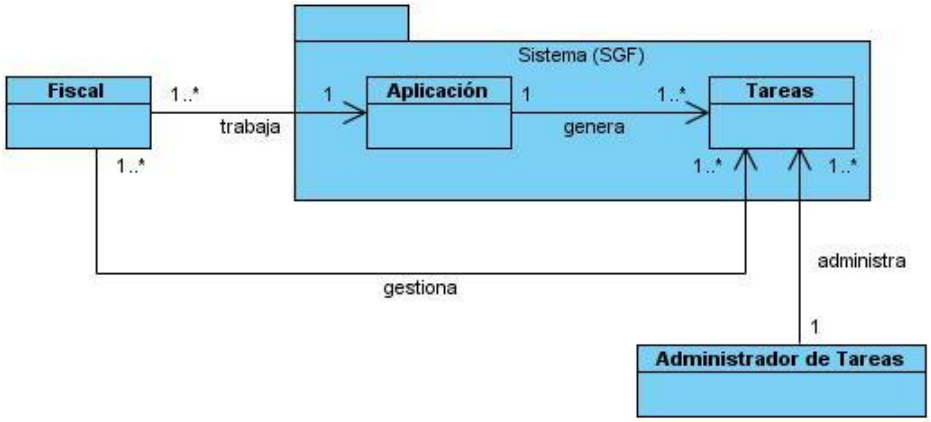


Figura 4. Diagrama de Clases del Modelo de Dominio.

2.2.5. Especificación de los Requisitos del Software

"El flujo de trabajo de requerimientos es uno de los más importantes, porque en él se establece qué es lo que tiene que hacer exactamente el sistema que se construya. En esta línea los requisitos son el contrato que se debe cumplir, de modo que los usuarios finales tienen que comprender y aceptar los requisitos que se especifiquen. Se dividen en dos grupos: los requisitos funcionales y los requisitos no funcionales.

El proceso de reunión de requisitos se intensifica y se centra específicamente en el software. Para comprender la naturaleza del (los) programa(s) a construirse, el ingeniero (analista) del software debe comprender el dominio de la información del software, así como la función requerida, comportamiento, rendimiento e interconexión". (18)

2.2.5.1. Requisitos Funcionales

Una vez descrito el modelo de dominio, para poder identificar que debe hacer el sistema y entender su funcionamiento, es fundamental conocer los requisitos funcionales que el sistema debe cumplir.

A continuación se muestran estructurados según los que debe cumplir el módulo de la aplicación web y los que debe cumplir la aplicación *desktop*.

- **Requisitos del lado de la Aplicación Web**

RF 1. Crear Tarea.

El sistema debe permitir crear cualquier tipo de tarea.

- RF 1.1. Crear Tarea de Tipo Notificación.
- RF 1.2. Crear Tarea de Tipo Ejecución de Archivos.
- RF 1.3. Crear Tarea de Limpieza a la BD.

RF 2. Mostrar Tarea.

El sistema debe mostrar al usuario todas las tareas creadas por él.

RF 3. Modificar Tarea.

El sistema debe permitir al usuario modificar las tareas que ha creado siempre y cuando no hayan sido ejecutadas.

RF 4. Eliminar Tarea.

El sistema debe permitir al usuario eliminar las tareas que ha creado.

RF 5. Buscar Tarea.

El sistema debe permitir buscar las tareas según un criterio de búsqueda definido por el usuario.

- RF 5.1. Buscar Tareas por fecha de ejecución.
- RF 5.2. Buscar Tareas ejecutadas.
- RF 5.3. Buscar Tareas no ejecutadas.
- RF 5.4. Buscar Tareas por tipo.
- RF 5.5. Buscar Tareas por creador.

RF 6. Mostrar Estado de Procesos.

El sistema debe permitir mostrar el estado de los procesos que está desarrollando un fiscal y en caso de ser un fiscal jefe debe permitir acceder al estado de los procesos de sus subordinados.

RF 7. Mostrar Notificaciones.

El sistema debe permitir mostrarle al usuario todas las notificaciones que le hayan sido enviadas.

- RF 7.1. Mostrar notificaciones nuevas.
- RF 7.2. Mostrar todas las notificaciones.

RF 8. Eliminar Notificaciones.

El sistema debe permitir al usuario eliminar sus notificaciones.

- **Requisitos del lado de la Aplicación Desktop.**

RF 9. Ejecutar Tarea.

El sistema debe ser capaz de chequear las tareas y ejecutarlas según sea el caso en el momento necesario.

RF 10. Actualizar estado de la tarea.

El sistema debe ser capaz de chequear las tareas y ejecutarlas según sea el caso en el momento necesario.

RF 11. Controlar ejecución de Tareas.

El sistema debe ser capaz de estar chequeando en todo momento si existen tareas por ejecutar.

RF 12. Configurar Sistema.

El sistema debe permitir al administrador configurar el sistema.

- RF 12.1. Configurar cuenta de correo.
- RF 12.2. Configurar intervalo de chequeo de tareas.
- RF 12.3. Configurar servidor de base de datos.
- RF 12.4. Salvar Configuración.

RF 13. Iniciar servicio de chequeo y ejecución de tareas.

El administrador debe tener la posibilidad de iniciar el servicio de chequeo de las tareas.

RF 14. Detener servicio de chequeo y ejecución de tareas.

El administrador debe tener la posibilidad de detener el servicio de chequeo de las tareas.

RF 15. Monitorear Tareas.

El sistema debe permitir al administrador listar todas las tareas que la herramienta debe estar chequeando constantemente.

- RF 15.1. Ver tareas en ejecución.
- RF 15.2. Ver tareas por ejecutar.

2.2.5.2. Requisitos no Funcionales

Una vez analizadas las funcionalidades que el sistema debe cumplir se hace necesario analizar las propiedades que el producto de software debe tener. A esto se le conoce como requisitos no funcionales.

RNF 1. Apariencia o interfaz externa:

- La interfaz debe ser sencilla, intuitiva y amigable para sus usuarios, acorde a la interfaz actual de la aplicación web *SGF*.
- La combinación de colores debe ser agradable a la vista del usuario.
- El sistema permitirá visualizar el contenido de la información de forma organizada y legible.

RNF 2. Usabilidad:

- Deberá poseer una interfaz y navegación asequibles y funcionales tanto para usuarios expertos como para los que no tienen conocimientos profundos de informática.

RNF 3. Rendimiento:

- El sistema deberá tener un nivel de respuesta aceptable, tanto para los accesos a la bases de datos, como para el proceso de administración de tareas.

RNF 4. Portabilidad:

- El sistema debe estar soportado en *Windows* y en *Linux*.

RNF 5. Disponibilidad:

- EL sistema debe estar disponible las 24 horas para permitir la ejecución de las tareas en el momento requerido.

RNF 6. Seguridad:

- El sistema debe mantener en todo momento la seguridad de la información asegurando la integridad y autenticidad de la misma.
- La seguridad se establecerá por roles que se le asignarán a los usuarios que interactúen con el sistema.

RNF 7. Confiabilidad:

- El sistema tiene debe tener soporte para recuperación ante fallos y errores.

RNF 8. Software:

- Sistema operativo: *Windows XP* (o superiores), *Linux* (Debian).

RNF 9. Hardware:

- 128 *MB* de *RAM* (Mínimo).
- Tarjeta de Red.

RNF 10. Administración:

- La herramienta debe estar bien organizada para facilitar su administración. En caso de alguna modificación en la programación debe permitir hacerse en el paquete correspondiente sin afectar a los demás.

2.2.6. Modelo de Caso de Usos del Sistema

El Modelo de Casos de Uso del Sistema es un modelo del sistema que contiene actores, casos de uso y sus relaciones.

2.2.6.1. Definición de los Actores del Sistema

Actor	Descripción
Usuario de *SGF* Autenticado	Se hace referencia al Fiscal que se autentica y accede al sistema.
Fiscal Autorizado	Fiscal que por sus permisos en el sistema tiene la posibilidad de ejercer determinadas acciones que para otro tipo de usuario no están permitidas. (Fiscales Jefes)
Administrador de la Herramienta	Persona encargada del mantenimiento y control de la herramienta.
Hilo del Sistema	Es un tipo especial de proceso informático que se ejecuta en

	segundo plano del Sistema Operativo en vez de ser controlado directamente por el usuario (es un proceso no interactivo). Se ejecutan de forma continua, aunque se intente cerrar o matar el proceso, este continuará en ejecución o se reiniciará automáticamente. Todo esto sin intervención de terceros y sin dependencia de consola alguna.

Tabla 2.2. Descripción de los Actores del Sistema.

2.2.6.2. Definición de los Casos de Uso del Sistema

Los casos de uso son artefactos narrativos que describen, bajo la forma de acciones y reacciones, el comportamiento del sistema desde el punto de vista del usuario. Por lo tanto, establece un acuerdo entre clientes y desarrolladores sobre las condiciones y requisitos que debe cumplir el sistema. A continuación se muestran los casos de uso del sistema para satisfacer los requisitos funcionales descritos con anterioridad. Las descripciones de cada caso de uso se pueden ver en el artefacto "Modelo de Casos de Uso del Sistema".

***CU* - 1**	Crear Tarea
Actor:	Fiscal Autorizado
Descripción:	El caso de uso comienza cuando un Fiscal Autorizado decide crear determinada tarea pudiendo ser una tarea de notificación, ejecución de archivos, o limpieza a la *BD*.
Prioridad:	Crítico.
Referencias	*RF* – 1.

Tabla 2.3. Resumen del *CU* Crear Tarea

***CU* – 2**	Gestionar Tarea
Actor:	Fiscal Autorizado
Descripción:	El caso de uso comienza cuando un Fiscal Autorizado desea realizar determinada acción con las tareas, ya sea Editar o Eliminar alguna de ella.
Prioridad:	Crítico.
Referencias:	*RF* – 2, *RF* – 3, *RF* – 4.

Tabla 2.4. Resumen del *CU* Gestionar Tarea.

***CU* – 3**	Buscar Tareas
Actor:	Fiscal Autorizado
Descripción:	El caso de uso comienza cuando un Fiscal Autorizado decide gestionar determinada tarea.
Prioridad:	Crítico
Referencia	*RF* – 5.

Tabla 2.5. Resumen del *CU* Buscar Tarea.

***CU* – 4**	Mostrar Estado de Procesos
Actor:	Usuario de *SGF* Autenticado
Descripción:	El caso de uso comienza cuando un Fiscal desea consultar el estado en que se encuentran sus procesos, ya sean los procesos que lleva dicho usuario o en caso de ser un Fiscal Jefe consultar los de sus subordinados.
Prioridad:	Secundario
Referencia:	*RF* – 3

Tabla 2.6. Resumen del *CU* Mostrar Estado de Procesos.

***CU* – 5**	Gestionar Notificaciones
Actor:	Usuario de *SGF* Autenticado
Descripción:	El caso de uso comienza cuando un Fiscal desea consultar las notificaciones que le han llegado, así como eliminar aquellas que ya no desee que permanezcan es su listado de notificaciones.
Prioridad:	Crítico
Referencia	*RF* – 7, *RF* – 8.

Tabla 2.7. Resumen del *CU* Gestionar Notificaciones.

