UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS

Facultad 9

TÍTULO: Implementación de los módulos de Facturación y Cobro del Sistema de Facturación y Cobro para la Empresa de Gas Manufacturado de Ciudad de la Habana.

TRABAJO DE DIPLOMA PARA OPTAR POR EL TÍTULO DE INGENIERO EN CIENCIAS INFORMÁTICAS

AUTOR: Pedro José Pérez González

TUTOR: Ing. Alain Sánchez Gutiérrez

Ciudad de la Habana, mayo 2010

AÑO 52 DE LA REVOLUCIÓN

Cuando estas a punto de caer, hay algo que debes hacer antes de recibir ayuda. Debes … caer por completo.

Dedicatoria

A mis padres, por toda la confianza que han tenido en mí.

A mi madre, porque se lo debo todo.

A mi padre, por entenderme en cada momento.

A mi hermano, por apoyarme en todas mis ideas.

A mis abuelos y a mi hermana, por ayudarme siempre que me hace falta.

A toda mi familia y aquellos que son como mi familia.

Agradecimientos

A mi madre por darme la oportunidad de ser quien soy.
A mi padre por todo lo que ha hecho por mí.
A mi hermano por siempre estar de acuerdo con las locas ideas.
A mi abuela, por todo el apoyo que me brindó en estos años.
A mi hermana y el resto de mis familiares que me ayudaron cuando me hizo falta.
A Ledian, Alexito y Yismay, por ser los mejores amigos.
A Katia y Lili por soportarme y ayudarme en todo este tiempo de Universidad.
A Lidia, ya que por ella me esforcé en los primeros años de estudio.
A Cesar y Yasmany, por enseñarme y mostrarme lo “lindo” de la programación.
A todos aquellos que en algún momento pensaron que no podría hacerlo, ya que me esforcé para demostrarle lo contrario.
A todas estas personas y muchas más que por un motivo u otro se me olvidó mencionar, pero pueden estar seguros que les agradezco.
A todos

Arigatou.

Pedro

Declaración de Autoría

Declaro que soy el único autor de este trabajo y autorizo a la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio.

Para que así conste firmo la presente a los 25 días del mes de Mayo del año 2010 .

_____ _____

Pedro José Pérez González Ing. Alain Sánchez Gutiérrez

Resumen

La Empresa de Gas Manufacturado de Ciudad de la Habana es la encargada de brindar a la población el servicio de gas metrado o gas por tubería y además es la encargada de facturar y cobrar el consumo de este producto. En la actualidad el sistema usado en esta empresa no cubre todas las necesidades. Para solucionar este problema se realiza un convenio con la Universidad de las Ciencias Informáticas (UCI), el cual consiste en desarrollar un software que cubra las necesidades que puedan aparecer en los procesos de facturación y cobro de la empresa.

El presente trabajo con título, Implementación de los módulos de Facturación y Cobro del Sistema de Facturación y Cobro para la Empresa de Gas Manufacturado de Ciudad de la Habana, tiene como objetivo realizar la implementación de los módulos de facturación y cobro, definiendo las principales herramientas, librerías y frameworks usados.

Palabras Claves

- SISMET.
- Gas Manufacturado.
- Sistemas de Facturación.
- Sistemas de Cobro.
- Implementación.

Índice

Índice de Figuras

Introducción

La Empresa de Gas Manufacturado de Ciudad de la Habana es la encargada de instalar, mantener y cobrar el servicio de gas por tubería o gas de la calle en los municipios de La Habana y Ciudad de la Habana. Cuenta en la actualidad con un sistema denominado SISMET, que no automatiza de forma correcta los procesos de Facturación y Cobro.

El sistema usado imposibilita que la facturación se realice usando una tarifa escalonada, lo que trae consigo una gran pérdida económica al país. Durante el proceso de carga del TPL los datos de los clientes son ordenados por el número del contrato, lo que imposibilita que los Lectores – Cobradores puedan hacer uso del dispositivo de forma eficiente. En el proceso de descarga del TPL, no se realizan todas las validaciones necesarias y es imposible adicionarle nuevos criterios de validación. El envío de datos desde las Casas Comerciales hacia la UEB Comercial se realiza de forma ineficiente, ya que se envían datos innecesarios y repetidos lo que trae consigo que los ficheros sean de gran volumen. La facturación de los clientes se realiza mediante una tarifa fija de 0.20 centavos, lo cual genera una pérdida económica significativa para la empresa y para el país.

En el proceso de cobro, la generación de reportes es de forma predefinida, por lo que es imposible realizar reportes según las necesidades de la empresa. La búsqueda de datos es lenta, superando en ocasiones los 5 minutos, el cual es considerado un tiempo excesivo. No cuenta con un sistema de seguridad confiable, ya que un usuario no autorizado puede realizar modificaciones en los datos de los clientes.

Actualmente no existe una comunicación persistente entre las Casas Comerciales y la UEB Comercial; pero debido a la creciente instalación del servicio de Internet en muchas compañías, la Empresa de Gas Manufacturado, necesita un sistema que sea flexible y que funcione correctamente, exista o no conexión en la Casa Comercial. El sistema actual no está desarrollado para funcionar en el momento con que se cuente con una aplicación y una base de datos centralizada.

Dado esta situación se plantea como **problema a resolver**: ¿Cómo renovar los procesos de Facturación y Cobro en la Empresa de Gas Manufacturado de Ciudad de la Habana? Se traza como **objeto de estudio** los procesos de facturación y cobro de la Empresa de Gas Manufacturado de Ciudad de la Habana, quedando enmarcado en el siguiente **campo de acción**: Sistemas de Facturación y Cobro.

Como **objetivo general** se define, Implementar los módulos de Facturación y Cobro del Sistema de Facturación y Cobro para la Empresa de Gas Manufacturado de Ciudad de la Habana. Para darle cumplimiento se realizaron las siguientes **tareas de la investigación**:

- Analizar los documentos referentes a los procesos de Facturación y Cobro de la Empresa de Gas Manufacturado de Ciudad de la Habana.
- Caracterizar las herramientas, librerías y framework usados para el desarrollo de aplicaciones web.
- Implementar los módulos de facturación y cobro.
- Validar los resultados obtenidos.

Como **idea a defender** se plantea que, “Con el desarrollo de los módulos de facturación y cobro del Sistema de Facturación y Cobro para la Empresa de Gas Manufacturado de Ciudad de la Habana se perfeccionarán los procesos de Facturación y Cobros de la Empresa de Gas Manufacturado de Ciudad de la Habana”.

Para garantizar la correcta solución del problema planteado se utilizaron los siguientes métodos de la investigación:

- **Analítico – Sintético**: Para llegar a conclusiones a partir de la documentación consultada para entender fenómenos relacionados con los procesos de facturación y cobro, así como el flujo de la información.
- **Histórico – lógico**: Para conocer las tendencias actuales en cuanto a la gestión de la información de los procesos de facturación y cobro, y uso de tecnologías de desarrollo.

Con la implementación de los módulos de facturación y cobro del Sistema de Facturación y Cobro para la Empresa de Gas Manufacturado de Ciudad de la Habana, se obtienen las siguientes mejoras:

- Correcta facturación según el consumo mensual de cada cliente.
- Creación de los reportes según las necesidades existentes en la empresa.
- Ordenar de forma correcta los datos de los clientes que son cargados al TPL, garantizando el uso eficiente de este dispositivo.
- Control de los datos que son manipulados por las personas que interactúan con la aplicación.

El presente trabajo de diploma está estructurado en 3 capítulos, a continuación se expondrá algunas características de los mismos.

Capítulo 1: Fundamentación Teórica: En el desarrollo de este capítulo se realiza un estudio del arte sobre los sistemas de facturación y cobro, así como una breve descripción de la situación actual de la Empresa de Gas Manufacturado, procesos que se realizan y flujo de la información.

Capítulo 2: Herramientas y Tecnologías: En este capítulo se describen las herramientas y tecnologías usadas para desarrollar los módulos de facturación y cobro; se definen los estándares de código a usar.

Capítulo 3: Solución Propuesta y Validación: En este capítulo se reflejan los principales componentes y diagramas usado en la implementación de los módulos de facturación y cobro, así como una descripción de las principales pruebas realizadas al sistema.

Capítulo 1
Fundamentación Teórica

1.1. Introducción

En el presente capítulo se realiza un estudio del arte de los sistemas de facturación y cobro, se abordan los conceptos asociados al dominio del problema, así como una descripción de las principales características de la Empresa de Gas Manufacturado de Ciudad de la Habana, así como los procesos y flujos de información que se realizan en dicha empresa.

1.2. Conceptos asociados al dominio del problema

1.2.1. Sistemas de Facturación

Los Sistemas de Facturación tienen como objetivo el registro y la facturación de todas las prestaciones que una empresa u organismo efectúen a un cliente determinado, puede ser una persona u otra empresa. Estos cuentan con un modelo estándar de cliente – empresa, el cual debe ser flexible y adaptable con el objetivo de satisfacer las necesidades de los clientes.

Estos sistemas están limitados porque son específicos para una tipo de empresa determinada. Debido a los complejos cálculos que realizan cada compañía en el proceso de facturación puede resultar complejo desarrollar un sistema genérico.

1.2.2. Sistemas de Cobro

Los sistemas de cobro están estrechamente relacionados a los sistemas de facturación, dichos sistemas son los encargados de recaudar el pago de los servicios ofrecidos por los sistemas de facturación.

A diferencia de los sistemas de facturación, los sistemas de cobro generan reportes, estados u otros datos que sean de interés para la empresa. Teniendo en cuenta estas

características, son más fáciles de adaptar a las necesidades de diferentes empresas, facilitando el trabajo con ellos.

1.3. Objeto de Estudio

1.3.1. Descripción General

A continuación se detallan los principales servicios que brinda la Empresa de Gas Manufacturado y se describen los principales procesos que dicha empresa realiza.

1.3.2. Procesos y Servicios que brinda

La Empresa de Gas Manufacturado de Ciudad de la Habana es la encargada de proveer los servicios de gas licuado a la población de Ciudad de la Habana. Mediante la realización de un contrato a un cliente (persona u otra empresa), la empresa garantiza la instalación, reparación, cobro por concepto de prestación de servicios y reparación del servicio de gas licuado o por tubería.

Entre los principales servicios que brinda la Empresa de Gas Manufacturado, está el servicio de facturación y el servicio de cobro. Muchos de los procesos realizados en estos servicios cuentan con ineficiencias que afectan a la empresa y a los clientes.

Los principales procesos identificados son:

- **Proceso de Facturación o Ventas de Gas Manufacturado**: se origina la acción después de haber realizado los contratos a los clientes, se le otorga su número de cliente y son procesados en el Sistema Automatizado de Dirección (SAD). La facturación se registra para el cliente no metrado de forma fija todos los meses y por la lectura mensual al cliente metrado.
- **Proceso de Cobro del servicio de Gas Manufacturado**: Se visita a los clientes en sus viviendas o entidades estatales para cobrar las facturaciones realizadas teniendo en cuenta las tarifas establecidas y lecturas de los metros contadores.

1.3.3. Descripción del domino del problema

La Empresa cuenta con una UEB Comercial, donde se realiza el proceso de facturación y cobro mediante el uso de SISMET, el cual presenta fallas. Conectado a esta, hay varias Casas Comerciales, que son las casas rectoras a nivel de municipio, encargada de llevar el control y cobro de las facturaciones de las lecturas de los metros contadores de cada núcleo familiar.

Al finalizar cada mes, las Casas Comerciales rectoras en cada municipio, le envían los datos de las facturas realizadas a la UEB Comercial, para que esta entonces actualice y registre dicha información. La información se trasmite a través del correo, proceso extremadamente lento porque las conexiones entre las casas municipales y la rectora son a través de un modem, en caso de no poder enviarse la información, esta es llevada de forma personal a la UEB Comercial mediante un dispositivo de almacenamiento (memoria flash, CD, disquete $3^{1/2}$) . Para el trabajo con dicha información se usa el TPL, que lleva un lector cobrador, donde recoge y almacena las lecturas de cada metro contador de gas de cada núcleo familiar.

1.3.4. Proceso de Facturación

El proceso de Facturación en la Empresa de Gas Manufacturado se inicia cuando el Lector – Cobrador se reporta a la Tramitadora para que esta le entregue el listado de clientes a los cuales tiene que efectuar la lectura de los Metros – Contadores. La Tramitadora selecciona los clientes teniendo en cuenta 2 criterios de selección (Lector–Libro o Ruta-Folio). Después de seleccionados los clientes la Tramitadora exporta los datos para el TPL, el cual es entregado al Lector-Cobrador junto con el Modelo de Incidencias.

El Lector – Cobrador procede a realizar la lectura de los Metros – Contadores. Al llegar a la residencia de los clientes, el Lector anota la lectura que aparece en el dispositivo y la clave de lectura correspondiente; en caso de cometer algún error en la escritura de los datos, registra la incidencia en el Modelo de Incidencia, señalando el error y el dato

correcto. Al terminar de realizar la lectura a todos los clientes, procede a la Casa Comercial para la descarga de los datos.

En la Casa Comercial entrega a la Tramitadora el TPL junto al Modelo de Incidencias. La Tramitadora descarga los datos de las lecturas del TPL y procede a la corrección automática de errores, después corrige de forma manual los errores reflejados en el Modelo de Incidencias. Al terminar el día, la Tramitadora exporta hacia un fichero todas las lecturas procesadas, el cual es enviado hacia la UEB Comercial por correo electrónico o mediante un dispositivo (memoria flash, CD, disquete $3^{1/2}$).

Al recibir las lecturas en la UEB Comercial, el Analista del Sistema carga los datos en la base de datos del sistema. Después de cargado los datos el Analista del Sistema realiza la selección de los clientes a facturar, teniendo en cuenta los criterios de facturación. El sistema factura a cada cliente seleccionado teniendo en cuenta la clave de facturación asociada a cada cliente y la tarifa asociada a la clave; al terminar el sistema muestra un resumen con los datos de la facturación de cada cliente, dando la posibilidad al Analista del Sistema de cambiar los datos de forma manual en caso de que se produzca algún error en el cálculo del importe a pagar por el cliente.

Al terminar el proceso de facturación de todos los clientes, el Analista del Sistema revisa la facturación de los clientes para cerciorarse de que la factura es correcta. Al finalizar se imprimen los Talonarios y las Chequeras. Se envían los datos de la facturación a cada Casa Comercial, se actualiza la base de datos con la información del cobro de los clientes y los reportes generados en el proceso de cobro.

1.3.5. Proceso de Cobro

El proceso de cobro se inicia en las Casas Comerciales, al final de cada mes al recibir los datos desde la UEB Comercial. Este proceso se basa fundamentalmente en cobrar el servicio de gas a los clientes. Al recibir los datos de la factura, las chequeras y los talonarios en la Casa Comercial; estos son entregados al Lector – Cobrador, el que debe ir a cada núcleo familiar y entregarle el talonario con la notificación de pago; el

cliente puede pagarle el importe al Lector – Cobrador en el momento de la notificación o puede dirigirse posteriormente a su Casa Comercial correspondiente a realizar el pago.

Al dirigirse el cliente a la Casa Comercial a realizar el pago, la Tramitadora busca los datos del cliente, según su número de contrato; comprueba que no tenga deudas de meses anteriores, después comprueba el importe a pagar por el cliente contra la notificación de pago y registra el pago del cliente. En caso de que el cliente tenga varios meses de atraso, se le notifica que no puede realizar el pago del mes actual hasta que realice el pago de los meses que debe.

Al final del mes, la Tramitadora tiene que generar una serie de informes los cuales son de suma importancia para el cierre de mes. Esta acción es la más importante en el proceso de cobro, debido a que mide las ganancias de la Casa Comercial en el mes, así como el trabajo realizado por los Lectores – Cobradores en el mes para cuadrarle el pago del salario mensual.
A continuación se muestran el listado de reportes generados.

- Recibo de Cobro.
- Cuentas por Cobrar.
- Devolución de Efectivo.
- Reporte de Lector – Cobrador.
- Reporte Certificación de Deuda.

Estos reportes deben ser dinámicos, debido a que los datos mostrados varían en dependencia de las necesidades de la empresa; lo que es actualmente imposible de configurar debido a la imposibilidad de corregir el código fuente de la aplicación actual. Estos reportes son generados en formato PDF, porque pueden ser enviados hacia la UEB Comercial o ser impresos.

El autor de esta investigación considera que estos procesos pueden ser automatizados de forma eficiente utilizando algunos de los reportadores existentes que cumplan las condiciones de configuración. La mayor imposibilidad de usar estos programas es

debido a que todos son propietarios, por lo que no se recomienda su uso debido a que hay que pagar por los derechos de uso. Se recomienda realizar un análisis de los servicios que ofrecen los sistemas mencionados, buscando las ventajas y servicios con que pueda contar el sistema a desarrollar.

1.4. Sistemas de Gestión

Las Tecnologías de la Información y las Comunicaciones (TIC) tienen un papel fundamental en la automatización de procesos industriales, las cuales cobran cada día un papel más importante al aumentar el uso de estas. Debido a esto conceptos tales como: Automatización Totalmente Integrada (TIA), Sistemas de Control Distribuido, Sistemas de Planificación de Recursos (ERP), entre otros, constituyen conceptos que las empresas modernas no pueden obviar debido a su eficiencia.

1.4.1. Sistemas ERP

Los Sistemas ERP son aplicaciones que facilitan el flujo de la información entre las funciones de manufactura, logística, finanzas y recursos humanos de una empresa.

Características principales de los Sistemas ERP: *(Maturana, 2009)*

- Cuentan con una Base de Datos centralizada.
- Los componentes interactúan entre sí consolidando las operaciones.
- Los datos son ingresados una sola vez, deben ser consistentes, completos y comunes.
- Están formados por módulos.
- Ofrecen aplicaciones especializadas para determinadas industrias.

Objetivos principales de un Sistema ERP: *(Maturana, 2009)*

- **Integración**: Los Sistemas ERP se encargan de integrar todos los procesos de una empresa, definiéndola como una serie de áreas que se relacionan entre sí, lo que permite una mayor eficiencia, reducción de costes y tiempo.

- **Modularidad**: Cada área funcional de una empresa se corresponde con un módulo del Sistema, estos aunque son independientes comparten información entre sí mediante una Base de Datos Centralizada; lo que facilita la adaptabilidad y la integración.
- **Adaptabilidad**: Debido a la modularidad e integración de un Sistema ERP, estos son fácilmente adaptables a las necesidades de una empresa, permitiendo una total configuración de estos.

1.5. Soluciones existentes en el mundo y Cuba

A raíz del análisis de los principales procesos que se realizan en la Empresa de Gas Manufacturado de Ciudad de la Habana y la arquitectura con que cuenta esta, se enuncian diferentes sistemas que implementan los procesos de facturación y cobro, con el objetivo de sintetizar características similares que puedan ser utilizadas en el desarrollo del sistema que se propone.

1.5.1. CommTrack

El Sistema de Facturación CommTrack fue diseñado para la industria de las Telecomunicaciones en el año 1997. Está adecuado para proveer de servicios a distintos tipos de compañías de las telecomunicaciones. Cuenta con un sistema para atender a los clientes, recibir pagos, procesar datos recibidos de múltiples portadores (por ej.: celulares, servicios locales, larga distancia, localizadores, servicios de internet, etc.), producir una factura consolidada e imprimir esa factura en formato para papel o electrónico. ***(CommTrack)***

1.5.2. ZFactura

ZFactura es un proyecto creado para empresas que no necesitan de una gestión comercial completa, sino varios componentes independientes. Este sistema realiza la facturación de forma eficiente, permite gestionar productos, clientes, proveedores, gastos, facturas de gastos e ingresos. Cuenta con un editor de informes, donde se pueden configurar las facturas al gusto de los clientes o la empresa. Entre sus funciones

más útiles destaca la posibilidad de obtener informes sobre los gastos e ingresos, así como calcular informes básicos sobre el listado de clientes, productos, facturas, etc. ***(Rivero Varona, 2009)***

1.5.3. OpenBravo

Openbravo es un reconocido desarrollador de soluciones profesionales en software libre para empresas, que ofrece la primera alternativa real al software propietario para empresas. Cuenta con un entorno web de gestión integral de empresas (ERP) y de gestión de punto de venta (PoS). El modelo de negocio de la compañía basado en el software libre comercial, elimina el coste de las licencias y ofrece soporte, servicios y mejoras de los productos mediante una suscripción anual. ***(Openbravo, 2010)***

Es un sistema de gestión profesional basado en software libre, que ofrece un conjunto de ventajas tales como:

- **Extensa cobertura funcional**: Contabilidad integrada, Ventas y Compras, Almacén, Producción y Gestión de Proyectos.
- **Diseñado para Internet**: Acceso seguro a todas las funcionalidades mediante un navegador web, con la posibilidad de integrarse a otras funcionalidades mediante servicios web.
- **Fácil de adaptar**: Cuenta con una arquitectura basada en modelos, que permite añadir nuevas funcionalidades para adaptarlo a las necesidades particulares de cada empresa.
- **Implantación flexible**: Ya sea en un servidor con Windows o Linux, con una o múltiples instancias, existe una opción disponible para cada necesidad.

1.5.4. SISMET

SISMET es el Sistema Metrado usado en la Empresa de Gas Manufacturado de Ciudad de la Habana, el cual realiza la facturación y cobro a los clientes metrados y no metrados. La aplicación cuenta con varias incoherencias en los procesos, la forma de almacenamiento de la información en la Base de Datos no es óptima, ya que hay gran

cantidad de datos replicados. No automatiza todas las actividades de la empresa, por lo que el proceso de facturación y cobro se demora varios días.

1.6. Sistemas web

Se entiende por sistema web o aplicación web, a un conjunto de páginas web (estáticas o dinámicas), entiéndase por página web estática a aquella en que es mostrada siempre al cliente de la misma forma sin importar las veces que sea llamada y por página web dinámica a la página cuyo contenido cambia en dependencia de los permisos de acceso que tenga el usuario que la solicita, de la cantidad de veces que sea llamada por el usuario, etc. ***(Connalen, 1999)***.

1.6.1. Características de los Sistemas Web

Las características principales de una aplicación web son: ***(Sánchez, et al., 2007)***

- Se encuentra alojada en un Servidor Web.
- Son accesibles mediante el internet, usando un navegador web.
- La lógica del sistema se ejecuta en el servidor, mientras que el cliente solamente representa los datos.
- El acceso al sistema puede ser público o restringido.
- La actualización del sistema no afecta ni depende del cliente.
- Los sistemas son multiplataforma, ya que pueden ejecutarse en cualquier Sistema Operativo que tenga un servidor web o un navegador web.

1.6.2. Ventajas de los Sistemas Web

Los sistemas web ofrecen numerosas ventajas que pueden ser utilizadas al máximo para darle solución a los problemas existentes en la Empresa de Gas Manufacturado, los cuales son: ***(Sánchez, et al., 2007)***

- **Compatibilidad Multiplataforma:** Los sistemas web funcionan de forma independiente a la plataforma que esté usando el cliente. Se cuenta con soporte para la mayoría de los sistemas operativos.

- **Actualización:** La actualización de los sistemas web se realiza sin la intervención del cliente, sin necesidad de interferir en su trabajo.
- **Fácil Instalación:** Para usar un sistema web no es necesario la descarga e instalación de este, el único requerimiento que piden estos sistemas es un navegador web y una conexión a internet.
- **Usuarios Concurrentes:** Los sistemas web brindan la posibilidad de tener varios usuarios conectados al sistema, sin que ninguno interfiera en el trabajo del otro.

1.6.3. Arquitectura de los Sistemas Web

Los sistemas web en pocos años han evolucionado desde sencillos sistemas estáticos a complejos sistemas con una detallada lógica de negocio y una interfaz altamente configurable lo más parecida a las aplicaciones de escritorio. Debido a que en la actualidad, estos sistemas dan servicios a procesos de negocio de contable envergadura, estableciendo complejos requisitos de accesibilidad y tiempo de respuesta, ha sido necesario modificar y mejorar las técnicas de la arquitectura y diseño. ***(Garrido, 2004)*** .

1.6.4. Modelo Cliente – Servidor

El uso de las aplicaciones que se encuentran publicadas en la Internet, es llamado modelo Cliente – Servidor. Cuando se hace uso de cualquier servicio que se encuentre en la Internet, ya sea visitar un sitio, consultar una Base de Datos, realizar una transferencia de ficheros; se establece un proceso en el cual intervienen dos integrantes. El cliente o usuario que ejecuta una aplicación en el ordenador local, el que se denomina programa cliente. Este cliente se encarga de comunicarse con el ordenador remoto para solicitar el servicio. El ordenador remoto atiende la solicitud del cliente mediante un programa que se denomina programa servidor. Los términos Cliente – Servidor son usados para referirse a los programas que cumplen con estas características. ***(Sánchez, et al., 2007)***.

IBM define el modelo Cliente – Servidor como "...la tecnología que proporciona al usuario final el acceso transparente a las aplicaciones, datos, servicios de cómputo o

cualquier otro recurso del grupo de trabajo y/o, a través de la organización, en múltiples plataformas. El modelo soporta un medio ambiente distribuido en el cual los requerimientos de servicio hechos por estaciones de trabajo inteligentes o "clientes", resultan en un trabajo realizado por otros computadores llamados servidores". ***(Alfaro)***

Los programas usados por los clientes para realizar las peticiones al servidor, realizan dos funciones principales; son los encargados de gestionar la comunicación con el servidor para solicitar el servicio deseado y recibir los datos enviados como respuesta, además son las herramientas que muestran los datos en la pantalla y ofrecen las funcionalidades necesarias para hacer uso de las prestaciones que ofrece el servidor. ***(Sánchez, et al., 2007)***.

El modelo Cliente – Servidor (véase Figura 1) facilita la integración y comunicación entre diferentes sistemas. De esta manera se puede usar sistemas que usen máquinas con diferentes sistemas operativos o sistemas medianos o grandes ***(Alfaro)***.

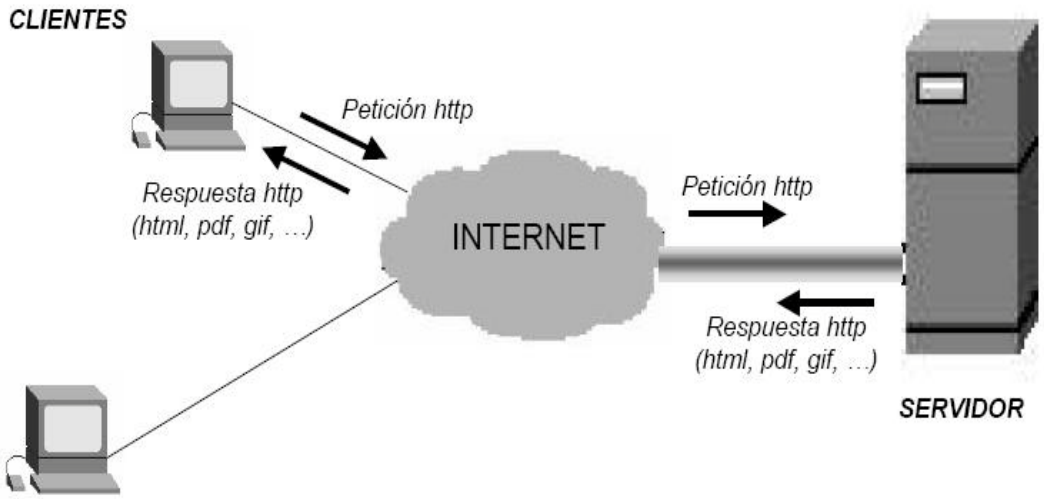

Figura 1: Modelo Cliente - Servidor

1.7. Frameworks de desarrollo web

El término de frameworks se utiliza para referirse a una estructura de software compuesta de componentes personalizables e intercambiables para el desarrollo de una aplicación. En otras palabras, un framework se puede considerar como una aplicación genérica incompleta y configurable a la que se le puede añadir las últimas piezas para construir una aplicación concreta.

Entre los principales objetivos de los frameworks de desarrollo se encuentra el de acelerar el proceso de desarrollo, reutilizar el código ya existente y promover las buenas prácticas de programación mediante el uso de patrones. Un framework de desarrollo web se define como un conjunto de componentes basado en un lenguaje web (PHP, Java, Python, Ruby, etc.), que componen un diseño reutilizable que facilita y agiliza el desarrollo web. ***(Prat Montesino, 2009)***.

1.7.1. Características de los frameworks web

Características fundamentales con que cuentan los frameworks de desarrollo web, las que permiten el amplio uso de estos y su gran aceptación para la elaboración de sistemas web. ***(Prat Montesino, 2009)***.

- **Abstracción de URL y Sesiones:** No es necesario manipular las URL ni las Sesiones, el framework lo hace de forma interna.
- **Acceso a datos:** Incluyen las herramientas y clases necesarias para interactuar con la mayoría de la Base de Datos (PostgreSQL, Oracle, MySql, MS SQL, etc.).
- **Control de peticiones:** Cuentan con clases controladoras que se encargan de atender las peticiones hechas por los clientes, estas son altamente adaptables.
- **Autenticación y Control de Acceso:** Tienen mecanismos de control de autenticación y control de acceso automático, los que se pueden adecuar a las necesidades de los clientes.

1.7.2. Ejemplos de frameworks web

Entre los frameworks más usados para el desarrollo de aplicaciones web se encuentra:

- **Symfony**
- **Zend Framework**
- **CakePHP**
- **Code Igniter**

Estos son algunos de los frameworks más usados para el desarrollo de sistemas web, aunque existen una gran variedad de estos.

1.8. Conclusiones Parciales

Luego de finalizar el capítulo se arribó a las siguientes conclusiones:

- Es importante señalar que aunque se han mencionado varios frameworks de desarrollo web, no se profundizó en las características específicas de cada uno de ellos, en caso de ser necesario la selección de alguno de los anteriormente mencionados, hay que tener en cuenta las características y requerimientos del negocio al cual se le va a desarrollar el sistema web. Mencionar además de que aunque se han descrito diferentes sistemas que se encargan de la facturación y cobro, ninguno se adecúa a las necesidades del sistema a desarrollar.
- Ninguno de los sistemas anteriormente mencionados cumple los requisitos necesarios para darle solución a los problemas existentes en la Empresa de Gas Manufacturado de Ciudad de la Habana; debido a que una de las características fundamentales de estos es que no son fácilmente configurable, por lo que no se pueden adecuar a las necesidades de la empresa. El más adecuado a darle solución a las necesidades es SISMET, al no poder adecuarse a las nuevas necesidades de la empresa no es una posible solución. Una solución a los problemas de la Empresa de Gas Manufacturado pudiera ser el Sistema ERP que se encuentra en fase de desarrollo en la Universidad de Ciencias Informáticas, al encontrarse en fase de desarrollo y prueba no se ha incluido en las soluciones mencionadas.

Capítulo 2
Herramientas y Tecnologías

2.1. Introducción

En este capítulo se realiza una descripción de las herramientas utilizadas en el desarrollo de un sistema web. Se aborda de forma general la metodología de desarrollo usada, así como las actividades fundamentales del rol implementador. Se definen los estándares de codificación usados en los leguajes especificados en el Modelo de Arquitectura del Sistema de Facturación y Cobro para la Empresa de Gas Manufacturado, tanto para el servidor como para el cliente.

2.2. Metodología de Desarrollo de Software

2.2.1. Rational Unified Process (RUP)

RUP es una metodología de desarrollo de software usada en la ingeniería de software, con el objetivo de garantizar un desarrollo eficiente y robusto; proporciona una serie de diagramas y componentes usados en el ciclo de desarrollo de un sistema, obteniendo como resultado un proceso dirigido por casos de uso, centrado en la arquitectura, iterativo e incremental.

RUP es un proceso que define "**QUIÉN**" está haciendo, "**QUÉ**" está haciendo, "**CUÁNDO**" lo hace y "**CÓMO**" lo hace, utiliza para ello el Lenguaje de Modelado Unificado (UML), para crear los esquemas del sistema de software ***(Jacobson, et al., 2004)***.

RUP es un proceso estándar y flexible, al que se le pueden realizar variaciones en dependencia de la aplicación que se desea desarrollar. Esta metodología, además de usarse en el desarrollo de software también se ha usado en otras ramas y procesos industriales. ***(Jacobson, et al., 2004)***.