***CU* – 6**	Ejecutar Tarea
Actor:	Hilo del Sistema
Descripción:	El caso de uso consiste en permitir a la herramienta ejecutar las tareas pendientes en el momento requerido.
Prioridad:	Crítico
Referencia	*RF* – 9, *RF* – 10, *RF* – 11.

Tabla 2.8. Resumen del *CU* Ejecutar Tareas.

***CU* – 7**	Monitorear Sistema
Actor:	Administrador de la Herramienta
Descripción:	El caso de uso comienza cuando el Administrador de la herramienta accede a ella para realizar determinada acción, ya sea realizar algún tipo de configuración la herramienta o ver las tareas existentes.
Prioridad:	Crítico
Referencia	*RF* – 12, *RF* – 13, *RF* – 14, *RF* – 15.

Tabla 2.9. Resumen del *CU* Monitorear Sistema.

2.2.7. Diagrama de Casos de Uso del Sistema

En el siguiente diagrama se representa la relación entre los actores y los casos de uso del sistema. Se realizó un *CRUD* Parcial (patrón de caso de uso) en el *CU*: Gestionar Tareas dividiendo el mismo en dos *CU*, el *CU*: Gestionar Tarea, donde se editan y eliminan las tareas y el *CU*: Crear Tares, donde se crean las tareas.

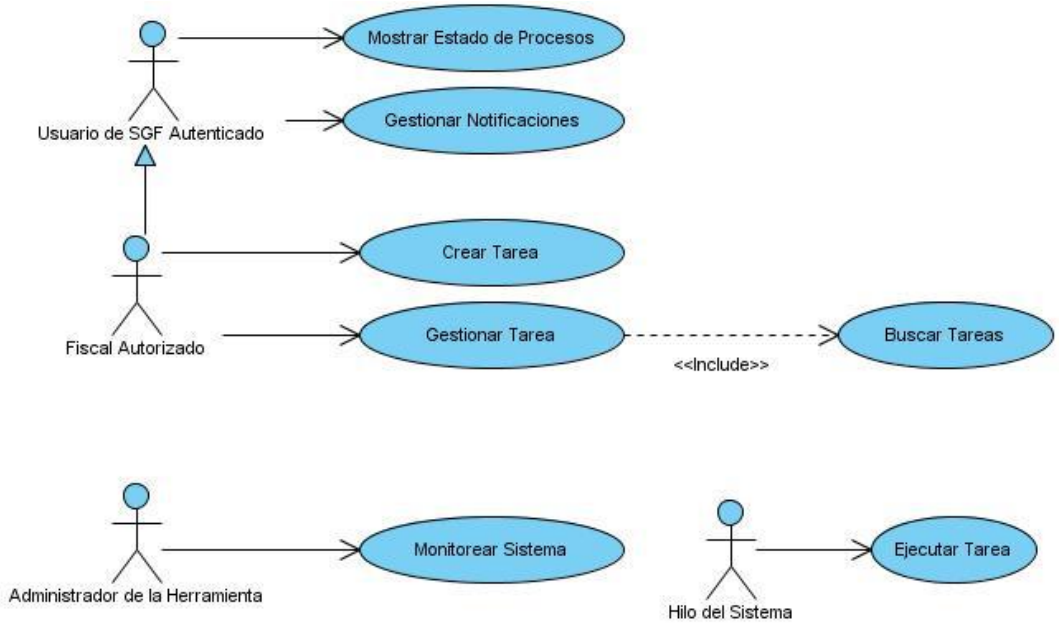


Figura 5. Diagrama de Caso de Usos del Sistema.

2.3. Conclusiones del capítulo

Luego de la realización de este capítulo se puede concluir que se realizó la descripción del sistema y el modelado del mismo en un modelo de dominio, así como la captura de los requisitos funcionales y no funcionales que debe cumplir el módulo. Se definieron los actores y casos de uso del sistema, representando en un diagrama de casos de uso las relaciones que existen entre ambos.

CAPÍTULO III: DISEÑO DEL SISTEMA

3.1. Introducción

En el presente capítulo se describen los pasos que se llevaron a cabo en el diseño de la aplicación siguiendo específicamente los pasos establecidos en la metodología de desarrollo seleccionada, lo que será de gran ayuda para la comprensión del sistema propuesto. Como parte de esta solución se explican las clases, los patrones utilizados, la arquitectura definida por los arquitectos del proyecto y se desarrollan los diagramas de clases del diseño así como las descripciones de dichas clases y el modelo de clases persistentes.

3.2. Flujo de Análisis y Diseño

“El modelo del análisis es un artefacto que usualmente se genera para entender mejor los requisitos y realizar mejor el diseño. Es típicamente un artefacto temporal, que brinda una aproximación al diseño y se realiza normalmente cuando no se tiene una idea clara de los procesos y requerimientos”. (16)

Se realiza el análisis para lograr un mejor acercamiento a lo que se desea construir, para sentar las bases del diseño.

Según un estudio realizado, se decide no realizar el modelo de análisis pues los procesos y requerimientos están bien definidos. Aunque el análisis ofrece una visión general del sistema, que puede ser el primer paso para el diseño, este no abarca el entorno de implementación que sí se tiene en cuenta en el diseño; lo que permite gestionar mejor el avance del proceso de desarrollo.

3.2.1. Modelo de Diseño

“El modelo de diseño es un modelo de objetos que describe la realización física de los casos de uso, centrándose en cómo los requisitos funcionales y no funcionales, junto con otras restricciones relacionadas con el entorno de implementación, tienen impacto en el sistema a considerar. El modelo de diseño sirve de abstracción a la implementación y se utiliza como una entrada fundamental de las actividades de implementación”. (16)

3.2.1.1. Patrón Arquitectónico *MVC* (Modelo – Vista – Controlador)

Para el desarrollo de la aplicación, como se explicó en el capítulo 1, se utilizará el *framework Symfony*; el cual está basado en un patrón clásico del diseño web conocido como arquitectura *MVC*. Este está formado por 3 niveles, el Modelo, la Vista y el Controlador:

- El modelo representa la información con la que trabaja la aplicación, es decir, su lógica de negocio.
- La vista transforma el modelo en una página web que permite al usuario interactuar con ella.
- El controlador se encarga de procesar las interacciones del usuario y realiza los cambios apropiados en el modelo o en la vista.

"La arquitectura *MVC* separa la lógica de negocio (el modelo) y la presentación (la vista) por lo que se consigue un mantenimiento más sencillo de las aplicaciones. Si por ejemplo una misma aplicación debe ejecutarse tanto en un navegador estándar como un navegador de un dispositivo móvil, solamente es necesario crear una vista nueva para cada dispositivo; manteniendo el controlador y el modelo original. El controlador se encarga de aislar al modelo y a la vista de los detalles del protocolo utilizado para las peticiones (*HTTP*, consola de comandos, *email*, etc.). El modelo se encarga de la abstracción de la lógica relacionada con los datos, haciendo que la vista y las acciones sean independientes de, por ejemplo, el tipo de gestor de bases de datos utilizado por la aplicación". (11)

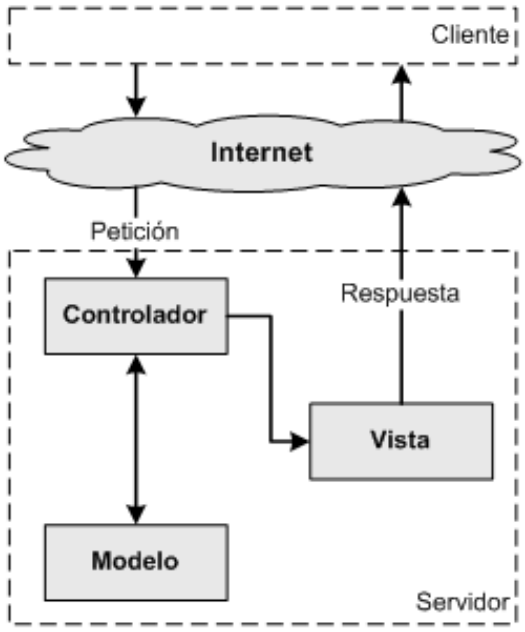


Figura 6. Patrón *MVC*.

El uso de *Symfony* resulta bastante útil además de restrictivo ya que obliga a dividir y organizar el código de acuerdo a las convenciones establecidas por el mismo. El código de la presentación se guarda en la vista, el código de manipulación de datos se guarda en el modelo y la lógica de procesamiento de las peticiones constituye el controlador.

"La implementación que realiza *Symfony* de la arquitectura *MVC* incluye varias clases como son:

- ***sfController*:** es la clase del controlador. Se encarga de decodificar la petición y transferirla a la acción correspondiente.
- ***sfRequest*:** almacena todos los elementos que forman la petición (parámetros, *cookies*, cabeceras)
- ***sfResponse*:** contiene las cabeceras de la respuesta y los contenidos. El contenido de este objeto se transforma en la respuesta *HTML* que se envía al usuario.
- El ***singleton*** de contexto almacena una referencia a todos los objetos que forman el núcleo de *Symfony* y puede ser accedido desde cualquier punto de la aplicación.

Symfony toma lo mejor de la arquitectura *MVC* y la implementa de forma que el desarrollo de aplicaciones sea rápido y sencillo.

El controlador frontal es un componente generado automáticamente por el *framework* que, unido con el *layout* son comunes para todas las páginas de la aplicación. Las clases de la capa del modelo son generadas automáticamente en función de la estructura de datos de la aplicación por el *sub-framework Propel* y la abstracción de la base de datos es completamente invisible al programador, ya que es realizada mediante un componente llamado *Creole*". (11)

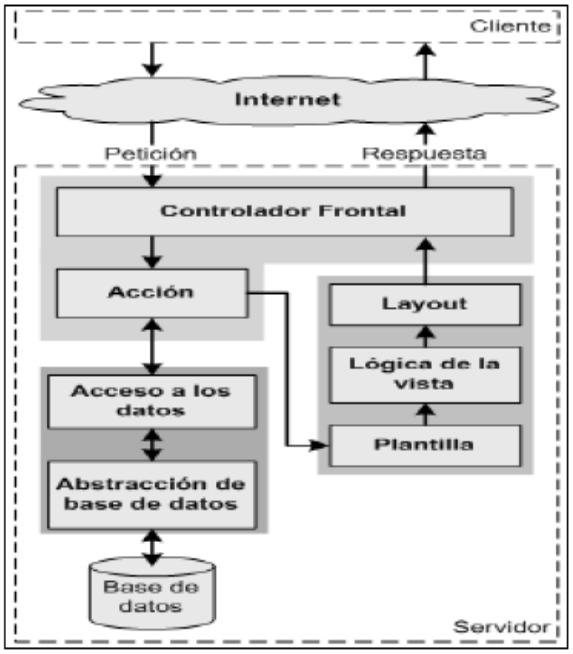


Figura 7. Estructura del *MVC* en *Symfony*.

3.2.1.2. Aplicación de los Patrones en *Symfony*

Se debe conocer la arquitectura del *framework* para una mejor comprensión del comportamiento que seguirán las clases contenidas en él y su estructura; la arquitectura brinda una vista panorámica del diseño de este sistema. Son muchos los patrones que se utilizan en la implementación con *Symfony.* A continuación se mencionan algunos ejemplos de los evidenciados, ubicándolos en las capas de Modelo y Control que plantea el patrón arquitectónico *MVC*.

Patrón Experto: En el modelo de la arquitectura *Symfony* existen dos tipos de clases que son fundamentales:

- Las encargadas de la abstracción de datos. (realizan todas las operaciones con la *BD*).
- Las de acceso a datos. (interactúan con las clases de abstracción de datos y devuelven los objetos necesarios por los controladores).

Por cada tabla *Symfony* genera 4 tablas: Clase, ClasePeer, BaseClase y BaseClasePeer (ejemplo de clases: tbtarea, tbtrareaPeer, Basetbtarea, BasetbtareaPeer). Las clases a las que el *framework* añade el sufijo Peer trabajan directamente con la base de datos y por lo tanto son las encargadas de la abstracción utilizando *Propel*, en ellas se encuentran los atributos necesarios para este proceso, de ahí

la necesidad de que implementen la responsabilidad de efectuar las operaciones con la base de datos, aplicando de esta manera el patrón experto.

Patrón Bajo Acoplamiento: Las clases que implementan la lógica de negocio y de acceso a datos se encuentran en el modelo, estas clases no tienen asociaciones con las de la vista o el controlador por lo que la dependencia en este caso es baja, cumpliéndose así perfectamente el patrón.

Patrón Controlador: El controlador se encarga de asignar la responsabilidad de controlar el flujo de eventos del sistema a clases específicas con las que mantiene un modelo de alta cohesión, esto facilita la centralización de actividades (validaciones, seguridad). Un ejemplo del patrón controlador se puede ver desde la clase *sfFrontController, sfWebFrontController, sfContex, los actions* , y el *index.php* del ambiente. *Symfony* implementa el patrón *Front Controller* (controlador frontal), en el cual existen varias clases controladoras que forman un flujo para atender las peticiones del usuario y de otras clases. La arquitectura del *framework* ayuda desde el principio, existiendo una capa específica para los controladores, que son el núcleo del mismo.

3.2.1.3. Diagrama de Clases del Diseño

Un diagrama de clases representa las clases que serán utilizadas dentro del sistema y las relaciones que existen entre ellas. La definición de clase incluye definiciones para atributos y operaciones. Los diagramas de clases por definición son estáticos, y representan qué partes interactúan entre sí.

A continuación se muestran algunos de los Diagramas de Clases del Diseño por casos de uso, necesarios para llevar a cabo la implementación del sistema. Primeramente se muestra el diagrama de clases de la aplicación *desktop* y posteriormente del módulo de la web el Gestionar Tarea de tipo Notificación. El resto de los diagramas se encuentran en el artefacto “Modelo de Diseño”.

3.2.1.3.1. Diagrama de Clases de la Aplicación *Desktop*

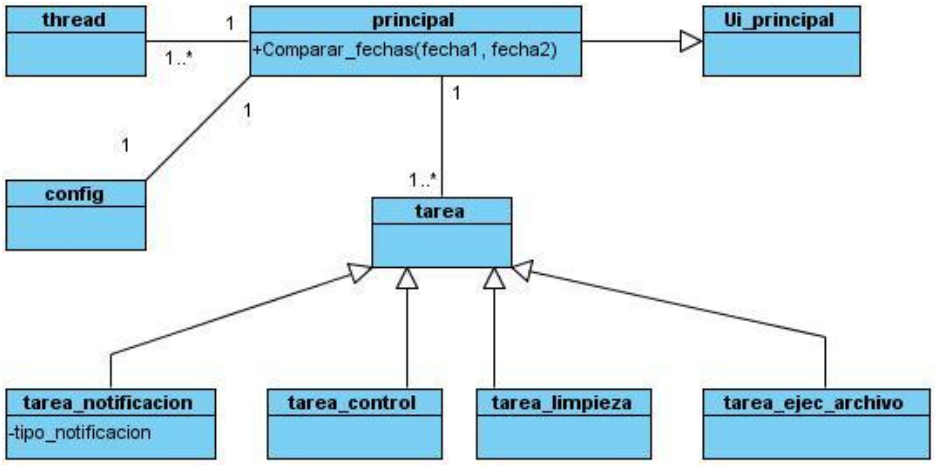


Figura 8. Diagrama de Clases del Robot Administrador de Tareas.

3.2.1.3.2. Diagrama de Clases del Módulo Web

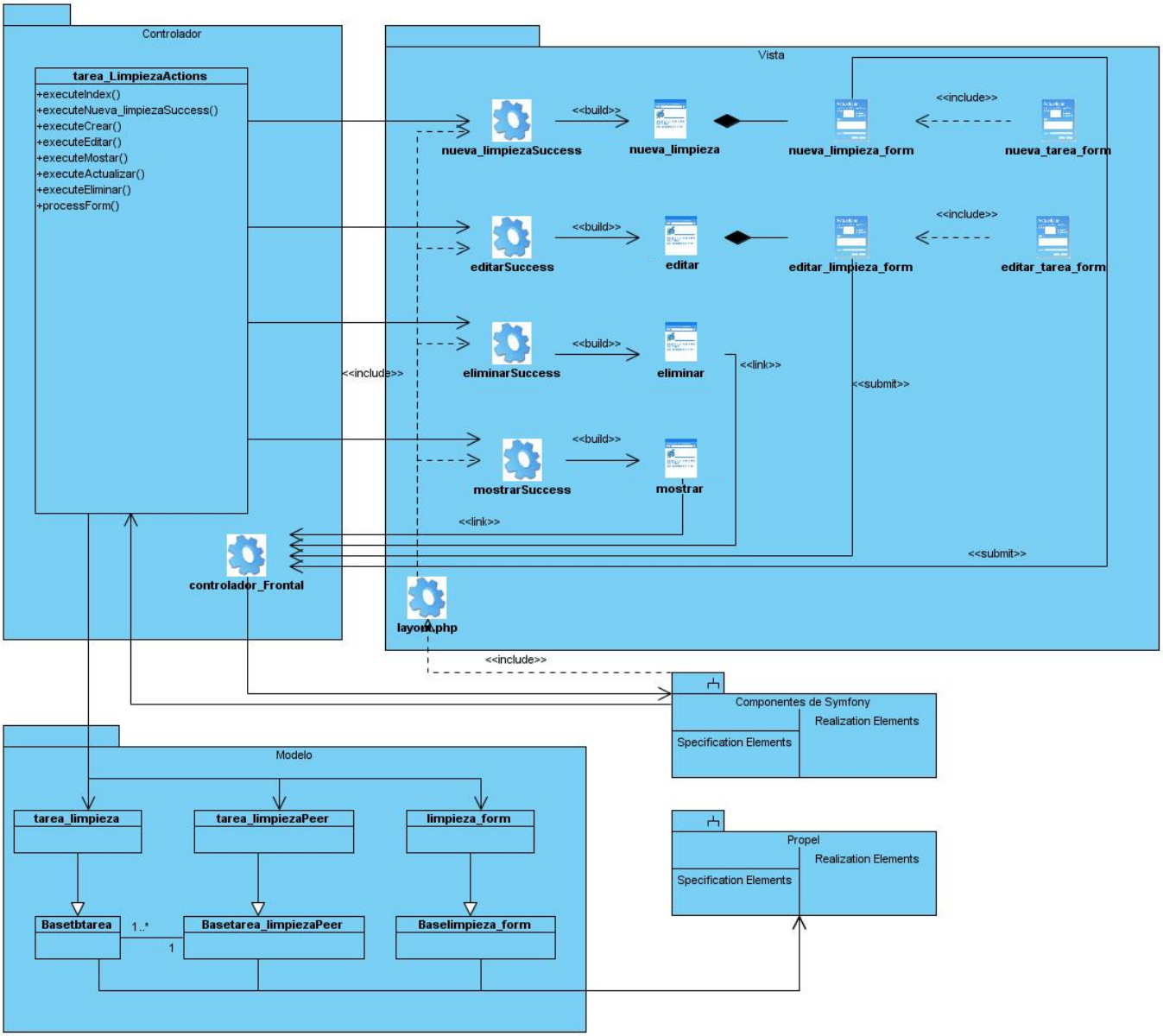


Figura 9. Diagrama de Clases del Diseño del CU Gestionar Tarea (Ejecución de Archivos).

3.2.1.4. Descripción de las Clases del Diseño

El Modelo de Diseño es un proceso para la definición detallada de un sistema con el fin de la realización física de los casos de uso para cubrir las funciones que realizará el sistema y otras restricciones del entorno de implementación que tienen impacto en el mismo, por tanto en él se definen las clases del diseño que conformarán el sistema que se va a implementar.

- **Páginas clientes**: Son las páginas encargadas de permitir a los usuarios interactuar con el sistema tanto para hacer solicitudes como para que sean mostradas las respuestas a las mismas.
- **Páginas servidoras**: Son las encargadas de la construcción de forma dinámica de las páginas clientes y sirven de enlace entre estas y el resto de las clases.
- **Páginas controladoras**: Son las responsables de realizar las operaciones que responden a los procesos de negocio y dar respuestas a las solicitudes hechas por el usuario.
- **Clases entidad:** Son las responsables de la persistencia de los datos físicamente.

A continuación se describe cada una de las clases del diseño y cada uno de los paquetes y subsistemas que conforman la solución.

Paquete controlador: Una parte importante de su trabajo es común a todos los controladores de la aplicación. Entre las tareas comunes se encuentran el manejo de las peticiones del usuario, el manejo de la seguridad, cargar la configuración de la aplicación y otras tareas similares. Por este motivo, el controlador se ha dividido en un controlador frontal, que se encarga de realizar las tareas comunes y las acciones (*actions*), que incluyen el código específico del controlador de cada página. Habrá un *actions* por cada uno de los módulos pero el controlador frontal será el mismo para todo el sistema.

Paquete vista: Las páginas *web* suelen contener elementos que se muestran de forma idéntica a lo largo de toda la aplicación: cabeceras de la página, el *layout* genérico, el pie de página y la navegación global. En la mayor parte de las veces sólo cambia el interior de la página. Por este motivo, la vista se separa en un *layout* y en una plantilla. El *layout* será global en toda la aplicación o para la mayoría de las páginas. La plantilla sólo se encarga de visualizar las variables definidas en el paquete del controlador.

Paquete del modelo: Solo contiene las clases encargadas del acceso a los datos almacenados en el gestor de base de datos, las cuales utilizan el *ORM* (*Object Relational Model*) *Propel* para el acceso a los mismos.

Las clases y subsistemas que se relacionan a continuación son comunes para cada módulo y debido a que son producto de cada proyecto de *symfony* se describen brevemente para un mayor entendimiento.

Clases Base: Las clases con nombre *Base* del directorio *lib/model/om/* son las que se generan directamente a partir del esquema. Nunca se deberían modificar porque cada vez que se genera el modelo se borran todas las clases.