2.2.1.1. Características de RUP

RUP se constituye por una serie de ciclos que forman la vida del proyecto, cada ciclo se caracteriza por: ***(Jacobson, et al., 2004)***

- **Dirigido por Casos de Uso**: Los casos de uso reflejan lo que los usuarios desean, lo cual se captura cuando se modela el negocio y se representa a través de los requerimientos. Guían el proceso de desarrollo, pues los modelos que se obtienen como resultado de los diferentes flujos de trabajo representan la realización de los casos de uso.
- **Centrado en la Arquitectura**: La arquitectura muestra la visión común del sistema, en la que el equipo de proyecto y los usuarios deben de estar de acuerdo. Describe los modelos que son más importante para su construcción, los cimientos del sistema que son necesarios como base para comprenderlo, desarrollarlo y producirlo. Las iteraciones que se realizan en RUP comienzan por los casos de uso relevantes desde el punto de vista de la arquitectura.
- **Iterativo e Incremental**: Cada fase de RUP se desarrolla en iteraciones. Una iteración involucra a todos los flujos de trabajo, aunque en cada iteración se profundiza en algunas fases más que en otras, se trabaja en todas.

A continuación se muestra una imagen donde se representa de forma gráfica todas las iteraciones y fases con que cuenta el proceso de RUP.

Figura 2: Fases e Iteración de la Metodología RUP

RUP describe la manera de implementar efectivamente “mejores prácticas” comercialmente probadas para el desarrollo de software. RUP ofrece a cada miembro del equipo las guías, plantillas y herramientas necesarias para aprovechar al máximo las siguientes mejoras: ***(Rivero Varona, 2009)***.

- Desarrollo de software iterativo.
- Administración de requerimientos.
- Uso de arquitectura basada en componentes.
- Modelado de software visual.
- Verificación en la calidad del software.
- Control de cambios de software.

2.2.1.2. Rol Implementador

El rol implementador o desarrollador, es responsable de los componentes de desarrollo y de prueba; de acuerdo a los estándares propuestos por el proyecto para la integración en subsistemas más grandes. Cuando los componentes de prueba, como controladores o fragmentos para simulación deben crearse para dar soporte a las pruebas, el implementador también es responsable del desarrollo y las pruebas de los componentes de prueba y los subsistemas correspondientes. ***(Jacobson, et al., 2004)***.

A un implementador se le puede asignar la responsabilidad de implementar una parte estructural del sistema (como un subsistema de implementación o de clases), o una parte funcional del sistema, como la ejecución de casos de uso o sus características. ***(Jacobson, et al., 2004)***.

Es habitual que una persona actúe como implementador y diseñador, desempeñando las responsabilidades de ambos roles. Es posible que dos personas actúen como implementador de un único componente del sistema, dividiendo las responsabilidades entre ellos o efectuando las tareas conjuntamente, como en un enfoque de programación en parejas. ***(Jacobson, et al., 2004)***.

El rol implementador se desarrolla en la fase de construcción en el flujo de Implementación, debido a que en esta fase es donde se construye el producto o los subsistemas de componentes necesarios para el desarrollo de un producto final.

Responsabilidades del rol implementador

- Artefactos de instalación.
- Elementos de implementación.
- Pruebas de desarrollo.
- Subsistema de implementación.
- Elemento de comprobabilidad.

Actividades que realiza el rol implementador (véase Figura 3)

- Analizar el comportamiento en tiempo de ejecución.
- Desarrollo del producto.
- Ejecutar las pruebas de desarrollo.
- Implementar las pruebas de desarrollo.
- Implementar elementos de comprobabilidad.

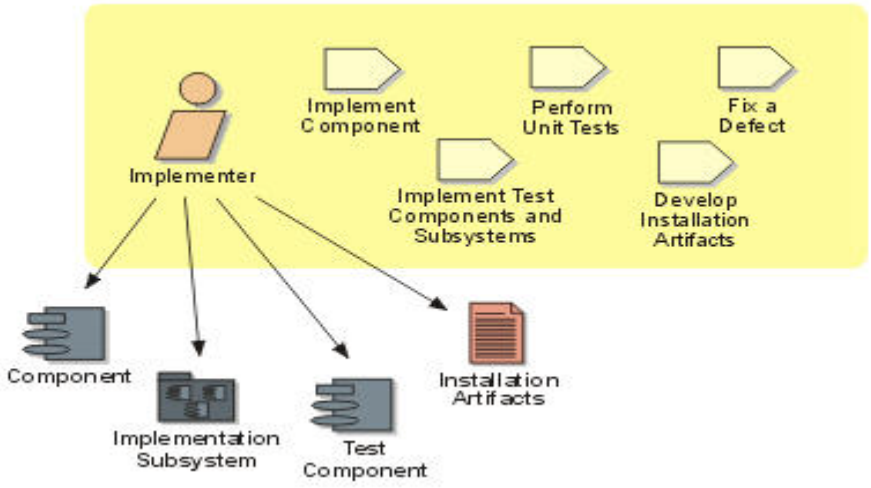

Figura 3 Rol Implementador según RUP

2.3. Tecnologías web del lado del servidor

Las tecnologías, herramientas y lenguajes que se reflejan a continuación son ejecutados y utilizados en el servidor web. A diferencia de las que se ejecutan en el lado del cliente, estas no dependen de un navegador web, sino que son interpretadas por el servidor.

En estas se procesa las peticiones de los clientes, se encuentra implementada la lógica del sistema, se valida la seguridad del sistema y la integridad de los datos enviados por los usuarios.

2.3.1. PHP

PHP es un lenguaje "Open Source[1]" interpretado de alto nivel, pensado para desarrollar sistemas web, el cual puede ser embebido en páginas HTML. La mayoría de su sintaxis es similar a C, Java y Perl, dándole la ventaja de la facilidad de aprendizaje. La meta de este lenguaje es permitir a los desarrolladores escribir páginas web dinámicas de manera fácil y rápida. ***(PHP Group, 2010)***

PHP es el heredero de PHP/FI, creado por Resmus Lerdford en 1995. Este proyecto en su fase inicial era un conjunto de script, basados en Perl para controlar el acceso online a los sitios. Debido a la gran flexibilidad de este lenguaje, basado en una estructura similar a C, muchas personas se interesaron en su uso, por lo que en 1997 su código fue liberado para todo aquel que quisiera usarlo, corregirle errores o adicionarles clases y funcionalidades.

Con el auge de la Programación Orientada a Objeto (POO), se creó en el 2002 la versión de PHP 5, está basada en las características fundamentales de la POO; se eliminó el uso de la programación estructurada, se mejoró el uso de las clases, la herencia y la redefinición de métodos. ***(php.net, 2010)***.

[1] **Open Source**: Término usado para denotar al Código Libre, la característica fundamental es que el código de una aplicación no puede ser privado, sino que todos tienen que tener acceso a ver, analizar o modificar este código.

La POO es una forma especial de programar, más cercana a cómo expresaríamos las cosas en la vida real que otros tipos de programación. Con la POO hay que aprender a pensar las cosas de una manera distinta, para escribir los programas en términos de objetos, propiedades, métodos. ***(Angel Alvarez, 2004)***.

La POO no es difícil, pero es una manera especial de pensar, a veces subjetiva de quien la programa, de manera que la forma de hacer las cosas puede ser diferente según el programador. Aunque podamos hacer los programas de formas distintas, no todas ellas son correctas, lo difícil no es programar orientado a objetos sino programar bien. Programar bien es importante porque así se pueden aprovechar todas las ventajas de la POO. ***(Angel Alvarez, 2004)***.

Con el uso de la POO en PHP 5 se resolvieron algunos de los principales problemas que existían en las versiones anteriores, tales como la clonación de objetos, que se realizaba al asignar un objeto a otra variable o al pasar un objeto por parámetro en una función. Para solventar este problema PHP 5 hace uso de los manejadores de objetos (Object handles), que son una especie de punteros que apuntan hacia los espacios en memoria donde residen los objetos. Cuando se asigna un manejador de objetos o se pasa como parámetro en una función, se duplica el propio manejador de objeto y no el objeto en sí. ***(Angel Alvarez, et al., 2007)***.

Características de PHP 5 y la POO *(Angel Alvarez, et al., 2007)*

- **Nombre fijo para los constructores y destructores:** Los nombres a usar en los constructores y destructores están definido de forma implícita y no pueden ser modificados (__contruct(), __destruct()). Estos métodos se encargan de realizar las tareas de inicialización y destrucción de los objetos.
- **Acceso public, prívate y protected a propiedades y métodos:** A partir de esta versión hay que usar los modificadores de acceso habituales a la POO. Estos métodos sirven para definir qué variables y métodos son accesibles desde cada entorno.
- **Uso de interfaces:** Las interfaces son usadas en la POO para definir un conjunto de métodos que implementa una clase. Una clase puede implementar un conjunto

de interfaces. En el desarrollo de aplicaciones es muy usado para suplir la falta de herencia múltiple en lenguajes como PHP y Java.

- **Método y clases "final":** Se pude definir que una clase o un método es "final". En caso de que un método sea definido como "final", este método no puede ser redefinido mediante la herencia. En caso de que lo definido como "final" sea una clase, esta no puede ser heredada por otra clase.
- **Operador instanceof:** Operador usado para saber si un objeto es instancia de una clase determinada.
- **Atributos y métodos static:** Se pueden definir métodos y atributos static, estos son las propiedades y funcionalidades a las que se puede acceder mediante el nombre de la clase, sin necesidad de realizar una instancia de la clase.
- **Clase y métodos abstractos:** Es posible crear clases y métodos abstractos. Las clases abstractas no se pueden instanciar, estas se usan para heredarlas desde otras clases que pueden o no ser abstractas. Los métodos abstractos no se pueden llamar, se utilizan para ser heredados por otras clases las que los redefinen según las necesidades propias de las clases.
- **Constantes de clases:** Se pueden definir constantes dentro de la clase, luego estas pueden ser accedidas a través de la propia clase.
- **Funciones que especifican la clase que reciben por parámetro:** Se puede definir funciones donde se especifique el tipo de objeto que se le pasa como parámetro. En caso de no ser de este tipo se produce un error.
- **Función __autoload:** Es habitual que en el desarrollo de un sistema se escriba un archivo por cada clase que se define, como técnica para organizar el código de las aplicaciones, pero debido a esto a veces es muy tedioso realizar la inclusión de cada clase. La función __autoload(), sirve para incluir el código de una clase que todavía no haya sido incluida en el código que se está ejecutando.
- **Clonado de objetos:** Se puede crear un objeto a través de la copia exacta de otro objeto, utilizando la instrucción "clone". También se puede definir el método __clone() para realizar tareas asociadas con la clonación de un objeto.

2.3.2. Symfony 1.4.x

Symfony es un framework usado para desarrollar sistemas web dinámicos basados en PHP 5. Este garantiza una rápida creación, mantenimiento y actualización de un sitio web. Debido a que solo necesita una cantidad mínima de requisitos para su instalación, hace que este framework sea uno de los más usados en el desarrollo de aplicaciones web.

El código usado para su implementación está escrito de forma natural, lo que garantiza que cualquier persona con conocimientos básicos de PHP, HTML y Javascript pueda desarrollar una aplicación web. Está pensado para usar cualquier metodología de desarrollo de software, ya que se adapta tanto a RUP, Extreme Programming [2](XP), Keep It Simple Stupid [3](KISS), Do Not Repeat Yourself [4](DRY).

Las aplicaciones creadas con Symfony cuentan con 3 ambientes de desarrollo: desarrollo (**dev**), producción (**prod**) y prueba (**test**), de esta forma se puede tener una versión del sistema en desarrollo y otro en producción, lo que facilita las actualizaciones y la corrección de errores. ***(Sensio Labs, 2010)***.

2.3.2.1. Patrón Modelo – Vista – Controlador (MVC)

Symfony está basado en el patrón clásico de diseño web conocido como arquitectura MVC, formado por 3 niveles:

- El modelo representa la información con que trabaja la aplicación, la lógica del negocio.
- La vista transforma el modelo en una página web que permite al usuario interactuar con ella.
- El controlador se encarga de procesar las iteraciones del usuario y realizar los cambios necesarios en el modelo o la vista, también controla la lógica del sistema.

[2] **Extreme Programming**: Programación Extrema, metodología ágil muy usada en el desarrollo de aplicaciones con un corto periodo de desarrollo.
[3] **Keep It Simple Stupid**: Mantén las cosas simple estúpido, metodología usada en aplicaciones donde es importante mantener el código simple y legible.
[4] **Do Not Repeat Yourself**: No te repitas a ti mismo, metodología usada en el desarrollo de aplicaciones, que dada sus características es posible la reutilización de código.

La arquitectura MVC (véase Figura 4) separa la lógica del negocio (el modelo) y la representación de la información (la vista), por lo que se consigue un mantenimiento más sencillo de las aplicaciones. En caso de que una misma página deba mostrarse en un navegador estándar o en un navegador de un dispositivo móvil, solamente es necesario crear la vista de cada dispositivo, ya que la lógica del sistema y del negocio es la misma independientemente del navegador.

El controlador se encarga de aislar al modelo y a la vista de los detalles del protocolo utilizado por la petición (HTTP[5], Consola de Comandos, Correo Electrónico, etc.). El modelo se encarga de la abstracción de la lógica relacionada con los datos, haciendo que la vista y los datos sea independiente del tipo de gestor de base de datos utilizado en la aplicación. ***(Potencier, et al., 2009)***.

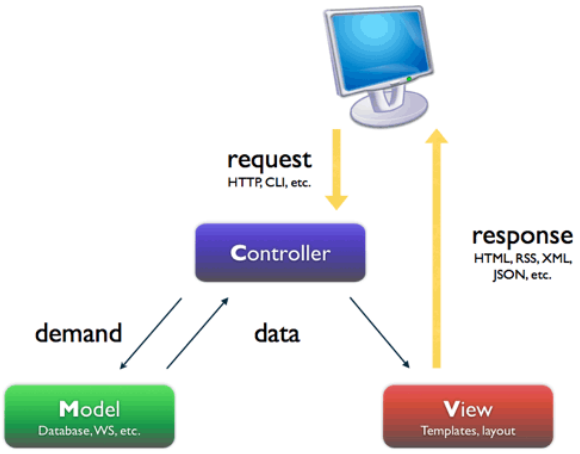

Figura 4 Patrón MVC

2.3.2.2. Mapeo de Objetos a Base de Datos (ORM)

Las bases de datos siguen una estructura relacional, por el contrario PHP 5 y Symfony usan una estructura orientada a objeto. Debido a esto, para acceder a los datos como si

[5] **HTTP**: Hipertext Transfer Protocol o Protocolo de Transferencia de Hipertexto, protocolo usado en la comunicación de los sistemas web mediante el uso de la Internet.

la base de datos fuera orientada a objeto, es necesaria una interfaz que traduzca la lógica de los objetos a la lógica relacional. Esta interfaz se denomina “Mapeo de Objeto a Base de Datos” (ORM, de sus siglas en inglés “Object – Relational Mapping”).

Un ORM consiste en una serie de objetos que permiten acceder a los datos y que contienen en su interior cierta lógica de negocio. Una de las ventajas de utilizar estas capas de abstracción de objetos/relacional es que se evita utilizar una sintaxis específica de un sistema de bases de datos concreto. Esta capa transforma automáticamente las llamadas a los objetos en consultas SQL optimizadas para el sistema gestor de bases de datos que se está utilizando en cada momento.

De esta forma, es muy sencillo cambiar a otro sistema de bases de datos completamente diferente en mitad del desarrollo de un proyecto. Estas técnicas son útiles por ejemplo cuando se debe desarrollar un prototipo rápido de una aplicación y el cliente aun no ha decidido el sistema de bases de datos que más le conviene. El prototipo se puede realizar utilizando SQLite y después se puede cambiar fácilmente a MySQL, PostgreSQL u Oracle cuando el cliente finalmente se decida. El cambio se puede realizar modificando solamente una línea en un archivo de configuración.

La capa de abstracción utilizada encapsula toda la lógica de los datos. El resto de la aplicación no tiene que preocuparse por las consultas SQL y el código SQL que se encarga del acceso a la base de datos es fácil de encontrar. Los desarrolladores especializados en la programación con bases de datos pueden localizar fácilmente el código. ***(Potencier, et al., 2009)***.