Clases de Objetos: Las clases de objetos propias que están en el directorio *lib/model* heredan de las clases con nombre Base. Estas clases no se modifican cuando se ejecuta la tarea *propel:build-model*, por lo que son las clases en las que se añaden los métodos propios. Esta clase hereda todos los métodos de la clase Base, pero no le afectan las modificaciones en el esquema.
Este mecanismo de clases personalizadas que heredan de las clases base permite empezar a programar desde el primer momento, sin ni siquiera conocer el modelo relacional definitivo de la base de datos. La estructura de archivos creada permite personalizar y evolucionar el modelo.

Clases Peer: Son las clases que tienen métodos estáticos para trabajar con las tablas de la base de datos. Proporcionan los medios necesarios para obtener los registros de las tablas. Sus métodos devuelven normalmente un objeto o una colección de objetos de la clase objeto relacionada.

Las clases **BaseClase** y **BaseClasePeer**, tienen una relación de asociación sin navegabilidad porque ambas pueden crear instancias una de la otra. La multiplicidad expresa que una instancia de la clase BaseClasePeer puede crear 1 o varias instancias de la clase BaseClase cuando se ejecuta.

La combinación de las clases objeto y las clases "*peer*" y las versiones básicas y personalizadas de cada una hace que se generen 4 clases por cada tabla del esquema.

Subsistema Propel: Su función es gestionar el acceso a la base de datos y gestionar el modelo del sistema. Implica que el acceso y la modificación de los datos almacenados en la base de datos se

realicen mediante objetos, nunca de forma explícita, permitiendo un alto nivel de abstracción y fácil portabilidad. *Propel* tiene incluido tareas para generar automáticamente las sentencias *SQL* necesarias para crear las tablas de la base de datos. La librería *Propel* se encarga de esta generación automática, ya que crea el esqueleto o estructura básica de las clases y genera automáticamente el código necesario. La abstracción de la base de datos es completamente transparente para el programador, ya que se realiza de forma nativa mediante *PDO* (*PHP Data Objects*).

Subsistema Componentes de *Symfony*: Representa todas las clases del *framework Symfony* que serán utilizadas durante el funcionamiento del sistema. Dígase validadores de formularios, *helpers* de objetos y formularios, plantillas, componentes de seguridad, etc.

Las descripciones detalladas de las clases del diseño se encuentran en el artefacto “Modelo de Diseño”.

3.2.1.5. Diagrama de Clases Persistentes

La persistencia es la capacidad de un objeto de mantener su valor en el espacio y en el tiempo. Las clases persistentes referencian directamente las entidades lógicas y sus atributos. Para definir las clases persistentes se aplicaron las siguientes reglas:

- Cuando una clase que está formada por otras clases es persistente, automáticamente las clases componentes también son persistentes. Lo contrario no se cumple necesariamente.
- Cuando una clase hija de una jerarquía es persistente, automáticamente son persistentes sus ancestros en el árbol de jerarquía. Lo contrario no se cumple necesariamente.
- Cuando hay herencia múltiple, esta debe ser resuelta antes si el medio de almacenamiento a utilizar no soporta este concepto. La solución más factible es que la clase hija herede de la clase, de la que redefine sus métodos y añada un atributo pasivo del tipo de la otra clase de la que heredaba. Si se redefinía comportamiento de más de una clase padre, hay que escoger de cual se quedaría heredando y añadir un atributo pasivo por cada una de las clases padres de las que no hereda. Los métodos que se redefinen que ya no se reciben por herencia, en su implementación incluirán las relaciones con las clases padres.

El siguiente diagrama de clases persistentes es una parte del diagrama de clases del proyecto *SGF* ya que el sistema propuesto constituye un módulo del mismo.

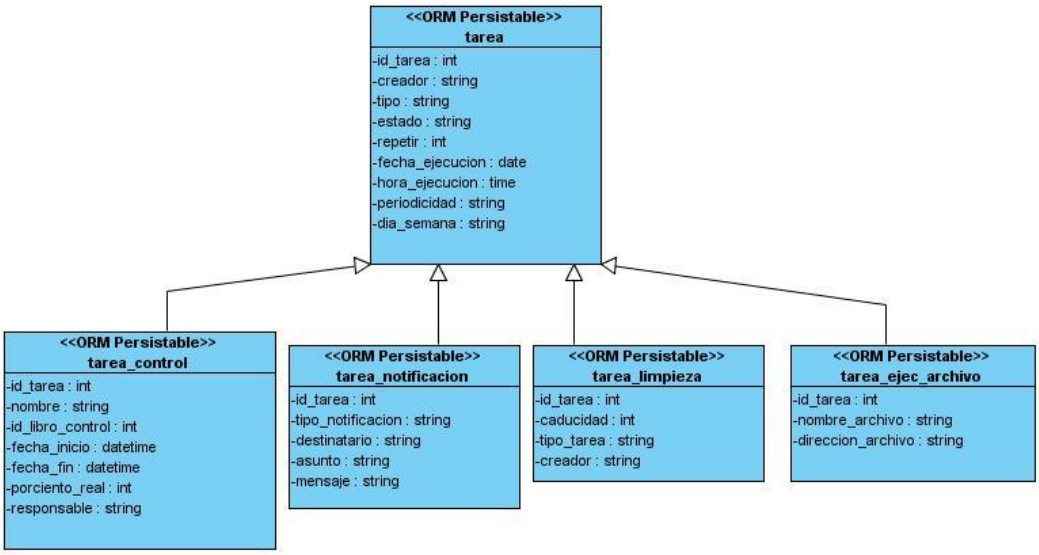


Figura 10. Diagrama de Clases Persistentes.

3.2.1.6. Modelo de Datos

El modelo de datos describe la representación lógica y física de la persistencia de los datos utilizados por la aplicación. Estará compuesto por las entidades que pasarán a ser las tablas de la base de datos que será utilizada por el sistema.

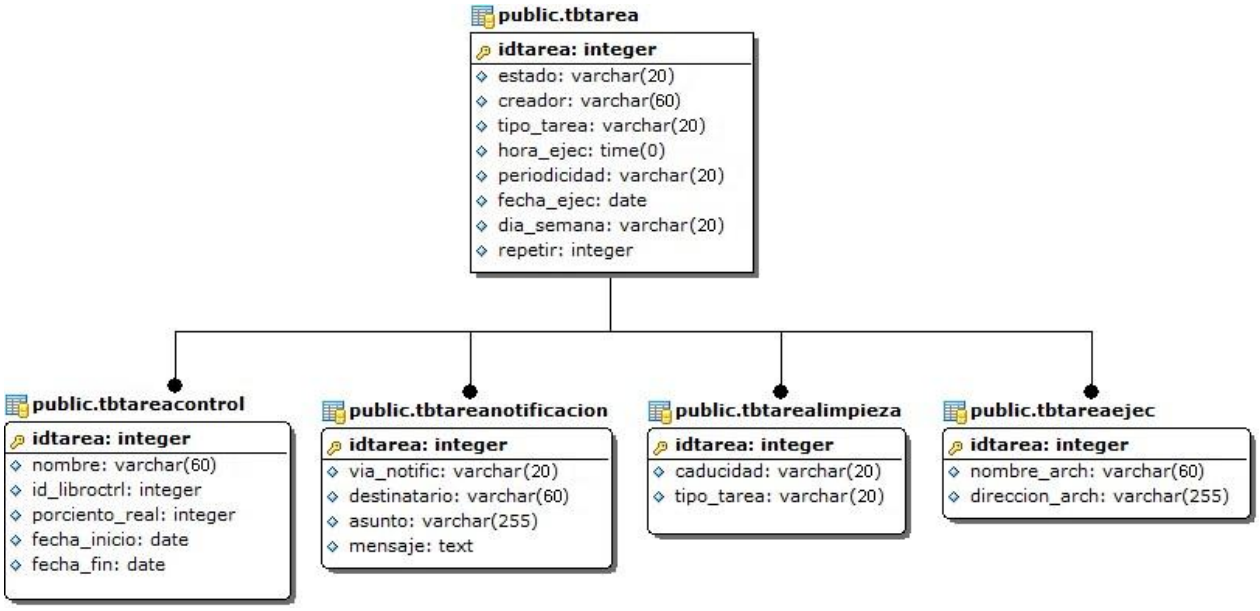


Figura 11. Modelo de Datos.

3.3. Conclusiones del capítulo

Con la culminación de este capítulo se puede concluir que se trataron temas relacionados con la arquitectura seleccionada permitiendo realizar el diseño del sistema que se presenta, así como elementos específicos dentro del mismo que quedan plasmados en el artefacto Modelo de Diseño debido a su extensión.

CAPÍTULO IV: IMPLEMENTACIÓN Y PRUEBA

4.1. Introducción

En el presente capítulo partiendo de los resultados en el diseño se presentan los Diagramas de Despliegue y de Componentes como objetivo primordial de la fase de implementación. Luego se valida la solución propuesta a través del método de caja negra con la realización de los diferentes casos de prueba.

4.2. Modelo de Implementación

En el modelo de implementación se describe cómo los elementos del modelo de diseño se implementan en términos de componentes, (ficheros de código fuente, ejecutables). El mismo está conformado por los Diagramas de Componentes y de Despliegue, los cuales describen los componentes a construir, y su organización y dependencia entre nodos físicos en los que funcionará la aplicación.

4.2.1. Diagrama de Despliegue

En el diagrama de despliegue se indica la situación física de los componentes lógicos desarrollados. Es decir se sitúa el software en el hardware que lo contiene. A continuación se muestra el diagrama de despliegue correspondiente a la aplicación.

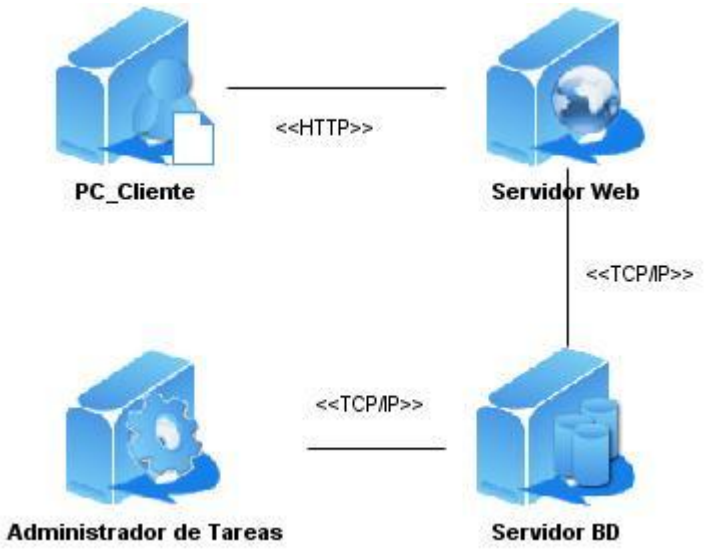


Figura 12. Diagrama de Despliegue.

4.2.2. Diagramas de Componentes

Los diagramas de componentes representan cómo un sistema es dividido en componentes y muestra las dependencias entre estos. Los componentes físicos incluyen archivos, cabeceras, librerías compartidas, módulos, ejecutables, o paquetes. Pueden ser usados para modelar y documentar cualquier arquitectura de sistema. Uno de los usos principales es que puede servir para ver qué componentes pueden compartirse entre sistemas o entre diferentes partes de un sistema.