El ORM usado en la implementación del Sistema de Facturación y Cobro para la Empresa de Gas Manufacturado es Propel. Propel es un servicio de objetos persistentes y consulta que provee un sistema de almacenar objetos, realizar búsquedas y restaurar objetos desde una base de datos sin escribir ningún código SQL.

Propel puede ser descrito como un mapeo objeto-relacional o una capa de objetos persistentes. Propel es desarrollado por el proyecto Torque y se encuentra optimizado para PHP. Cuenta con dos componentes principales:

- Un motor generador para construir sus clases y archivos SQL. (Propel-Generator).
- Un ambiente de ejecución que proporciona herramientas para construir consultas SQL, ejecutando consultas compiladas y herramientas para el manejo de conexiones para múltiples bases de datos. (Propel).

Propel proporciona una capa de abstracción y encapsulación de base de datos y reglas lógicas del negocio. Las clases de Propel representan la capa de modelo del modelo MVC, diseñada para encapsular cualquier nivel de validación de datos necesitados en el negocio de un sistema. El siguiente diagrama muestra la relación entre Propel, Creole y las Base de Datos subyacentes. ***(Lellelid, 2005)***

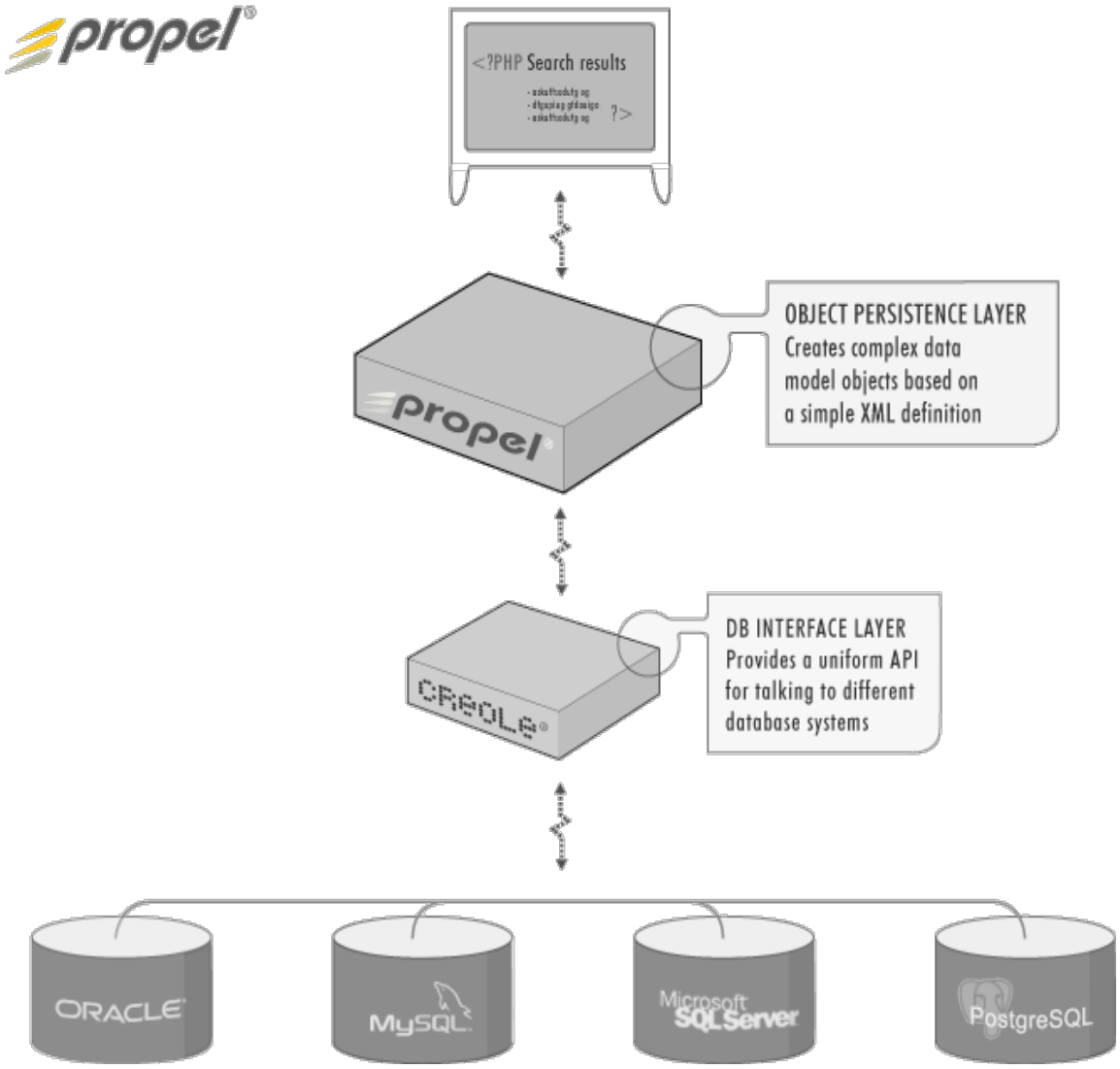

Figura 5 Propel

En Propel las clases de entrada de datos son llamadas *clasePeer*, mientras que las clases de fila de entrada de datos son llamadas *entidad* o clase *objeto*. Al integrarse con Symfony se crean dos clases adicionales, en las cuales se incluye el código referente al proceso de negocio del sistema. Teniendo en cuenta esto, una tabla ***Persona***, generaría las siguientes clases:

- **BasePersonaPeer**: Tiene los métodos estáticos para interactuar con la tabla, cuenta con el código SQL asociado a las acciones de insertar, leer, modificar y eliminar; es una clase abstracta.
- **BasePersona**: Tiene el código asociado a las acciones sobre una fila de la tabla, cuenta con el código asociado a las acciones de leer y modificar las valores de los atributos de la fila; es una clase abstracta.
- **PersonaPeer**: Hereda de la clase BasePersonaPeer, inicialmente está vacía, en esta clase se incluye el código relacionado al proceso de negocio de las acciones sobre la tabla.
- **Persona**: Hereda de la clase BasePersona, inicialmente está vacía, en esta clase se incluye el código relacionado al proceso de negocio referente a las acciones sobre una fila de la tabla.

2.3.2.3. YAML

YAML no es un lenguaje de marcado (del inglés YAML Ain't Markup Language). Es un formato para serializar datos de forma tal que es fácil procesarlos por la máquina, fácil de leer por las personas y fácil interactuar con los lenguajes de script. YAML permite escribir los datos como en XML, pero con una sintaxis mucho más sencilla, es un formato especialmente útil para describir los datos que pueden ser transformados en arreglos simples y asociativos.

YAML utiliza la tabulación para indicar su estructura, los elementos que forman una secuencia utilizan un guión medio y los pares clave/valor de los arreglos asociativos se separan con 2 puntos. YAML también dispone de una notación resumida para describir la misma estructura con menos líneas: los arreglos simples se definen entre corchetes y los arreglos asociativos se definen entre llaves.

YAML es mucho más rápido de escribir que XML (ya que no hacen falta las etiquetas de cierre y el uso continuo de las comillas) y es mucho más poderoso que los tradicionales archivos .ini (ya que estos últimos no soportan la herencia y las estructuras complejas).

Por este motivo, Symfony utiliza el formato YAML como el lenguaje preferido para almacenar su configuración. ***(Potencier, et al., 2009)***.

YAML está diseñado para ser usado de forma fácil y amigable por personas que trabajen con datos. Se usan caracteres Unicode[6] para definir la estructura de información del fichero y el otro valor representa el valor de la variable. Logran organizar de forma mínima la enorme cantidad de datos con que se trabaja en algunas aplicaciones web. Está siendo usado en las nuevas versiones de lenguajes de desarrollo rápido como PHP, Perl, Python, Ruby y Javascript.

En el desarrollo de un sistema web es necesario saber varios lenguajes de programación, pero solo es necesario uno para almacenar y transferir datos. YAML fue creado para trabajar con ficheros específicos como: ficheros de configuración, archivo de registros, lenguaje cruzado de transferencia de datos, persistencia y depuración de objetos en complejas estructuras de datos. ***(Ben-Kiki, et al., 2006)***.

2.3.2.4. Estándares de Codificación

Los estándares de código se definen como las reglas o las directivas que se usan cuando se escribe el código de un programa. Estos estándares ayudan a que otros programadores puedan entender, usar o corregir errores de un fragmento de código escrito por otro programador. Estos patrones son generalmente creados para un lenguaje de programación específico, de forma que un estándar escrito para C++ no pueda ser aplicado a Python, aunque muchas reglas puedan ser aplicadas a distintos lenguajes de programación. Symfony propone un estándar de codificación a usar en el desarrollo de sistemas web basados en este framework, de forma que se garantice que todos puedan reutilizar el código escrito por otros desarrolladores. ***(Sensio Labs, 2010)***.

Estándar de código propuesto para el Sistema de Facturación y Cobro de la Empresa de Gas Manufacturado:

[6] **Caracteres Unicode**: Caracteres definidos en una tabla donde cada caracter tiene un valor numérico. Utilizados para lograr la estandarización entre varios sistemas.

- No usar al tab para identar el código, use 2 espacios en blanco.
- No usar los espacios después de abrir y antes de cerrar los paréntesis.
- Use la notación camello o camelCase para las variables, métodos y funciones, esta notación define que los nombres empiezan por minúscula, en caso de ser un nombre compuesto el primer nombre empieza con minúscula y el segundo seguido empieza con mayúscula.
- Las llaves **{}** siempre se definen en una línea, úselas para definir el cuerpo de las estructuras de control y ordenar las expresiones definidas en el cuerpo de estas.
- Cada método de la clase o definición de miembro debe tener declarado de forma explícita la visibilidad de este, usando las palabras reservadas **public**, **prívate**, **protected**.
- No termine los ficheros **.php** con la etiqueta de cierre **?>**, esto no es necesario y puede provocar errores de difícil detección, en caso de que exista un espacio después de esta etiqueta.
- En el cuerpo de una función la declaración de retorno debe de estar separada por una línea en blanco, de esta forma se aumenta la legibilidad del código.
- Todo comentario que solamente tenga una línea, tiene que ser escrito usando la etiqueta de comentario de línea seguido por un espacio y el comentario en minúscula.
- Evite evaluar variables en conjunto con un **string**, en lugar de eso use la concatenación.
- Use las variables constantes definidas en php de forma minúscula, tales como **true**, **false**, **null**, **array**. De forma contraria cada vez que defina una constante propia declárela en mayúscula, de la siguiente forma: **define** (‘**PI’**,3.14), o defina una clase explícitamente para las constantes.
- Para comprobar si una variable es nula no use la función nativa de php **is_null**, use el operador de comparación exactamente distinto (**!==**).
- Cuando compare una variable con un **string**, ponga primero el **string** y después la variable, en caso de poderse usar el operador de exactamente igual (**===**).
- Use el nombre del tipo de dato en la declaración de los parámetros de los métodos y las funciones.

- Cada función y cada método declarado debe contar con la documentación php, donde se explique de forma clara y sencilla el funcionamiento, los tipos de datos pasados por parámetros y un ejemplo de su uso.

Ejemplo de estos estándares se pueden encontrar en el Anexo I: Ejemplo de Estándares de Codificación de Symfony.

2.3.3. Swiftmailer

Uno de los requisitos más demandados en la nueva aplicación, es la funcionalidad de envío de ficheros mediante el correo electrónico. Para darle solución a esta petición, se usa la librería Swiftmailer, la cual se encuentra incluida en el framework a partir de la versión 1.3 de Symfony.

Esta librería es una de las más usadas para el envío de correos usando el protocolo SMTP[7]. Se desarrolló sobre un proyecto de código libre, por lo que puede ser modificada según las necesidades de cada usuario, mientras no se viole lo establecido en la licencia GPL v3.0[8]. Está desarrollada en PHP 5, ya que la versión anterior de este lenguaje no contaba con soporte para el servicio de correo. ***(Corbyn, 2010)***.

Esta librería es un paquete de clases que sirven para comunicar una aplicación desarrollada en PHP con un servidor SMTP y permitir el envío de correos. Cuenta con las funcionalidades básicas para el servicio de correo, como establecer dinámicamente el remitente o el destinatario, permite el adjunto de ficheros de diferentes formatos, soporta una capacidad máxima de envío de 2 MegaByte (MB) ***(Corbyn, 2010)***.

[7] **SMTP**: Simple Mail Transfer Protocol o Protocolo Simple de Transferencia de Correo, es un protocolo de la capa de aplicación utilizado para el intercambio de mensajes entre computadoras u otro dispositivo. Es un estándar oficial de internet, definido en el RFC 2821.

[8] **GPL v3.0**: Es la última versión de la licencia de software libre GPL, la cual establece que cualquier persona es libre de copiar y distribuir el software, en caso de modificación se debe referenciar al autor original.

2.4. Tecnologías web del lado del cliente

El desarrollo de una aplicación web implica algo más que conocimiento de HTML y la utilización de un editor "Lo que ves es lo que obtienes" (WYSIWYG, del nombre en inglés What You See Is What You Get), debido a la rápida evolución de las tecnologías relacionadas con los sistemas web, las que permiten que estos sean más interactivos y dinámicos.

Las tecnologías utilizadas en el lado del cliente están insertadas en el código HTML de la página del cliente y son interpretadas y ejecutadas por el navegador web. Su funcionamiento depende del navegador y la versión utilizada por el cliente ***(Sánchez, et al., 2007)***.

2.4.1. ExtJS 1.3.x

ExtJS es un framework basado en Javascript Orientado a Objeto (JOO), uno de los más usados en el desarrollo de aplicaciones web en la parte del cliente, debido a que cuenta con una gran cantidad de componentes, clases y validaciones que pueden ser configuradas según las necesidades de cada sistema o del cliente.

Cuenta con una ayuda en inglés, la que tiene diversos ejemplos, de los cuales se pueden auxiliar los desarrolladores para realizar las aplicaciones. Tiene un diseñador de interfaces en línea, lo que facilita la creación de ventanas y componentes, garantizando un rápido desarrollo de sistemas web. ***(ExtJS development Group, 2010)***

ExtJS no es solamente otro framework Javascript, debido a que se puede integrar con otras librerías como la YUI de Yahoo, ampliado las funcionalidades. Con este framework se pueden desarrollar aplicaciones web de forma fácil y rápida debido a:

- Provee una forma fácil de uso de Document Object Model (DOM), compatible con las ventanas, formularios y otros componentes.
- Maneja de forma transparente los distintos navegadores web, facilitando y agilizando el trabajo al programador, al no tener que definir varias Hojas de Estilo (CSS) en dependencia del navegador del cliente.

- Las peticiones y la comunicación con el servidor es usando AJAX, por lo que no es necesario recargar la página, garantizando mayor velocidad en las peticiones y respuestas.
- El uso del DOM se realiza de forma transparente para el programador, debido a que ExtJS cuenta con una capa de abstracción que soporta a los navegadores más usados: Internet Explorer 6 o superior, Mozilla Firefox 2 o superior, Safari 2 o superior, Opera 9.0 o superior.

La librería ExtJS usa la notación JSON (del inglés JavaScript Object Notation) para el manejo de los datos y la configuración de los componentes, usando esta ventaja se le agregó la funcionalidad de carga en tiempo de ejecución de código javascript, donde se encuentran definidos los componentes a usar de la ExtJS; logrando que el sistema sea más rápido en el proceso de carga.

JSON es un lenguaje definido para el trabajo de objetos en JOO, cuenta con las siguientes características: ***(Frederick, y otros, 2008)***

- Los objetos se definen entre llaves **{}**, lo que simboliza que todo en el interior pertenece a la configuración del objeto, puede ser datos de configuración u otro objeto (**{records}**).
- Cada registro del objeto consiste en una par nombre/valor, separados por dos puntos, y los pares de datos separados por coma (**{name0: value0, name1: value1**}).
- Los valores de los registros pueden contener cualquier tipo de dato, ya sea un bool, un arreglo, una función u otro objeto.

2.4.1.1. Estándares de Codificación

Los ficheros javascript no cuentan con un estándar de codificación definido, debido a que al publicar una aplicación en modo de producción estos ficheros son comprimidos usando un compresor de código lo que convierte al código javascript ilegible, imposibilitando su modificación, pero acelerando la carga y ejecución de estos.