Paquete Vista: Paquete que agrupa todos los componentes relacionados con la vista de la aplicación tales como:

- Componente *layout*.php: Componente que implementa la clase *layout* del diseño. Contiene los elementos que se muestran de forma idéntica a lo largo de toda la aplicación.
- Componentes *success*: Componentes que implementan el código correspondiente a cada plantilla que utiliza el módulo.

Paquete Controlador: Paquete que agrupa todos los componentes *actions.class.php* de cada uno de los módulos que son implementados por las clases *actions* del diseño. Estos componentes incluyen el código específico del controlador para cada página del módulo.

Paquete Modelo: Se especifican las clases generadas por el subsistema *Propel* que es el *ORM* que utiliza *Symfony* el cual proporciona persistencia para los objetos y un servicio de consultas. *Propel* a su vez utiliza el componente *Creole* como sistema de abstracción de la base de datos, sistema similar a los *PDO* (*PHP Data Object*) y proporciona una interfaz entre el código *PHP* y el código *SQL* de la base de datos, permitiendo cambiar fácilmente de sistema gestor de base de datos.

A continuación se muestran los diagramas de componentes desarrollados:

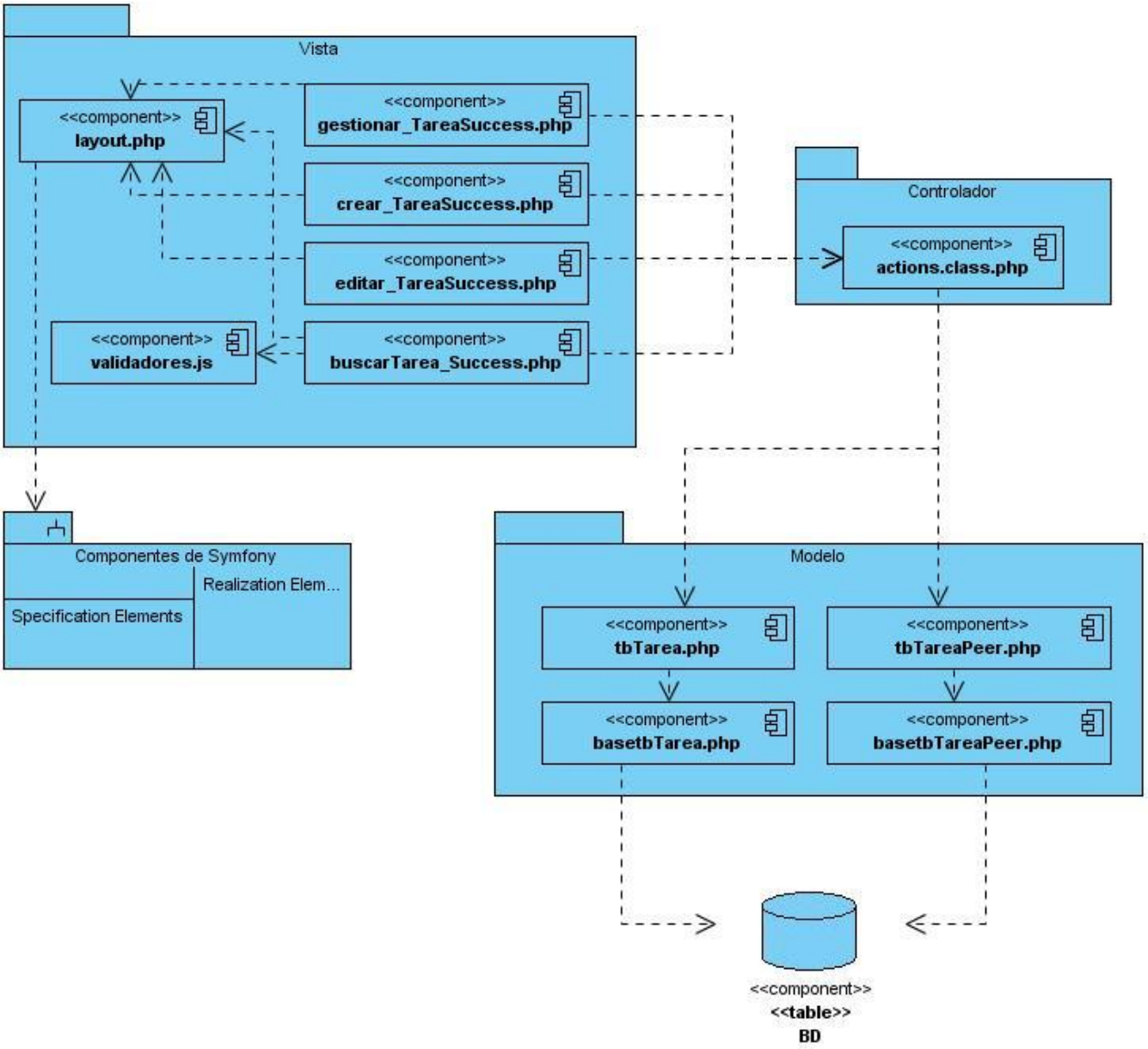


Figura 13. Diagrama de Componentes Gestionar Tarea.

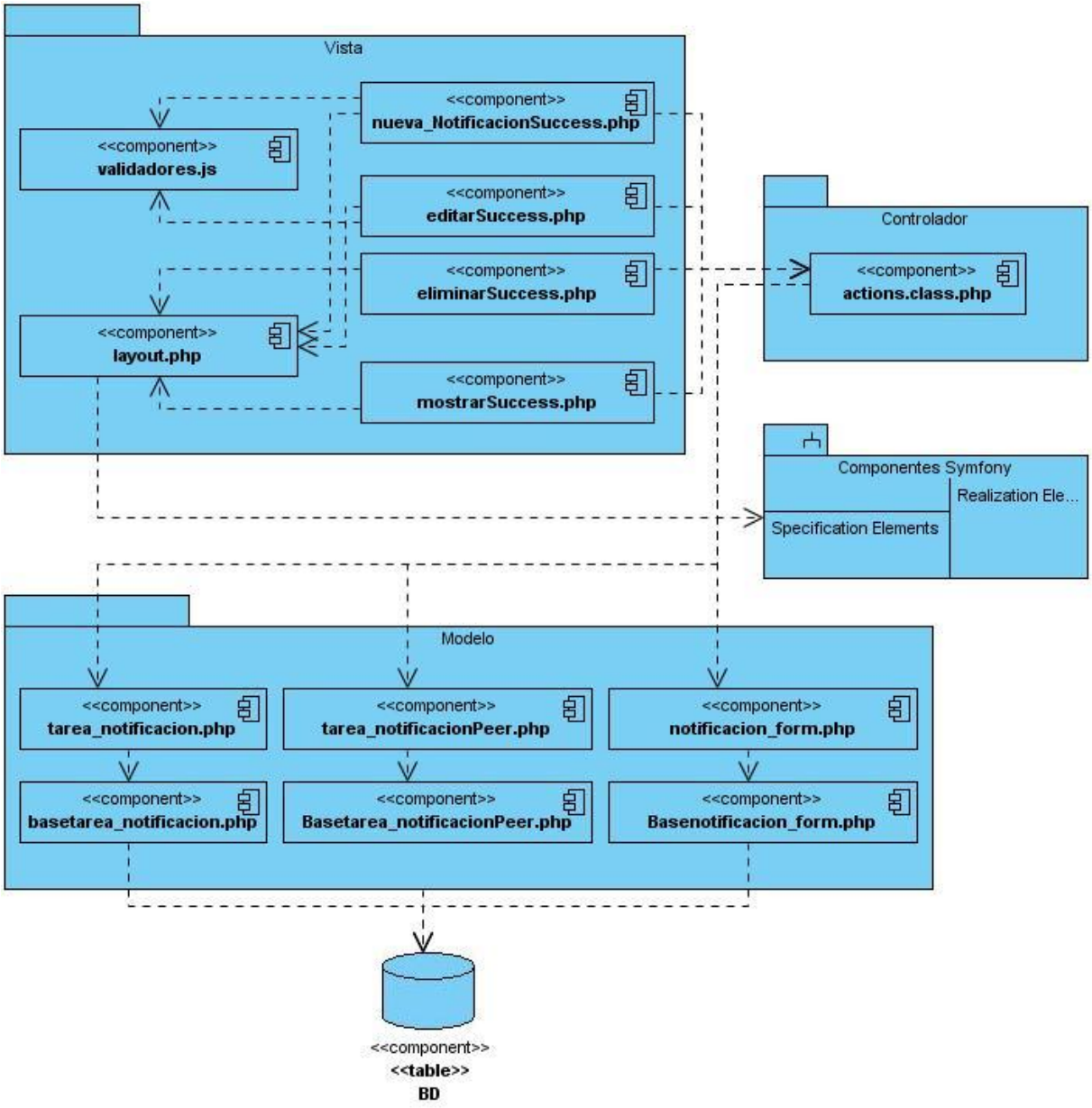


Figura 14. Diagrama de Componentes Gestionar Tarea de Notificación.

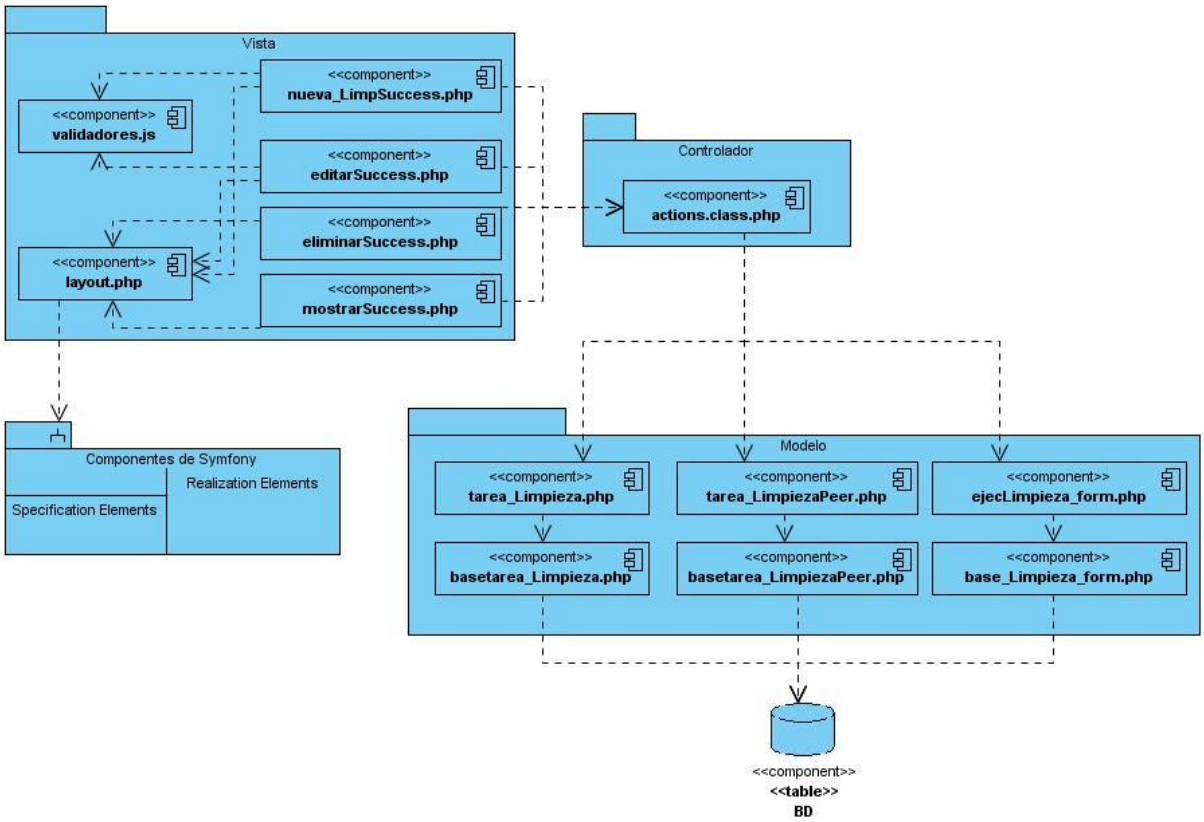


Figura 15. Diagrama de Componentes Gestionar Tarea de Limpieza.

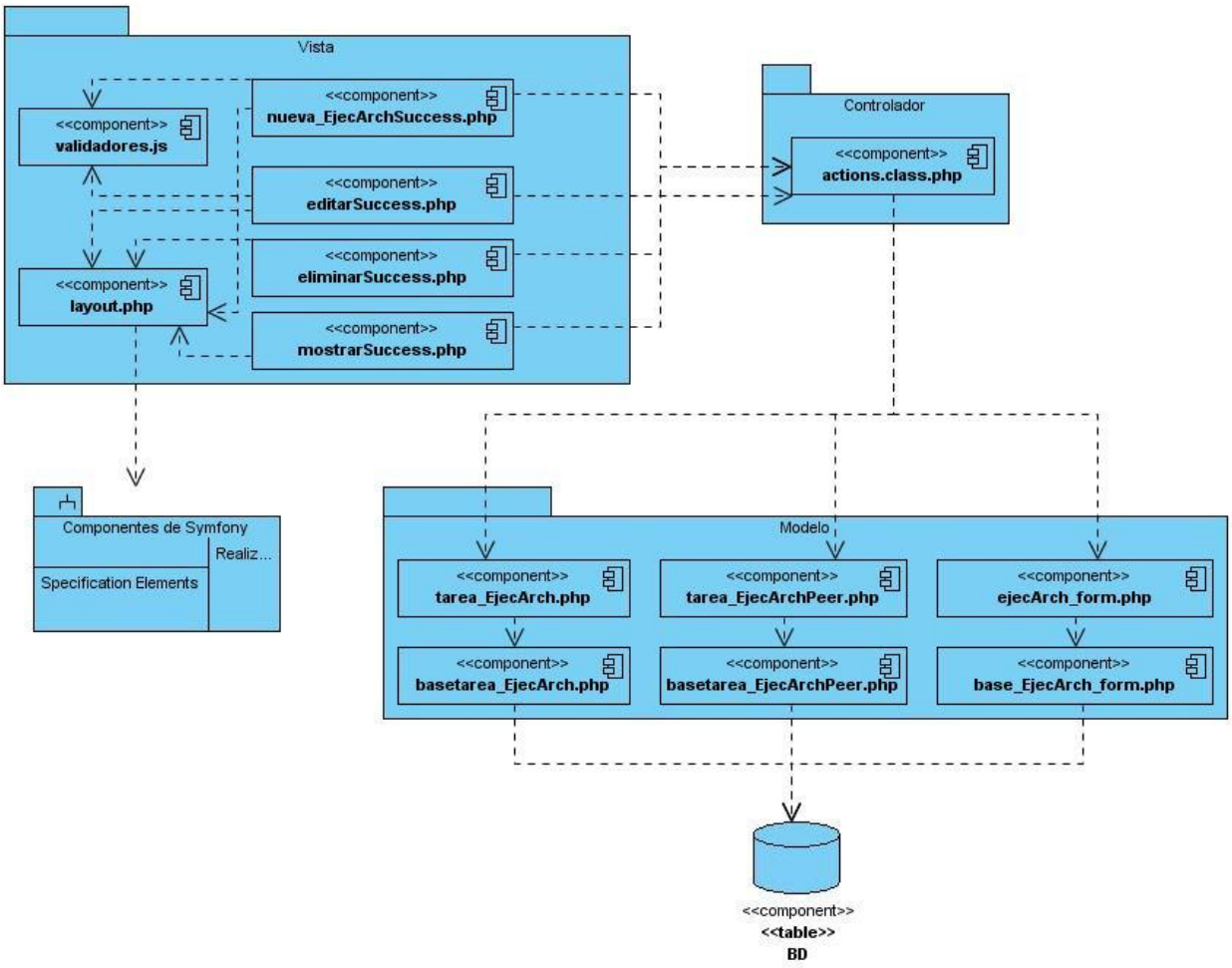


Figura 16. Diagrama de Componentes Gestionar Tarea de Ejecución de Archivo.

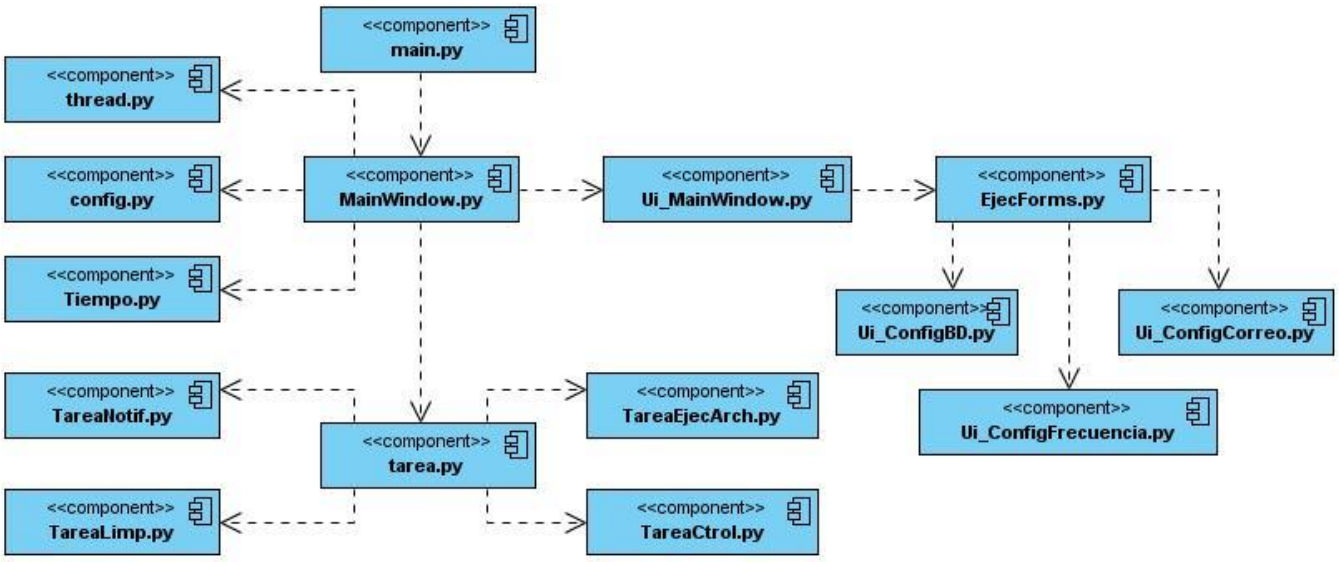


Figura 17. Diagrama de Componentes [Robot]

4.3. Estándares de código

4.3.1. Estándares de desarrollo para *PHP*

Estándar 1: Las funciones deben ser llamadas sin espacios entre el nombre de la función, el signo de paréntesis y el primer parámetro; espacios entre cada coma por parámetro y sin espacios entre el ultimo paréntesis, el signo de paréntesis cerrado y el signo de punto y coma (**;**).

Estándar 2: El estilo de los comentarios debe ser como el estilo de comentarios para *C* (/* */ ó //), no debe de utilizarse el estilo de comentarios de Perl (#).

Estándar 3: Cuando se incluya un archivo de dependencia incondicionalmente utilice **require_once** y cuando sea condicionalmente, utilice **include_once**.

Estándar 4: Siempre utilice las etiquetas <?php?> para abrir un bloque de código. No utilice el método de etiquetas cortas, porque esto depende de las directivas de configuración en el archivo *PHP.INI* y hace que el script no sea tan portable.

Estándar 5: Los nombres de las clases deben de iniciar con letra mayúscula. Los nombres de las variables y de las funciones pueden iniciar con letra minúscula, pero si estas tienen más de una palabras, cada nueva palabra debe iniciar con letra mayúscula (el nombre puede escribirse separado por signos de guión mayor). Si una función, en una clase, es privada; deberá comenzar

con el signo de guión mayor para una fácil identificación. Las constantes deben de escribirse siempre en mayúsculas y tanto estas como las variables globales deben de tener como prefijo el nombre de la clase a la que pertenecen.

Estándar 6: Los archivos con código *PHP*, deben de ser guardados en formato *ASCII* utilizando la codificación *ISO-8859-1*. El formato *ASCII* con codificación *ISO-8859-1*, es el formato en que se guardan los archivos de texto plano (.txt). La razón de este estándar es que determinados editores *HTML* (en especial Dreamweaver), agregan códigos de carácter extraño de salto de línea (como si se tratara de un archivo binario) y esto puede ocasionar que el intérprete de *PHP*, encuentre problemas a la hora de leer el script.

4.3.2. Para mostrar una cadena

Debe estar dentro de comillas dobles o simples (ejemplo: "Hola Mundo", 'Lo que quiero mostrar'). Cabe destacar que si se desea mostrar el símbolo " o ' debe encerrarse en el otro tipo de comillas ("...'...", '..."...') o usarse un escape (\', \"). Toda línea de instrucción siempre termina en un punto y coma (;), al igual que el lenguaje *C*.

4.3.3. Insertar un comentario

Para insertar un comentario de una línea debe empezar por // o por #. El resto de la línea es tratado entonces como un comentario.
Para insertar un bloque de comentario, de una o más líneas, se utiliza la combinación /* y */, por ejemplo: /* <*COMENTARIOS*> */

4.4. Prueba de la solución propuesta

La calidad del *software* es uno de los principales problemas que intervienen en el éxito de los proyectos, es por ello que el proceso de pruebas es sin dudas uno de los aspectos fundamentales para medir la calidad de las aplicaciones de *software*. Este proceso se realiza en las fases terminales antes de entregar el *software* para su explotación. “Esta fase añade valor al producto que se maneja: todos los programas tienen errores y la fase de pruebas los descubre; ese es el valor que añade. El objetivo específico de la fase de pruebas es encontrar cuantos más errores, mejor “.

Existen diversos mecanismos para llevar a cabo las pruebas al software. En el proyecto *SGF* se aplican las pruebas de caja negra por lo que este es el mismo proceso a aplicar en el módulo Administrador de Tareas.

4.4.1. Pruebas de Caja Negra

Las pruebas de caja negra se centran en los requisitos funcionales, es decir intentan encontrar casos en los que el *software* no cumple con sus funcionalidades. El probador se limita a introducirle datos como entrada y estudiar la salida, sin preocuparse lo que se esté haciendo por dentro.