Por esta razón en los proyectos desarrollados con Symfony se cuentan con dos versiones del mismo proyecto, una en modo desarrollo y a otra en producción, con la posibilidad de realizar los cambios y correcciones en el modo desarrollo y después actualizarlos en el modo producción.

Teniendo en cuenta lo anteriormente planteado se propone el siguiente estándar de codificación a usar en los ficheros con javascript vinculados al frameworks ExtJS, basado en JOO:

- Use 4 espacios para identar el código.
- Defina las variables a usar dentro de la configuración de la propia variable.
- El comentario de línea defínalos en una sola línea comenzando por el signo de comentario seguido de un espacio y el comentario en letra minúscula.
- Las variables **string** defínalas entre comilla simple.
- Use la estructura de JOO para definir los objetos, entre llaves.
- Use la estructura de JOO para definir arreglos de objetos, entre corchetes.
- Las variables reservadas del lenguaje escríbalas en minúscula, tales como: **true**, **false**, **null**, **new**, etc.
- En caso de que tenga que escribir un comentario de varias líneas, utilice varios comentarios de línea.

2.5. Servidores

En el presente epígrafe se describen los servidores usados en el desarrollo de la aplicación web, definidos en el Modelo de Arquitectura del Sistema de Facturación y Cobro para la Empresa de Gas Manufacturado de Ciudad de la Habana.

2.5.1. Servidor Web Apache

Apache es un servidor web que se encarga de atender las solicitudes realizadas por los usuarios mediante la Internet, brinda la funcionalidad de publicar sitios en línea. El proyecto de Apache es creado y actualizado por la Apache Software Fundation (ASF). Es una organización no lucrativa, ya que distribuyen el servidor apache de forma libre

para todo aquel que desee usarlo. ASF es una comunidad descentralizada de desarrolladores que trabajan cada uno en sus propios proyectos de código abierto. Se fundó en 1999 a partir de la creación del Grupo Apache, el cual tenía como objetivo la creación de un servidor Http de código abierto. Entre los objetivos principales de la ASF está la de dar protección legal a todos los voluntarios que trabajan en "Proyectos Apaches". El Proyecto Apache es el origen de un grupo de licencias de código abierto, las cuales se denominan "Estilo Apache". ***(ASF, 2010)***

2.5.2. Servidor de Base de Datos PostgreSQL

PostgreSQL es un potente sistema de base de datos objeto ‐ relacional de código abierto. Cuenta con un historial de 15 años de activo desarrollo, en los cuales se ha ganado una gran reputación por ser confiable, correcta integración de datos y alto rendimiento. Se puede ejecutar sobra la mayoría de los Sistemas Operativos (SO) que existen en la actualidad, como Linux, Microsoft Windows, Mac OS X, Solaris, Fedora, etc. Cuenta con un completo soporte para llaves, llaves foráneas, uniones, vistas, disparadores y procedimientos almacenados. Tiene todos los tipos de datos que son usados por SQL Server 2008 y cuenta con interfaces para su comunicación en la mayoría de los lenguajes existentes como C, C++, C#, Perl, PHP, Java, Python, Ruby, ODBC, etc. ***(PostgreSQL Fundation, 2009)***

Mediante el uso de las clases de PostgreSQL se cuenta con un avanzado compendio de opciones, tales como, Control Concurrente de Multi ‐ Versión (MVCC), puntos de recuperación por tiempo, replicación asincrónica, puntos de salvamento, permite realizar respaldos de forma local o en línea, según la configuración del servidor. Cuando se activa el servidor postgres en ambiente de desarrollo avanzado se cuenta con una capacidad de almacenaje de 4 Tera Byte (TB), pero en modo de producción los datos de rendimiento y almacenaje son los siguientes:

- Capacidad máxima por tabla 32 TB
- Capacidad máxima por fila 1.6 TB.
- Capacidad máxima por campo 1 TB.
- Ilimitada cantidad de filas por tabla.

- Cantidad de columna por tabla entre 250 – 1600, en dependencia de los tipos de datos de cada columna.
- Ilimitado indexado por tabla.

2.6. Herramientas de Desarrollo

En el presente epígrafe se describen las herramientas usadas en el desarrollo de la aplicación web, definidas en el Modelo de Arquitectura del Sistema de Facturación y Cobro para la Empresa de Gas Manufacturado de Ciudad de la Habana.

2.6.1. Visual Paradigm

Visual Paradigm para UML es una herramienta UML profesional muy potente que soporta el ciclo de vida completo del desarrollo de software: análisis y diseño orientados a objetos, construcción, pruebas y despliegue. Diseñado para varios tipos de usuarios, incluyendo Ingenieros de Software, Analistas de Sistemas, Analistas de Negocio, Arquitectos de Sistema y Desarrolladores.

El software de modelado UML ayuda a una más rápida construcción de aplicaciones de calidad, mejores y a un menor coste. Permite dibujar todos los tipos de diagramas de clases, ingeniería inversa, generar código desde diagramas y generar documentación. Esta herramienta soporta UML 2.1 completo, Notación para el Modelado de Procesos de Negocio (BPMN) y permite realizar ingeniería tanto directa como inversa. ***(Visual Paradigm Design Group, 2009)***

A partir de un modelo relacional en SQL Server, MySql, etc. Es capaz de desplegar todas las clases asociadas a las tablas (siguiendo el patrón de diseño Una Clase-Una Tabla). Para gestionar la persistencia y el mapeo de estas clases con la base de datos utiliza Hibernate para Java y NHibernate en el caso de un proyecto .Net. Es colaborativa, soporta múltiples usuarios trabajando sobre el mismo proyecto; genera la documentación del proyecto automáticamente en varios formatos como web o PDF, y permite control de versiones. Es posible generar código desde Visual Paradigm para

plataformas como .NET, Java y PHP, así como obtener diagramas a partir del código, esto es de gran utilidad pues ahorra tiempo a los desarrolladores y reduce las posibilidades de cometer errores. Brinda la posibilidad de obtener una base de datos relacional y el código necesario para acceder a esta a partir de un Diagrama Entidad Relación, se puede conectar de forma fácil a diversos servidores de base de datos. Se integra con varios ambientes de desarrollo integrados (IDE) lo cual permite pasar del código al modelado y viceversa. Establece interoperabilidad con otras aplicaciones como el Visio y el Rational Rose. Disponible en múltiples lenguajes y plataformas: Microsoft Windows (98, 2000, XP, o Vista) Linux, Mac OS X, Solaris o Java. ***(Rivero Varona, 2009)***

2.6.2. Zend Studio

Zend Studio es un IDE usado para el desarrollo de aplicaciones web, tanto para lenguaje PHP como para Java. Al ser creado y mantenido por el mismo equipo que desarrolla PHP, está optimizado para la integración con este lenguaje de programación. Está diseñado para aumentar la productividad, velocidad y rendimiento en el trabajo colectivo e individual. ***(Zend GDE, 2010)***

Entre las funcionalidades que brinda está la refactorización de métodos y variables, generación dinámica de código lo que aumenta el desarrollo de las aplicaciones, cuenta con un analizador semántico, etc. En conjunto todas estas funcionalidades permiten y garantizan el trabajo eficiente, rápido y seguro de los sistemas web.

Cuenta con 2 versiones del IDE, una se encarga de la parte de desarrollo, esta se denota IDE Cliente, mientras que la otra se encarga de limpiar o analizar el código generado por el otro IDE, esta se denota IDE Servidor, esto es una gran ventaja ya que es muy difícil correr (tracear) o analizar la aplicaciones web. El IDE Servidor se encarga de perfilar, limpiar el código, inspeccionar, generar pruebas y reportes del código creado por el programador, lo que facilita la optimización del código y corrección de errores de programación.

Cuenta con la posibilidad de añadir más funcionalidades, mediante la inclusión de plugins, uno de los más usados en el desarrollo en equipo es el subclipse, el cual permite la conexión con un repositorio o control de versiones para asegurar un correcto trabajo en equipo. Cuenta con soporte y reconocimiento de la última versión de PHP y Eclipse Galileo usado para la programación en ficheros javascript, hojas de estilo y páginas clientes. Tiene un amplio completamiento de código para la web y la posibilidad de integrarles las librerías que sean usadas en el desarrollo de proyectos. ***(Zend GDE, 2010)***.

2.7. Conclusiones Parciales

Al concluir la investigación referente al presente capítulo, se arribó a las siguientes conclusiones:

- RUP es una metodología robusta que soporta a grandes sistemas, pero teniendo en cuenta las características de esta metodología al aplicárselas al sistema desarrollado, en vez de garantizar eficiencia en el proceso de desarrollo, trae demoras debido a la cantidad de documentos a generar y el tiempo de duración de cada iteración, teniendo en cuenta esto, una solución más factible seria el uso de una metodología ágil, además de que estas integran al cliente al equipo de desarrollo, lo que fue un factor clave en la demora del desarrollo del Sistema de Facturación y Cobro de la Empresa de Gas Manufacturado.
- Symfony es un framework de desarrollo robusto para sistemas pequeños, medianos y grandes, cuenta con varias funcionalidades que aceleran el proceso de implementación y validación, por lo que es correcto su uso en la implementación del Sistema de Facturación y Cobro para la Empresa de Gas Manufacturado.
- El ORM usado para la conexión con la base de datos es Propel, debido a las ventajas anteriormente mencionadas, además de integrarse con la clase Criteria, lo que facilita y agiliza la realización de consultas.
- El IDE Zend Studio para la implementación del sistema es una gran ventaja, debido a que este IDE cuenta con varias comodidades que garantizan que los desarrolladores realicen sus programas de forma rápida y óptima.

Capítulo 3
Solución Propuesta y Validación

3.1. Introducción

En este capítulo se especifica el Modelo de Implementación y los Modelos de Pruebas Unitarias y Pruebas Funcionales del Sistema de Facturación y Cobro para la Empresa de Gas Manufacturado de Ciudad de la Habana. En caso de ser necesario un mejor entendimiento de los diagramas siguientes, dirigirse al Anexo II Casos de Uso del Sistema de Facturación y Cobro.

3.2. Modelo de Implementación

El Modelo de Implementación describe cómo los elementos del Modelo de Diseño son implementados en términos de componentes. Describe cómo se organizan los componentes de acuerdo con los mecanismos de estructuración disponible en el entorno de implementación y en el lenguaje o lenguajes de programación utilizados, y la dependencia entre los componentes. ***(Garrido, 2004)***

3.2.1. Subsistema de Facturación

El Subsistema de Facturación es donde se encuentran implementados todos los Casos de Uso (CU) relacionados con el proceso de facturación de la Empresa de Gas Manufacturado de Ciudad de la Habana.

Está integrado por los siguientes CU:

- Cargar TPL
- Leer TPL
- Corregir Errores
- Exportar Datos de Lecturas
- Cargar Lecturas
- Facturar

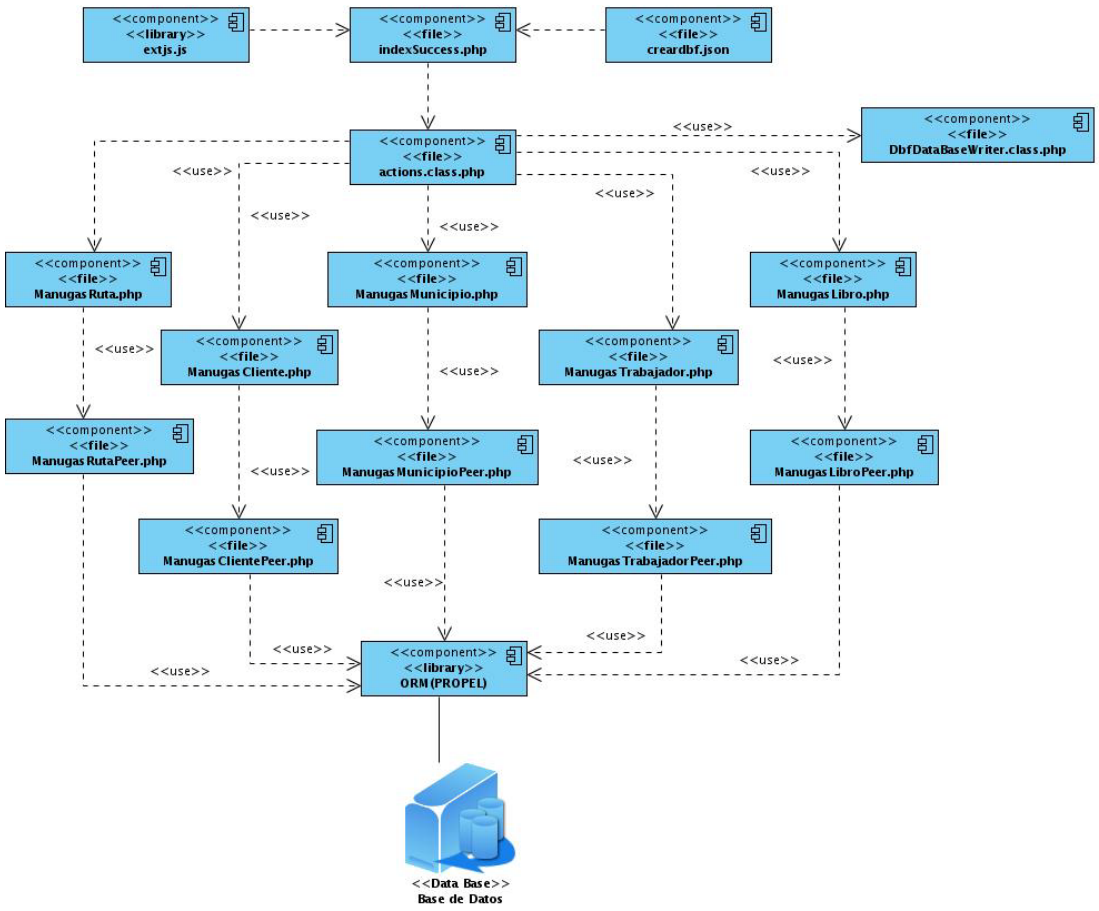

Figura 6 Diagrama de Componentes. Subsistema "Facturación". CU Cargar TPL

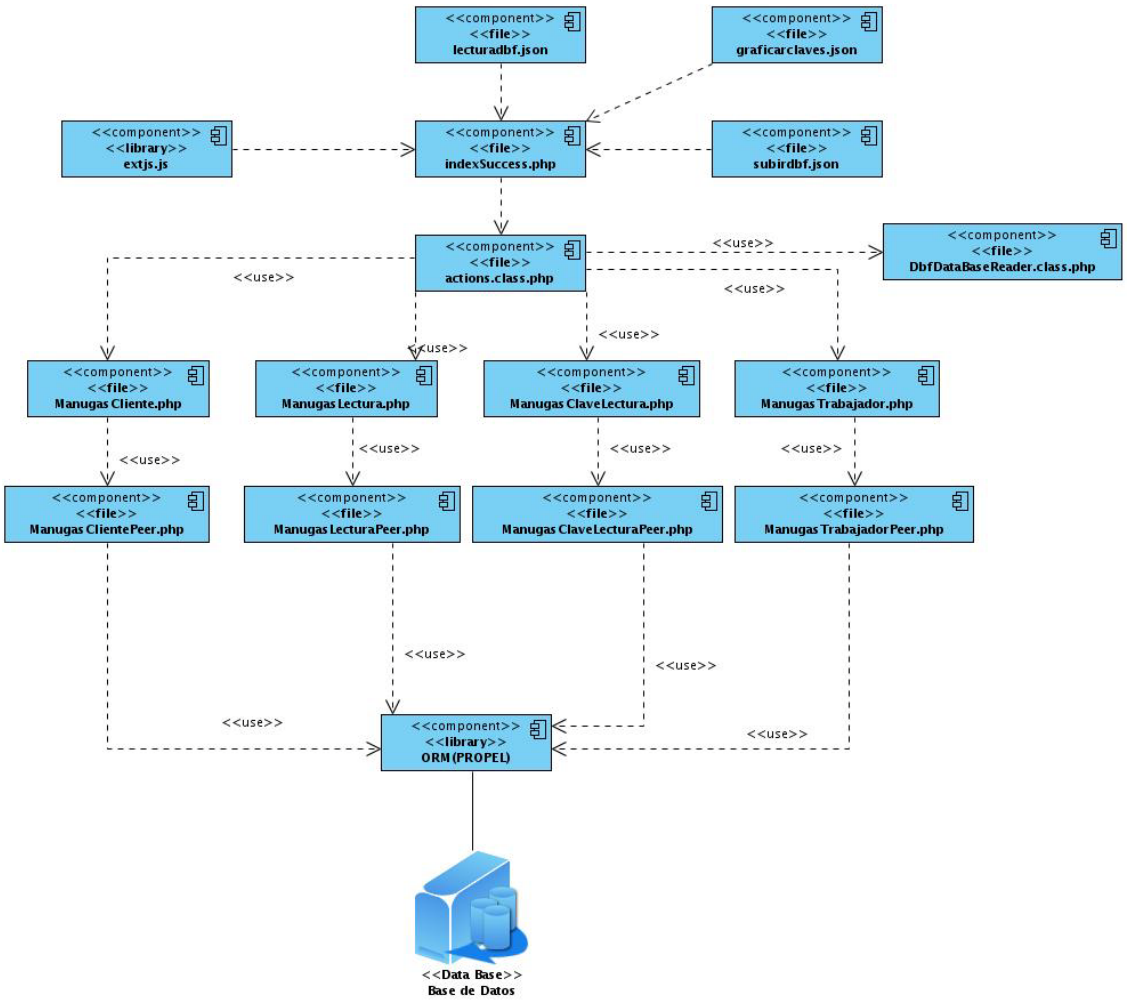

Figura 7 Diagrama de Componentes. Subsistema "Facturación" CU Leer TPL

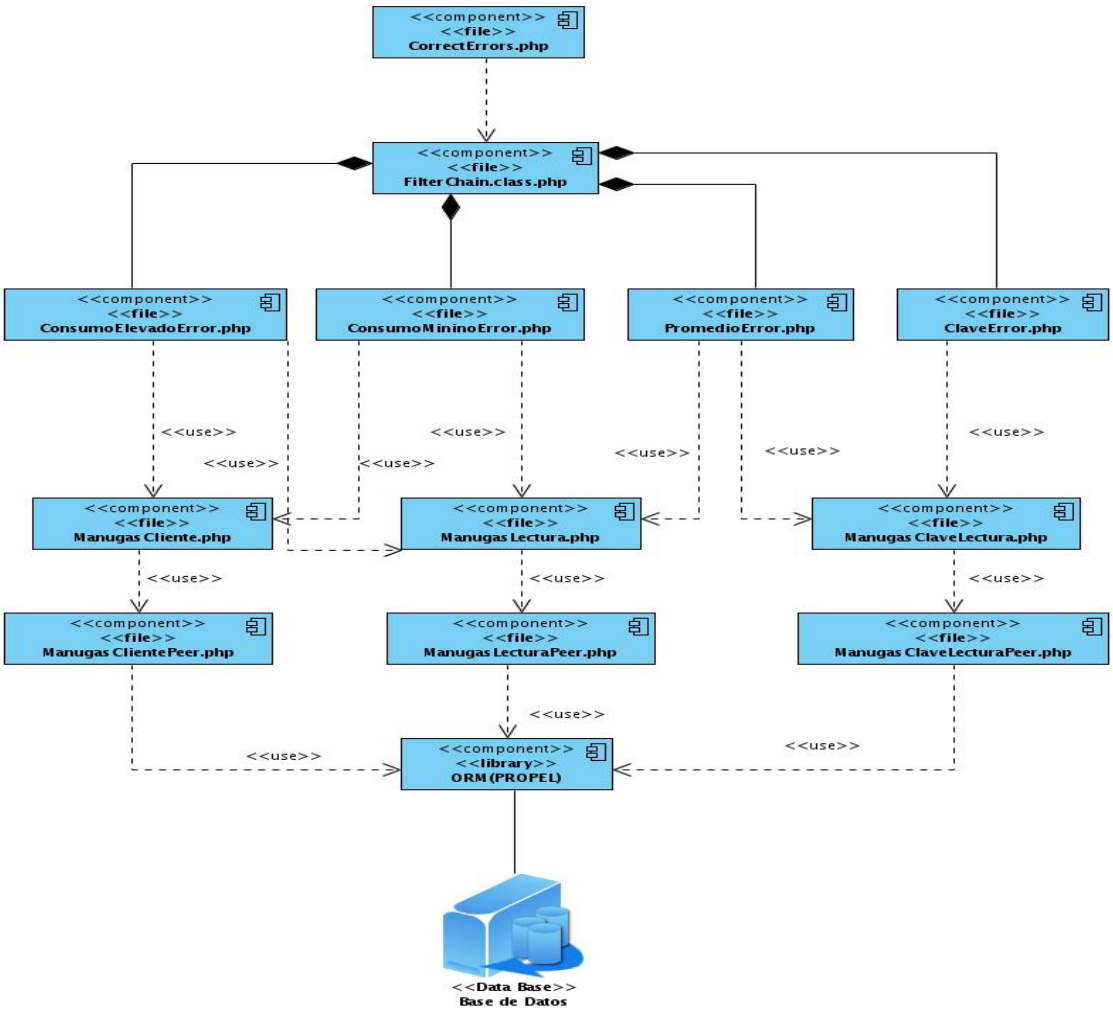

Figura 8 Diagrama de Componentes. Subsistema "Facturación" CU Corregir Errores

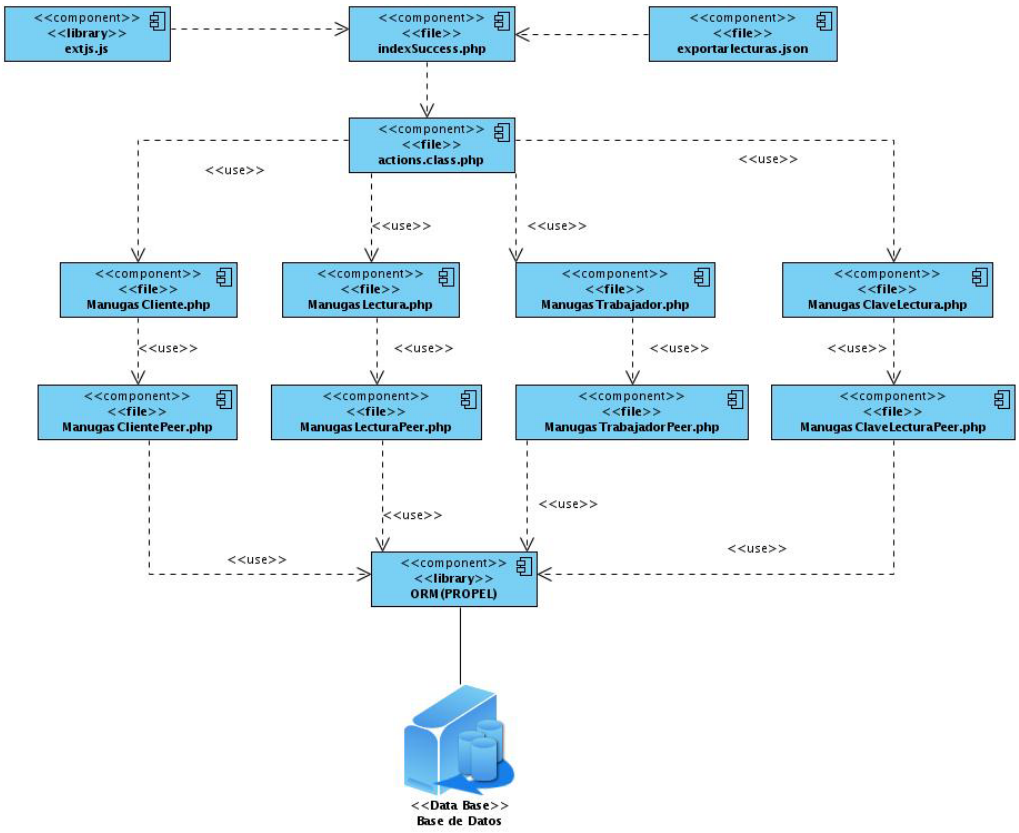

Figura 9 Diagrama de Componentes. Subsistema "Facturación" CU Exportar Datos de Lectura

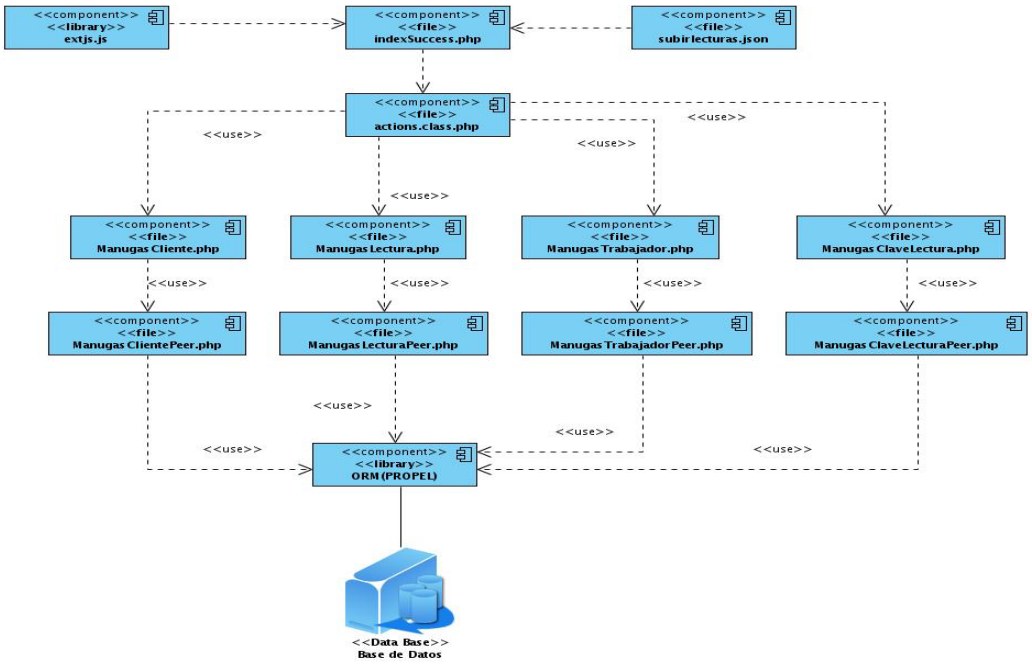

Figura 10 Diagrama de Componentes. Subsistema "Facturación" CU Cargar Lecturas

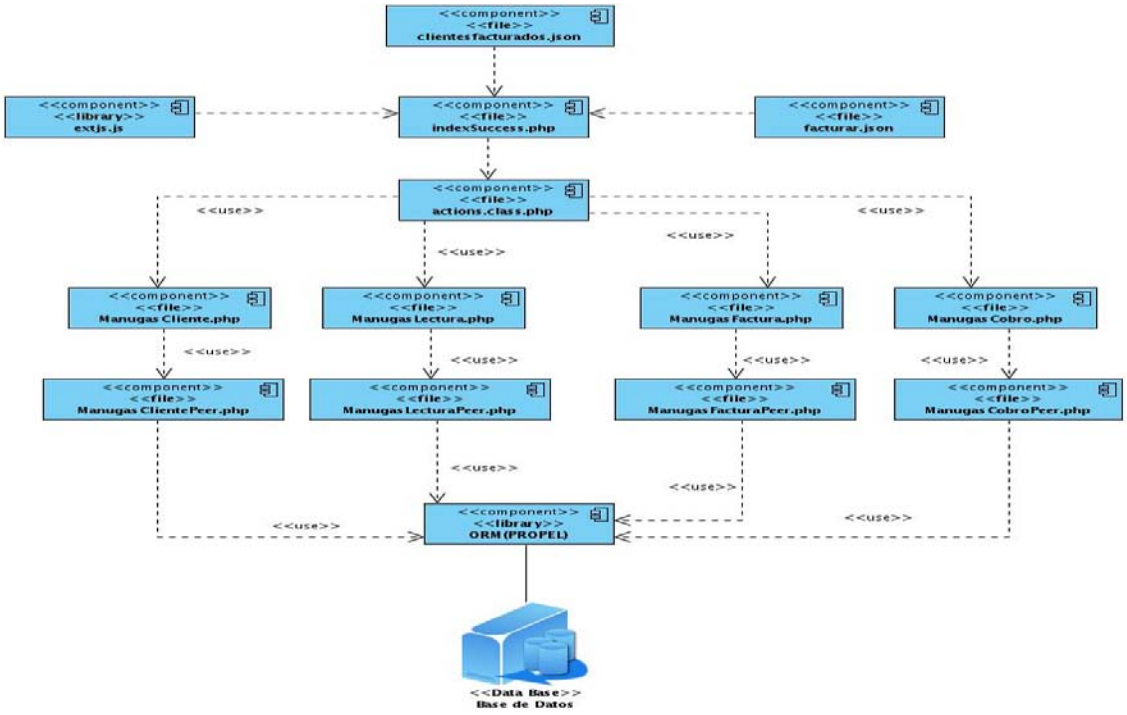

Figura 11 Diagrama de Componentes. Subsistema "Facturación" CU Facturar

3.2.2. Subsistema de Cobro

El Subsistema de Cobro es donde se encuentran implementados todos los CU relacionados con el proceso de cobro de la Empresa de Gas Manufacturado de Ciudad de la Habana.

Está integrado por los siguientes CU:

- Registrar Cobro en Casa Comercial
- Buscar Cliente
- Generar Recibo del Cobro
- Generar Cuentas por Cobrar.
- Generar Reporte de Lector – Cobrador
- Generar Reporte de Certificación de Deuda.

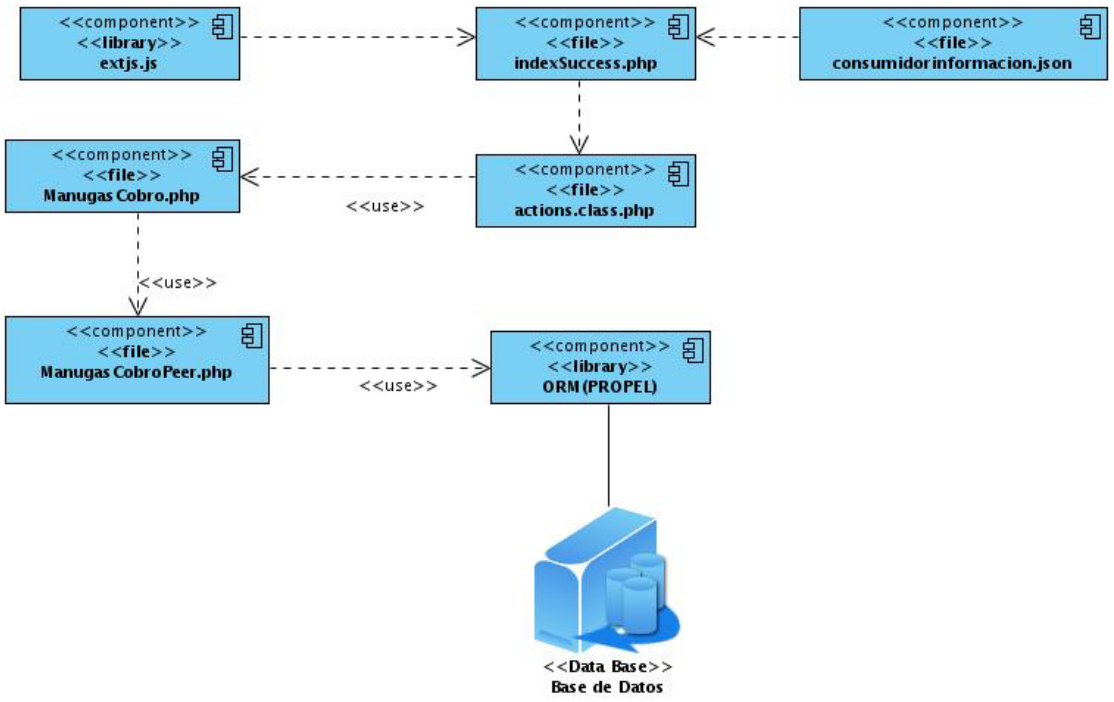

Figura 12 Diagrama de Componentes. Subsistema de "Cobro" CU Registrar Cobro en Casa Comercial