4.4.1.1. Resumen de las pruebas de Caja Negra

Las descripciones de los casos de pruebas desarrollados se pueden encontrar en el artefacto “Diseño de Casos de Prueba”. A continuación se muestra un resumen de los resultados obtenidos luego de desarrollar 3 iteraciones, donde se detectaron una serie de no conformidades que al culminar la etapa de pruebas fueron solucionadas, generándose el acta de de liberación por el grupo de calidad del centro CEGEL de la Facultad 15 la cual se muestra en el Anexo 2.

Caso de Prueba	Secciones del Caso de Prueba	Escenarios del Caso de Prueba	Resultados Obtenidos
1. Crear Tarea.	1.1. Crear Tarea.	1.1.1 Escoger un tipo de tarea a crear.	Satisfactorio.
		1.1.2. No escoger ningún tipo de tarea y dar siguiente.	Satisfactorio.
		1.1.3. Cancelar “Crear Tarea”.	Satisfactorio.
	1.2. Crear Tarea de tipo Notificación.	1.2.1. Entrar los datos correctamente.	Satisfactorio.
		1.2.2. Entrar los datos incorrectamente.	Insatisfactorio.
		1.2.3. Seleccionar Ir Atrás.	Insatisfactorio.
		1.2.4. Seleccionar Cancelar.	Satisfactorio.
	1.3. Crear Tarea de tipo Ejecución de Archivos.	1.3.1. Entrar los datos correctamente.	Satisfactorio.
		1.3.2. Entrar los datos incorrectamente.	Satisfactorio.
		1.3.3. Seleccionar Ir Atrás.	Insatisfactorio.
		1.3.4. Seleccionar Cancelar.	Satisfactorio.
	1.4. Crear Tarea de tipo Limpieza a la BD.	1.4.1. Entrar los datos correctamente.	Satisfactorio.
		1.4.2. Entrar los datos	Satisfactorio.

		incorrectamente.	
		1.4.3. Seleccionar Ir Atrás.	Insatisfactorio.
		1.4.4. Seleccionar Cancelar.	Satisfactorio.
5. Buscar Tarea.	2.1. Buscar Tarea.	2.1.1. Seleccionar criterio de búsqueda y dar buscar.	Satisfactorio.
		2.1.2. No seleccionar criterio de búsqueda y dar buscar.	Satisfactorio.
		2.1.3. Cancelar.	Satisfactorio.
6. Gestionar Tarea.	3.1. Gestionar Tarea.	3.1.1. Seleccionar Editar Tarea.	Satisfactorio.
		3.1.2. Seleccionar Cancelar.	Satisfactorio.
	3.2. Detalles.	3.2.1. Seleccionar Detalles.	Satisfactorio.
		3.2.2. Seleccionar Cancelar.	Satisfactorio.
		3.2.3. Seleccionar Ir Atrás.	Insatisfactorio.
	3.3. Modificar.	3.3.1. Seleccionar Modificar.	Satisfactorio.
		3.3.4. Seleccionar Cancelar.	Satisfactorio.
		3.2.3. Seleccionar Ir Atrás.	Insatisfactorio.
	3.4. Eliminar.	3.4.1. Seleccionar Eliminar.	Satisfactorio.
7. Gestionar Notificaciones.	4.1. Gestionar Notificaciones.	4.1.1. Seleccionar el icono Gestionar Notificaciones.	Satisfactorio.
	4.2. Mostrar	4.2.1. Seleccionar Mostrar.	Satisfactorio.
	4.3. Eliminar	4.2.2. Seleccionar Eliminar.	Satisfactorio.
	4.4. No Eliminar	4.2.3. Cancelar el Eliminar.	Satisfactorio.
8. Mostrar Estado de los Procesos.	5.1. Mostrar Estado de los Procesos.	5.1.1. Seleccionar ícono de estado de los procesos.	Satisfactorio.
9. Monitorear Sistema	6.1. Monitorear Sistema.	6.1.1. Seleccionar las tareas ejecutadas.	Satisfactorio.
		6.1.2. Seleccionar las tareas no ejecutadas.	Satisfactorio.
		6.1.3. Seleccionar las tareas en ejecución.	Satisfactorio.
	6.2. Cambiar Estado del Servicio.	6.2.1. Iniciar Servicio de Chequeo.	Satisfactorio.
		6.2.2. Detener Servicio de Chequeo.	Satisfactorio.
	6.3. Configurar Sistema	6.3.1. Entrar los Datos correctamente.	Satisfactorio.
		6.3.2. Entrar los Datos	Satisfactorio.

		incorrectamente.	
	6.4. Configurar cuenta de correo.	6.4.1. Entrar los Datos correctamente.	Satisfactorio.
		6.4.2. Entrar los Datos incorrectamente.	Satisfactorio.
	6.5. Cancelar	6.5.1. Cancelar.	Satisfactorio.

Tabla 4.10. Resumen de las pruebas de Caja Negra aplicadas

9.1.1.1. Resumen de las No Conformidades

Caso de Prueba	Escenario	No Conformidad	Significativo	Estado de la No Conformidad
1. Crear Tarea.	1.2. Crear tarea de tipo Notificación.	El sistema al entrar un correo incorrecto no muestra ningún error.	X	Solucionada (14/6/2010)
	Crear tarea de tipo (Notificación, Ejecución de Archivos, Limpieza a la BD)	Al seleccionar Ir Atrás el sistema no va a la página anterior.	X	Solucionada (14/6/2010)
3. Gestionar Tarea.	3.2. Detalles	Al seleccionar Ir Atrás el sistema no muestra la página anterior.	X	Solucionada (15/6/2010)
	3.3. Modificar	Al seleccionar Ir Atrás el sistema no muestra la página anterior.	X	Solucionada (15/6/2010)

Tabla 4.11. Resumen de las No Conformidades

En los artefactos generados se encontraron algunas no conformidades, las cuales fueron solucionadas satisfactoriamente.

9.2. Conclusiones del capítulo

En el presente capítulo se han definido los diagramas de despliegue y componentes, además se han definido los casos de prueba y el método de prueba de prueba de caja negra, lo cual permitió garantizar la correcta funcionalidad del sistema.

CONCLUSIONES GENERALES

A partir del desarrollo del módulo Administrador de Tareas se arriba a las siguientes conclusiones:

- Se realizó el modelo del dominio del sistema el cual permitió relacionar todos los conceptos del mismo.
- Se especificaron los requisitos funcionales y no funcionales con los que debía cumplir el sistema a desarrollar.
- Se realizó el diseño del sistema obteniéndose un grupo de artefactos que se utilizaron durante la implementación de la solución.
- Se implementó y evaluó el módulo dándole cumplimiento al objetivo planteado.

RECOMENDACIONES

- Integrar el módulo Administrador de Tareas al resto de los módulos de *SGF*.
- Estudiar la posibilidad de aplicar el sistema a otros proyectos con características similares ya que la solución es fácilmente integrable con otras aplicaciones.

CITAS BIBLIOGRÁFICAS

1. Definición.org. [Online] [Cited: 12 11, 2009.] http://www.definicion.org/diccionario/62.

2. **Magaña, Carlos Leopoldo.** ¿Qué es un Cron Job? [Online] 2007. [Cited: 12 11, 2009.] http://techtastico.com/post/ques-es-un-cron-job.

3. **Cifuentes, F.** Elementos físicos y lógicos del computador. [Online] 2008. [Cited: 12 20, 2009.] http://www.slideshare.net/ares_egeo/elementos-fsicos-y-lgicos-del-computador-presentation.

4. **Torre, Anibal de la.** Lenguajes del Lado del Servidor o Cliente. [Online] [Cited: 12 22, 2009.] http://www.adelat.org/media/docum/nuke_publico/lenguajes_del_lado_servidor_o_cliente.html.

5. **Duque, Raúl González.** *Python para todos.*

6. **Fowler, Martin and Sccott, Kendall.** *UML Gota a Gota.* 1999.

7. Visual Parading para UML. . *Visual-Parading.com.* [Online] 2001. [Cited: 1 20, 2010.] http://www.google.com/http://www.visual-paradigm.com/product.

8. Eclipse.org. . [Online] The Eclipse Foundation, 2010. [Cited: 12 23, 2009.] http://www.eclipse.org.

9. **Javier Eguíluz Pérez.** liblosweb.es. *Introducción a Ajax.* [Online] [Cited: 1 22, 2010.] http://www.librosweb.es/ajax/capitulo1.html.

10. **Team, Prototype Core.** prototypejs.org. [Online] 2006 - 2007. [Cited: 1 22, 2010.] http://prototypejs.org/.

11. **Potencier, Fabien and Zaninotto, Francois.** *Symfony la guía definitiva.*

12. Algo sobre Python… . *Centro de Estudiantes de Ingeniería de Sistemas.* [Online] [Cited: 1 20, 2010.] http://ceisuss.wordpress.com/2008/11/04/algo-sobre-python.

13. SoftwareLibre.ec. *Qt.* [Online] [Cited: 1 21, 2010.] http://www.softwarelibre.ec/site/index.php?option=com_content&view=article&id=419%3Aqt&catid=40%3Aides&Itemid=165.

14. AprenderPython.com. [Online] [Cited: 1 21, 2010.] http://aprenderpython.com/smf/index.php?topic=19.0.

15. **Mora, Maurice Eyssautier de la.** *Metodología de la investigación: desarrollo de la inteligencia, 5 edición (en español) .* s.l. : s.l.: Cengage Learning Editores, 2006.

16. **JACOBSON, Ivar, BOOCH, Grady and RUMBAUGH, James.** *El Proceso Unificado de Desarrollo de Software.* 2000.

17. Tecnología y Synergix. *Modelo de Dominio.* [Online] [Cited: 1 26, 2010.] http://synergix.wordpress.com/2008/07/10/modelo-de-dominio/.

18. **Pressman, Roger S.** *Ingeniería de Software. Un enfoque práctico. .* La Habana : s.n., 2005.

BIBLIOGRAFÍA CONSULTADA

- **Booch, G., Rumbaugh, J. y Jacobson, I. 2000.** *El Lenguaje Unificado de Modelado.* 2000.
- **Pressman, Roger S.** *Ingeniería de Software. Un enfoque práctico.* La Habana : s.n., 2005.
- **Martin Fowler, Kendall Sccott.** *UML Gota a Gota .* 1999.
- **Larman, Craig.** *UML y Patrones. Introducción al análisis y diseño orientado a objetos.*
- Visual Parading para *UML*. *Visual-Parading.com.* [En línea] 2007. http://www.google.com/http://www.visual-paradigm.com/product.
- PATRONES GRASP. [El línea] 2006. http://sophia.javeriana.edu.co/~lcdiaz/ADOO2006-3/grasp_cpaternostro-lvargas-jviafara.pdf.
- desarrolloweb.com. *PHP a fondo.* [En línea] http://www.desarrolloweb.com/php/.
- **Marzal, Andrés y García, Isabel.** *Introducción a la Programación con Python.* 2003.
- **Duque, Raúl González.** *Python para todos.*
- Choosing a Python IDE . [En línea] http://blog.showmedo.com/python-showmedos/choosing-a-python-ide-which-ides-need-covering/.
- Sitio oficial en Argentina de PythonCard. “About PythonCard”. [En línea] http://python.org.ar/pyar/PythonCard.
- **Fabien Potencier, Francois Zaninotto.** *Symfony la guía definitiva*.
- **Maestros del Web.** *¿Qué es Javascript?.* [En línea] http://www.maestrosdelweb.com/editorial/%C2%BFque-es-javascript/
- **Pérez, Javier Eguíluz.** *Introducción a Ajax*. *liblosweb.es.* [En línea] http://www.librosweb.es/ajax/capitulo1.html.
- Etapa: Pruebas. [En línea] 2008. http://lsi.ugr.es/~arroyo/inndoc/doc/pruebas/pruebas_d.php

GLOSARIO DE TÉRMINOS

API: Una *API* representa una interfaz de comunicación entre componentes de *software*. Se trata del conjunto de llamadas a ciertas bibliotecas que ofrecen acceso a ciertos servicios desde los procesos y representa un método para conseguir abstracción en la programación, generalmente entre los niveles o capas inferiores y los superiores del *software*.