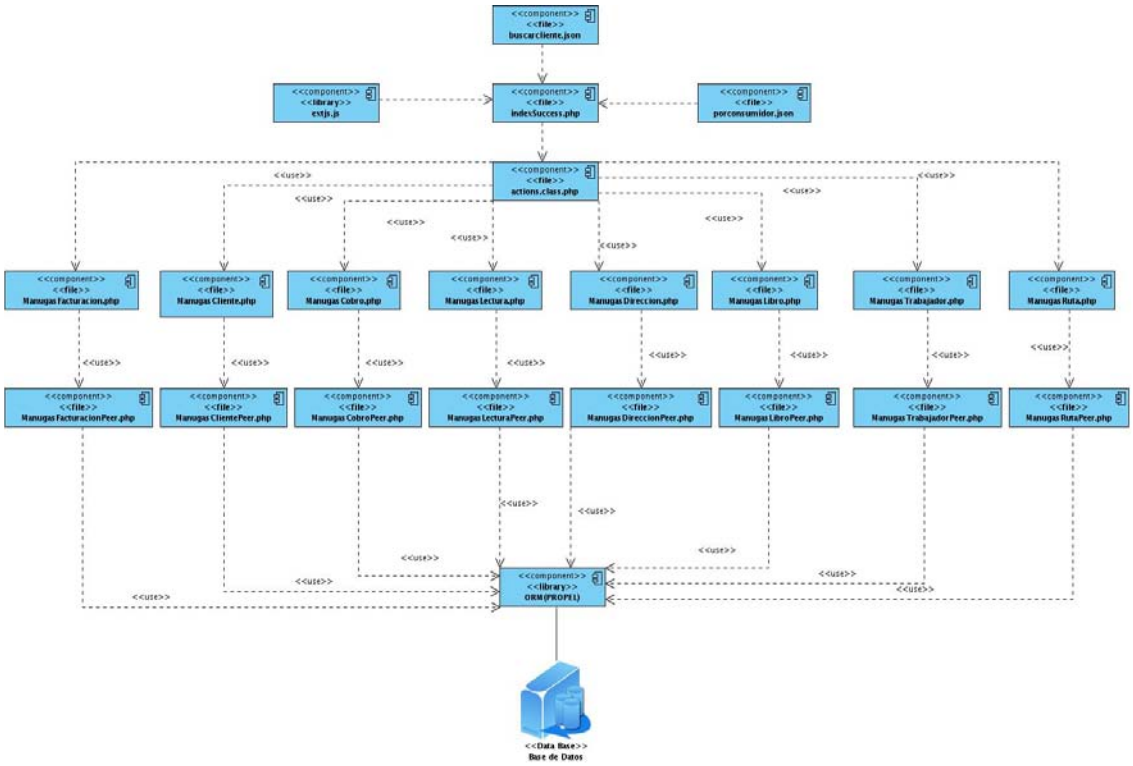

Figura 13 Diagrama de Componentes. Subsistema de "Cobro" CU Buscar Cliente

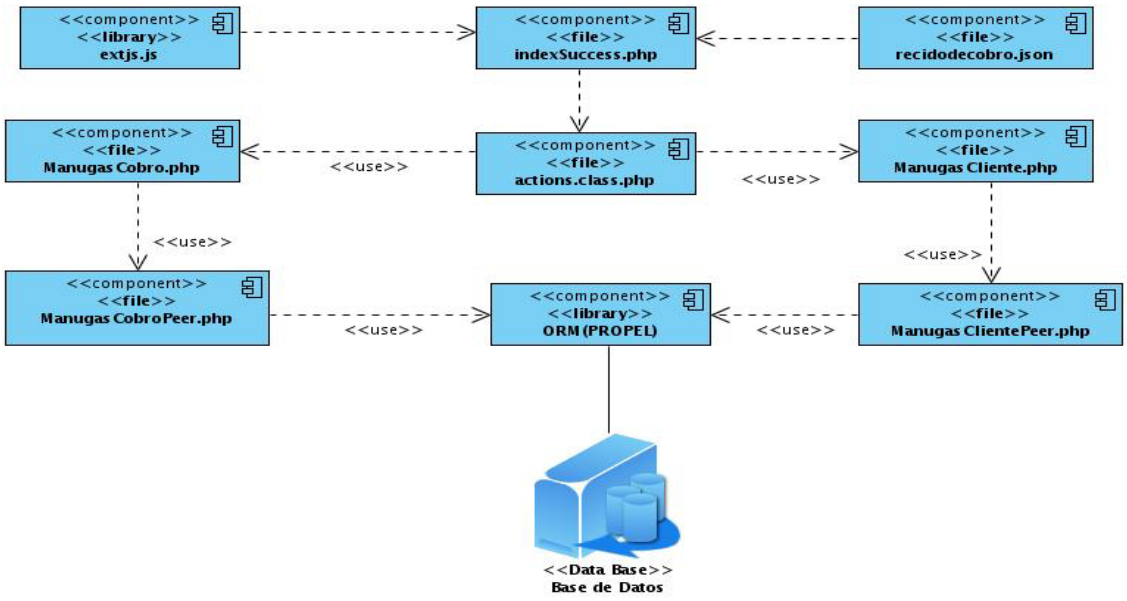

Figura 14 Diagrama de Componentes. Subsistema de "Cobro" CU Generar Recibo de Cobro

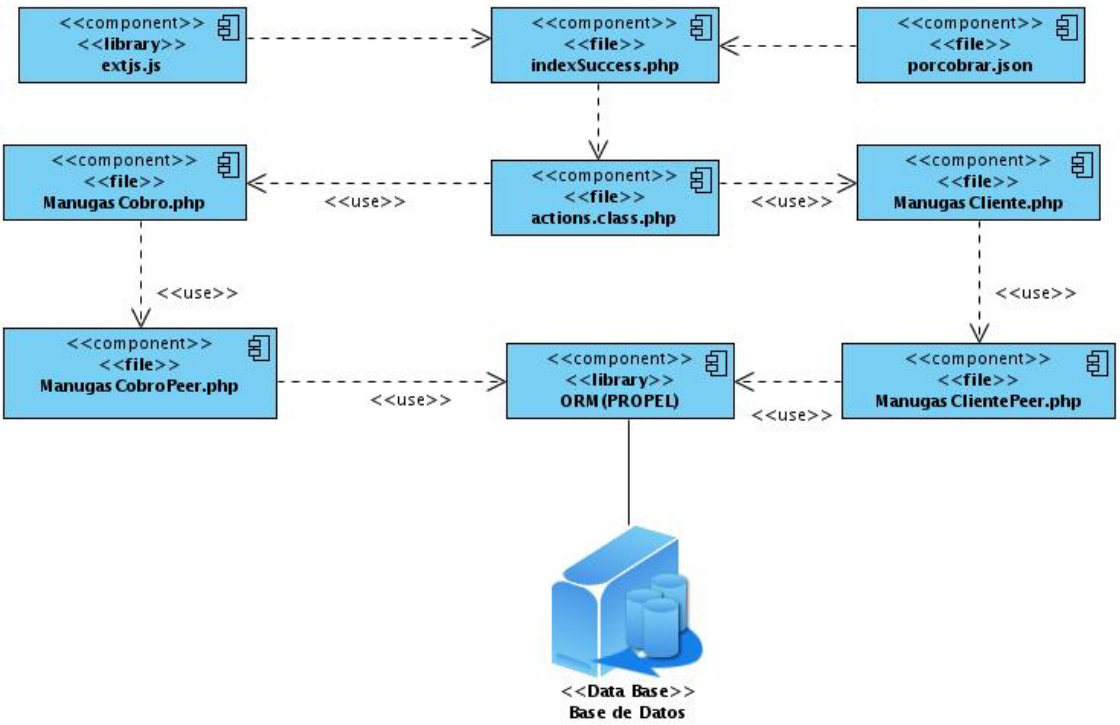

Figura 15 Diagrama de Componentes. Subsistema de "Cobro" CU Generar Cuentas por Cobrar

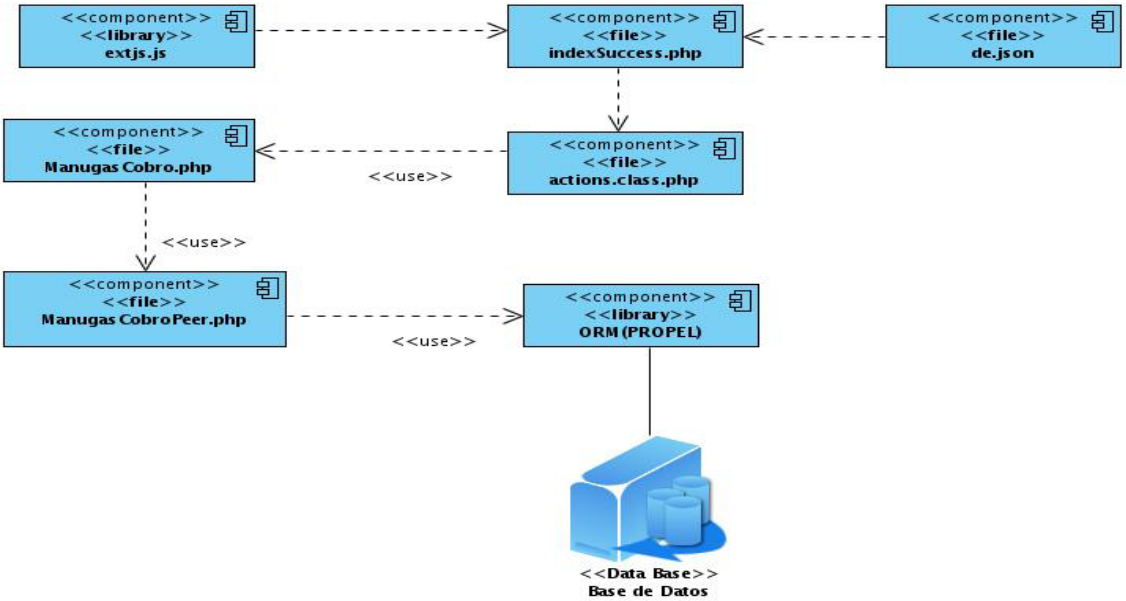

Figura 16 Diagrama de Componentes. Subsistema de "Cobro". CU Generar Devolución de Efectivo

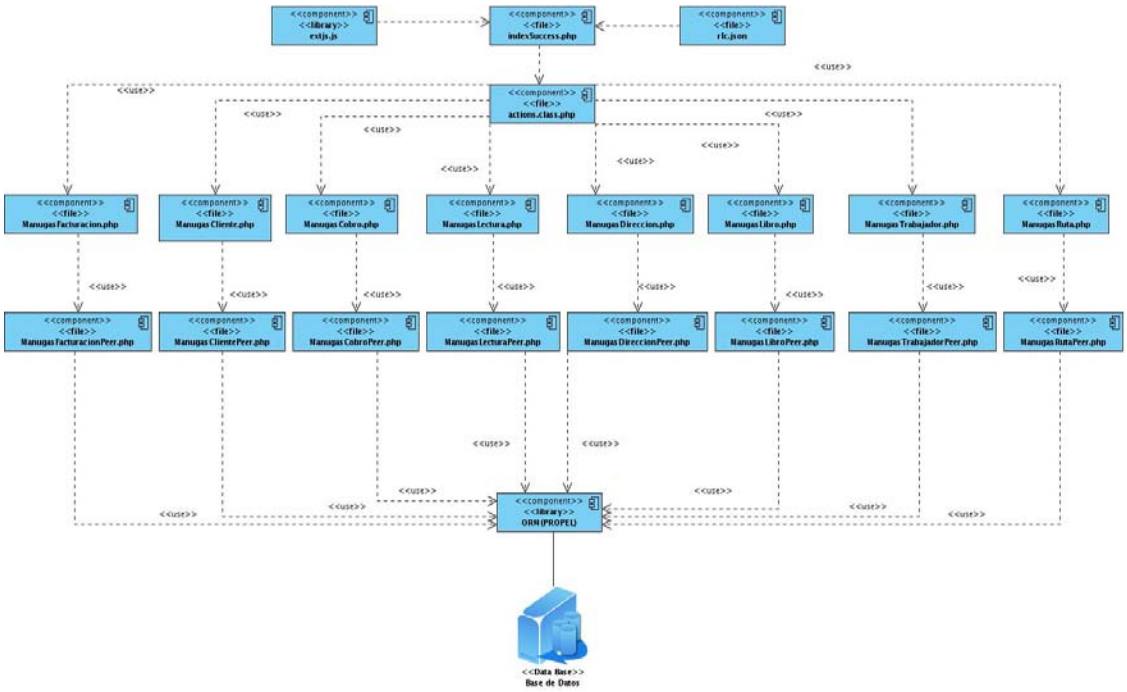

Figura 17 Diagrama de Componentes. Subsistema de "Cobro". CU Generar Reporte de Lector – Cobrador

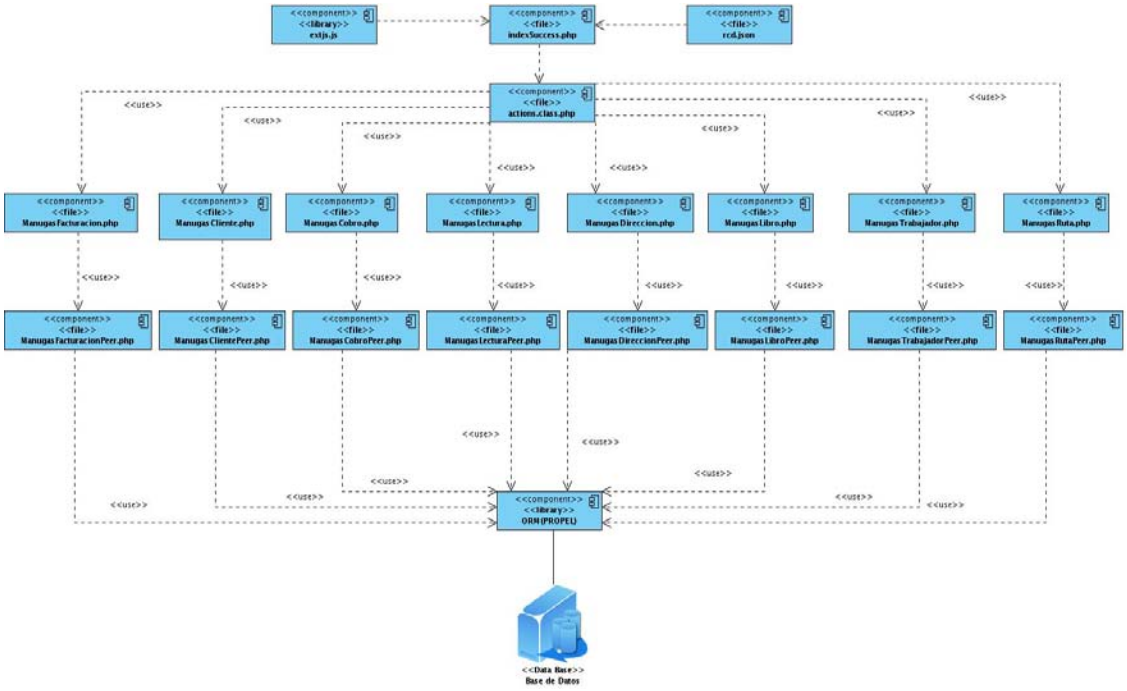

Figura 18 Diagrama de Componentes. Subsistema de "Cobro". CU Generar Reporte Certificación de Deuda

3.3. Pruebas Unitarias

Las pruebas unitarias verifican que cada método y función está trabajando correctamente, cada prueba deberá ser lo más independiente posible de las demás. Una de las buenas prácticas del desarrollo web que más cuesta a los programadores, consiste en realizar pruebas unitarias; debido a que no están acostumbrados a probar a profundidad los métodos y algoritmos diseñados, surgen varias dudas: ¿Tengo que escribir las pruebas antes de programar la nueva funcionalidad? ¿Qué debo probar? ¿Las pruebas tienen que probar hasta los casos más extremos? ¿Cómo puedo asegurarme de que estoy probando todo bien?

Teniendo en cuenta el problema surgido en los programadores de sistemas web; Symfony incluye un Módulo de Pruebas Unitarias, donde los programadores pueden realizar este tipo de pruebas a los sistemas implementados. El principal problema de las Pruebas Unitarias Automáticas es que es un poco complejo aprender a usarlas y configurarlas para que se realicen las pruebas según las necesidades del desarrollador, debido a esto Symfony incluye su propia librería para realizar Pruebas Unitarias, llamada **lime**, la que simplifica la realización de las pruebas. ***(Potencier, 2010)***.