CMS: Un Sistema de Gestión de Contenidos, es un programa que permite crear una estructura de soporte (*framework*) para la creación y administración de contenidos, principalmente en páginas web, por parte de los participantes. Consiste en una interfaz que controla una o varias bases de datos donde se aloja el contenido del sitio. El sistema permite manejar de manera independiente el contenido y el diseño. Así, es posible manejar el contenido y darle en cualquier momento un diseño distinto al sitio sin tener que darle formato al contenido de nuevo, además de permitir la fácil y controlada publicación en el sitio a varios editores.

DOM**:** *Document Object Model* (una traducción al español no literal, pero apropiada, podría ser Modelo en Objetos para la representación de Documentos), abreviado *DOM*, es esencialmente una interfaz de programación de aplicaciones que proporciona un conjunto estándar de objetos para representar documentos *HTML* y *XML*.

FRAMEWORK: Es un marco de trabajo definido en el cual otro *software* puede ser construido, puede incluir soporte de programas, bibliotecas, entre otros *software* para ayudar a desarrollar o unir componentes de un proyecto. También se le suele llamar plataforma.

GPL: Es una licencia creada por la *Free Software Foundation*, está orientada principalmente a proteger la libre distribución, modificación y uso de *software*. Su propósito es declarar que el *software* cubierto por esta licencia es *software* libre y protegerlo de intentos de apropiación que restrinjan esas libertades a los usuarios.

GUI *(*interfaz gráfica de usuario*)*: es un programa informático que actúa de interfaz de usuario, utilizando un conjunto de imágenes y objetos gráficos para representar la información y acciones disponibles en la interfaz.

IDE: es un entorno de programación que ha sido empaquetado como un programa de aplicación, es decir, consiste en un editor de código, un compilador, un depurador y un constructor de interfaz gráfica (*GUI*). Los *IDEs* pueden ser aplicaciones por sí solas o pueden ser parte de aplicaciones existentes.

J2EE (*Java 2 Enterprise Edition*): es una plataforma de programación, parte de la Plataforma *Java*, para desarrollar y ejecutar *software* de aplicaciones en Lenguaje de programación *Java* con arquitectura de *N* niveles distribuida, basándose ampliamente en componentes de *software* modulares ejecutándose sobre un servidor de aplicaciones.

PLUGIN: es una aplicación que se relaciona con otra para aportarle una función nueva y generalmente muy específica.

SCRIPT: es un conjunto de instrucciones. Permiten la automatización de tareas, creando pequeñas utilidades. Son ejecutados por un intérprete de línea de órdenes y usualmente son archivos de texto.

SCRIPT CGI*:* En español Interfaz común de puerta de enlace. No es en realidad un lenguaje o un protocolo. Es solo es un conjunto de variables y convenciones, nombradas comúnmente, para pasar información en ambos sentidos entre el servidor y el cliente. En sí, es un método para la transmisión de información hacia un compilador instalado en el servidor. Su función principal es la de añadir una mayor interacción a los documentos web que por medio del *HTML* se presentan de forma estática.

TICs: Las tecnologías de la información y la comunicación son un conjunto de servicios, redes, *software* y dispositivos que tienen como fin la mejora de la calidad de vida de las personas dentro de un entorno, y que se integran a un sistema de información interconectado y complementario.

XMLHttpRequest: es una interfaz empleada para realizar peticiones *HTTP* y *HTTPS* a servidores Web. Para los datos transferidos se usa cualquier codificación basada en texto, incluyendo: texto plano, *XML, JSON, HTML* y codificaciones particulares específicas. La interfaz se presenta como una clase de la que una aplicación cliente puede generar tantas instancias como necesite para manejar el diálogo con el servidor.

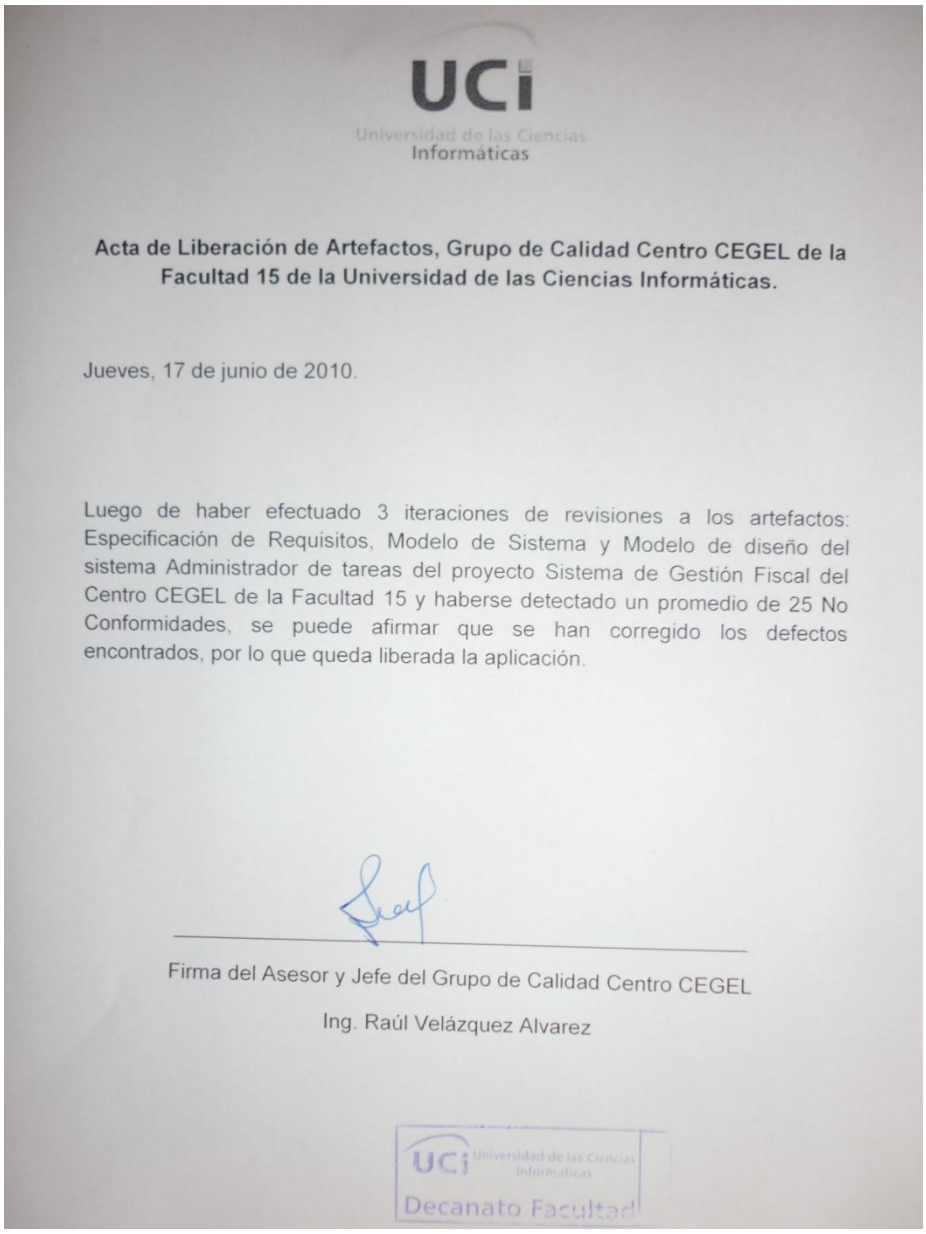
ANEXOS

Anexo 1. Clasificaciones de los procesos para la ejecución de las tareas de Control, Seguimiento y Aviso.

PROCESO SUMARIO					
No.	**Nombre del Aviso**	**Tiempo disponible para el proceso**	**Aviso a**	**Mensaje**	**Tiempo de Chequeo**
1.- ATESTADOS CON DETENIDOS					
1.1	Atestado con Detenido fuera de término.	24 horas a partir de creado el atestado en el sistema.	Fiscal Actuante	"El Atestado No.___ con detenido se encuentra fuera de término en la Fiscalía"	100% de la fecha
			Fiscal Jefe Municipio	"El Atestado No.___ con detenido que corresponde al Fiscal _____ se encuentra fuera de término en la Fiscalía"	
2.- ATESTADO SIN DETENIDOS					
2.1	Atestado sin Detenido fuera de término.	72 horas a partir de creado el atestado en el sistema.	Fiscal Actuante	"El Atestado No.___ sin detenido se encuentra fuera de término en la Fiscalía"	100% de la fecha
			Fiscal Jefe Municipio	"El Atestado No.___ sin detenido que corresponde al Fiscal _____ se encuentra fuera de término en la Fiscalía"	
3.- ATESTADO DEVUELTO POR EL TRIBUNAL					
3.1	Atestado devuelto del tribunal fuera de término.	48 horas a partir de creado el atestado en el sistema devuelto por el tribunal.	Fiscal Actuante	"El Atestado No.___ devuelto del tribunal se encuentra fuera de término en la Fiscalía"	100% de la fecha
			Fiscal Jefe	"El Atestado No.___	

			Municipio	devuelto por el tribunal que corresponde al Fiscal _____ se encuentra fuera de término en la Fiscalía"	

Anexo 2. Acta de Liberación de Artefactos por el Grupo de Calidad del Centro CEGEL.



UCI
Universidad de las Ciencias Informáticas

Acta de Liberación de Artefactos, Grupo de Calidad Centro CEGEL de la Facultad 15 de la Universidad de las Ciencias Informáticas.

Jueves, 17 de junio de 2010.

Luego de haber efectuado 3 iteraciones de revisiones a los artefactos: Especificación de Requisitos, Modelo de Sistema y Modelo de diseño del sistema Administrador de tareas del proyecto Sistema de Gestión Fiscal del Centro CEGEL de la Facultad 15 y haberse detectado un promedio de 25 No Conformidades, se puede afirmar que se han corregido los defectos encontrados, por lo que queda liberada la aplicación.

Firma del Asesor y Jefe del Grupo de Calidad Centro CEGEL

Ing. Raúl Velázquez Alvarez

UCI Universidad de las Ciencias Informáticas
Decanato Facultad