3.3.1. Subsistema de Facturación

En el subsistema de Facturación todas las acciones giran sobre la clase Cliente, debido a esto, todas las pruebas son sobre esta clase, ya que las otras clases son auxiliares y no se consideró necesario realizar las pruebas pertinentes a estas clases. En caso de ser necesaria la validación de las demás clases, dirigirse al documento de Pruebas de Caja Blanca, Pruebas de Caja Negra y Pruebas de Validación del Sistema de Facturación y Cobro para la Empresa de Gas Manufacturado de Ciudad de la Habana.

```php
<?php
require_once '/home/pedro/Tools/frameworks/symfony-1.4.2/test/bootstrap/propel.php';
$test = new lime_test(7);
$cliente = getCliente(5);

// Pruebas sobre un cliente
$test->is($cliente->getNombre(),'Solangel Daniela Carbajal Romero',
          '->getNombre() -- CU Crear DBF, Leer DBF, Facturar');
$test->is($cliente->getLastlectura()->getLectura(),10,
          '->getLastLectura()->getLectura() -- CU Crear DBF, Leer DBF, Facturar');
$test->is($cliente->getLastlectura()->getClaveLecturaId(),1,
          '->getLastLectura()->getClaveLecturaId() -- CU Crear DBF, Leer DBF, Facturar, Exportar Lecturas');
$test->is($cliente->getPromedioLectura(), 4.2,
          '->getPromedioLectura() -- CU Crear DBF, Leer DBF, COrregir Errores, Facturar');
$test->is($cliente->getDireccionCompleta(),'San Lazaro 65   Amistad e infanta',
          '->getdireccionCompleta() -- CU Crear DBF, Exportar Lecturas');
$test->is($cliente->getLastConsumo(),5,
          '->getLastConsumo() -- CU Facturar');
$test->is($cliente->facturarLastLectura(1),null,
          '->facturarLastLectura() -- Metodo usado para facturar la ultima lectura del cliente');

function getCliente($cliente_id)
{
  $criteria =  new Criteria();
  $criteria->add(ManugasClientePeer::CLIENTE_ID,$cliente_id);

  $cliente = ManugasClientePeer::doSelectOne($criteria);

```

Figura 19 Código usado en la prueba (Subsistema de Facturación)

```
pedro@geysed-backup:~/Proyect/Manugas/trunk$ symfony-1.4 test:unit ManugasLectura
1..7
ok 1 - ->getNombre() -- CU Crear DBF, Leer DBF, Facturar
ok 2 - ->getLastLectura()->getLectura() -- CU Crear DBF, Leer DBF, Facturar
ok 3 - ->getLastLectura()->getClaveLecturaId() -- CU Crear DBF, Leer DBF, Facturar, Exportar Lecturas
ok 4 - ->getPromedioLectura() -- CU Crear DBF, Leer DBF, COrregir Errores, Facturar
ok 5 - ->getdireccionCompleta() -- CU Crear DBF, Exportar Lecturas
ok 6 - ->getLastConsumo() -- CU Facturar
ok 7 - facturarLastLectura() -- Metodo usado para facturar la ultima lectura del cliente
# Looks like everything went fine.
pedro@geysed-backup:~/Proyect/Manugas/trunk$ symfony-1.4 test:unit ManugasLectura
1..7
ok 1 - ->getNombre() -- CU Crear DBF, Leer DBF, Facturar
ok 2 - ->getLastLectura()->getLectura() -- CU Crear DBF, Leer DBF, Facturar
ok 3 - ->getLastLectura()->getClaveLecturaId() -- CU Crear DBF, Leer DBF, Facturar, Exportar Lecturas
ok 4 - ->getPromedioLectura() -- CU Crear DBF, Leer DBF, COrregir Errores, Facturar
ok 5 - ->getdireccionCompleta() -- CU Crear DBF, Exportar Lecturas
ok 6 - ->getLastConsumo() -- CU Facturar
ok 7 - ->facturarLastLectura() -- Metodo usado para facturar la ultima lectura del cliente
# Looks like everything went fine.
pedro@geysed-backup:~/Proyect/Manugas/trunk$ symfony-1.4 test:unit ManugasLectura
1..7
ok 1 - ->getNombre() -- CU Crear DBF, Leer DBF, Facturar
ok 2 - ->getLastLectura()->getLectura() -- CU Crear DBF, Leer DBF, Facturar
ok 3 - ->getLastLectura()->getClaveLecturaId() -- CU Crear DBF, Leer DBF, Facturar, Exportar Lecturas
ok 4 - ->getPromedioLectura() -- CU Crear DBF, Leer DBF, COrregir Errores, Facturar
ok 5 - ->getdireccionCompleta() -- CU Crear DBF, Exportar Lecturas
ok 6 - ->getLastConsumo() -- CU Facturar
ok 7 - ->facturarLastLectura() -- Metodo usado para facturar la ultima lectura del cliente
# Looks like everything went fine.
```

Figura 20 Salida de la prueba (Subsistema de Facturación)

3.3.2. Subsistema de Cobro

En el subsistema de Cobro el CU principal es Registrar Cobro en la Casa Comercial, debido a esto todas las pruebas serán sobre este CU, además, los otros CU son en su totalidad generar reportes, los cuales no necesitan muchas funcionalidades y métodos

de las modelos. En caso de ser necesario otro tipo de prueba dirigirse al documento Pruebas de Caja Negra y Pruebas de Caja Blanca del Sistema de Facturación y Cobro de la Empresa de Gas Manufacturado.

```php
<?php
require_once '/home/pedro/Tools/frameworks/symfony-1.4.2/test/bootstrap/unit.php';

$test = new lime_test(6);
$cliente = getCliente(5);

$test->is($cliente->getDeudasCliente(),array(),
          '->getDeudasCliente() -- CU Buscar_Cliente, Registrar_Cobro_Casa_Comercial');
$test->is($cliente->getPagos(),array(),
          '->getPagos() -- CU Buscar_Cliente, Registrar_Cobro_Casa_Comercial');
$test->is($cliente->getCambioMetro(),array(),
          '->getCambioMetro() -- CU Buscar_Cliente, Registrar_Cobro_Casa_Comercial');
$test->is($cliente->registrarCobro(), true,
          '->registrarCobro() -- CU Buscar_Cliente, Registrar_Cobro_Casa_Comercial');
$test->is($cliente->getDireccionCompleta(),'San Lazaro 65   Amistad e infanta',
          '->getdireccionCompleta() -- CU Crear DBF, Exportar Lecturas');
$test->is($cliente->datosReciboCobro(),array(),
          '->datosReciboCobro() -- CU Buscar_Cliente, Registrar_Cobro_Casa_Comercial');
$test->is($cliente->getDatosCliente(),array(),
          '->getDatosCliente() -- CU Buscar_Cliente, Registrar_Cobro_Casa_Comercial');

function getCliente($cliente_id)
{
  $criteria =  new Criteria();
  $criteria->add(ManugasClientePeer::CLIENTE_ID,$cliente_id);

  $cliente = ManugasClientePeer::doSelectOne($criteria);

```

Figura 21 Código usado en la prueba (Subsistema de Cobro)

```
pedro@geysed-backup:~/Proyect/Manugas/trunk$ symfony-1.4 test:unit Cobro
1..6
ok 1 - ->getDeudasCliente() -- CU Buscar_Cliente, Registrar_Cobro_Casa_Comercial
ok 2 - ->getPagos() -- CU Buscar_Cliente, Registrar_Cobro_Casa_Comercial
ok 3 - ->getCambioMetro() -- CU Buscar_Cliente, Registrar_Cobro_Casa_Comercial
ok 4 - ->registrarCobro() -- CU Buscar_Cliente, Registrar_Cobro_Casa_Comercial
ok 5 - ->datosReciboCobro() -- CU Buscar_Cliente, Registrar_Cobro_Casa_Comercial
ok 6 - ->getDatosCliente() -- CU Buscar_Cliente, Registrar_Cobro_Casa_Comercial
# Looks like everything went fine.
pedro@geysed-backup:~/Proyect/Manugas/trunk$ symfony-1.4 test:unit Cobro
1..6
ok 1 - ->getDeudasCliente() -- CU Buscar_Cliente, Registrar_Cobro_Casa_Comercial
ok 2 - ->getPagos() -- CU Buscar_Cliente, Registrar_Cobro_Casa_Comercial
ok 3 - ->getCambioMetro() -- CU Buscar_Cliente, Registrar_Cobro_Casa_Comercial
ok 4 - ->registrarCobro() -- CU Buscar_Cliente, Registrar_Cobro_Casa_Comercial
ok 5 - ->datosReciboCobro() -- CU Buscar_Cliente, Registrar_Cobro_Casa_Comercial
ok 6 - ->getDatosCliente() -- CU Buscar_Cliente, Registrar_Cobro_Casa_Comercial
# Looks like everything went fine.
pedro@geysed-backup:~/Proyect/Manugas/trunk$ symfony-1.4 test:unit Cobro
1..6
ok 1 - ->getDeudasCliente() -- CU Buscar_Cliente, Registrar_Cobro_Casa_Comercial
ok 2 - ->getPagos() -- CU Buscar_Cliente, Registrar_Cobro_Casa_Comercial
ok 3 - ->getCambioMetro() -- CU Buscar_Cliente, Registrar_Cobro_Casa_Comercial
ok 4 - ->registrarCobro() -- CU Buscar_Cliente, Registrar_Cobro_Casa_Comercial
ok 5 - ->datosReciboCobro() -- CU Buscar_Cliente, Registrar_Cobro_Casa_Comercial
ok 6 - ->getDatosCliente() -- CU Buscar_Cliente, Registrar_Cobro_Casa_Comercial
# Looks like everything went fine.
```

Figura 22 Salida de la prueba (Subsistema de Cobro)

3.4. Pruebas Funcionales

Las pruebas funcionales verifican que la aplicación resultante se comporta correctamente en todo su conjunto. Las pruebas funcionales son la mejor opción para probar la aplicación de extremo a extremo, desde la petición realizada por un navegador hasta la respuesta enviada por el servidor. Las pruebas funcionales prueban todas las capas de la aplicación: el sistema de enrutamiento, el modelo, las acciones y las plantillas. Son muy similares a lo que se hace manualmente: cada vez que añades o modificas una acción, se prueba en el navegador para comprobar que todo funciona bien al pulsar sobre los enlaces y botones y que todos los elementos se muestran correctamente en la página.

Las pruebas funcionales de Symfony permiten describir de forma sencilla los escenarios de la aplicación. Una vez definidos, los escenarios se pueden ejecutar automáticamente una y otra vez de forma que simule el comportamiento de un usuario con su navegador. Al igual que las pruebas unitarias, las pruebas funcionales te dan la confianza y tranquilidad de saber que lo que estás programando no va a romper nada en la aplicación. ***(Potencier, 2010)***.

Debido al uso de ExtJS para el diseño de la interfaz de la aplicación, uso de carga dinámica en tiempo de ejecución de los ficheros con el código perteneciente a la interfaz y la inexistencia de una herramienta para probar (testear) peticiones Ajax, resulta muy difícil realizar las pruebas funcionales a los módulos de facturación y cobro del Sistema de Facturación y Cobro de la Empresa de Gas Manufacturado de Ciudad de la Habana.

Teniendo en cuenta lo anteriormente planteado en caso de ser necesario validar alguna de las funcionalidades del sistema, dirigirse a los documentos de Pruebas de Caja Negra del Sistema de Facturación y Cobro para la Empresa de Gas Manufacturado de Ciudad de la Habana.

3.5. Conclusiones Parciales

Al concluir la investigación de este capítulo se arribó a las siguientes conclusiones:

- Las pruebas realizadas al subsistema de facturación y subsistema de cobro del Sistema de Facturación y Cobro de la Empresa de Gas Manufacturado de Ciudad de la Habana, validan la mayoría de las funcionalidades de estos subsistemas, en caso de ser necesario validar el sistema con una mayor profundidad, consultar los documentos de Prueba de Caja Blanca y Prueba de Caja Negra.
- El código usado en la implementación de los subsistemas puede ser mejorado, al agrupar varias funcionalidades que se encuentran estrechamente relacionadas en diferentes CU.
- Los diagramas de implementación presentados, representan los CU principales de los subsistemas de facturación y cobro, para un mayor entendimiento de estos CU, consultar el documento Modelo de Casos de Uso del Sistema, del Sistema de Facturación y Cobro para la Empresa de Gas Manufacturado de Ciudad de la Habana.

Conclusiones Generales

Una vez terminada la investigación, se evidencia que con el desarrollo de los módulos de facturación y cobro del Sistema de Facturación y Cobro para la Empresa de Gas Manufacturado de Ciudad de la Habana, se han cumplido los objetivos establecidos al inicio de la investigación, mejorando los procesos de facturación y cobro de la Empresa de Gas Manufacturado de Ciudad de la Habana.

El objetivo planteado en el diseño de la investigación fue cumplido, comprobándose la idea a defender como solución al problema científico que originó la investigación, tributando a los procesos de facturación y cobro.

Como resultado de la presente investigación, la Empresa de Gas Manufacturado cuenta con una aplicación web capaz de automatizar correctamente los procesos de facturación y cobro que se desarrollan en la empresa, la cual es capaz de manejar eficientemente los datos usados en los procesos anteriormente mencionados y reducir de forma considerable los errores humanos que puede ocurrir.

Recomendaciones

Luego de la presentación de la investigación para la creación de los módulos de facturación y cobro para la Empresa de Gas Manufacturado de Ciudad de la Habana, teniendo en cuenta los resultados obtenidos en la misma y basándose en la experiencia acumulada en el desarrollo del presente trabajo, se proponen las siguientes mejoras.

- Continuar la implementación de los módulos faltantes, tal es el caso del Módulo de Gestión de Cliente, Módulo de Gestión de Personal y Módulo de Seguridad.

- Incorporar una ayuda al sistema, garantizando la usabilidad del mismo.

- Implantar el Sistema de Facturación y Cobro para la Empresa de Gas Manufacturado y comenzar su uso.

- Profundizar en el estudio del frameworks Symfony en conjunto con ExtJS, debido a que no es común que estos frameworks sean usados en el desarrollo de una misma aplicación web, sería notable reflejar las características de esta condición.

Bibliografía Referenciada

Alfaro, F.M. *Tecnología Cliente - Servidor.*

Angel Alvarez, Miguel and López, Daniel, Gutiérrez,Manu. 2007. *PHP 5: Programación Orientada a Objeto.* 2007.

Angel Alvarez, Miguel. 2004. Que es la Programación Orientada a Objeto. [Online] Julio 24, 2004. [Citado: Marzo 03, 2010.] http://www.desarrolloweb.com/articulos/499.php.

ASF. 2010. Apache - Web Server. [Online] 01 05, 2010. [Citado: 10 de Febrero del 2010.] http://www.apache.org/.

Autoridad Aeronáutica Civil. Implementación del sistema de facturación y cobro FYC4. [En línea] [Citado: Octubre 12, 2009.] http://www.aeronautica.gob.pa/index.php?option=com_content&task=view&id=522&Itemid=2.

Ben-Kiki, Oren, Evans, Clark and Ingerson, Brian. 2006. *YAML Ain't Markup Language.* 2006.

CommTrack. Sistema de Facturación CommTrack. [En línea] [Citado: Octubre 10, 2009.] http://www.techmarksoftware.com/media/tsi_pdf/cbs4_espanol.pdf..

Connalen, J. 1999. *Building Web Applications with UML.* s.l. : Adison Wesley, 1999.

Corbyn, Chris. 2010. SwiftMailer. [En línea] Febrero 12, 2010. [Citado: Marzo 03, 2010.] http://www.swiftmailer.org/.

ExtJS development Group. 2010. ExtJS Javascript Framework and RIA Plataform. [En línea] Marzo 03, 2010. [Citado: Marzo 08, 2010.] http://www.extjs.com/.

Garrido, J. S. C. 2004. *Arquitectura y diseño de sistemas web modernos.* 2004.

Jacobson, Ivar, Booch, Grady and Rumbaugh, James. 2004. *El Proceso Unificado de Desarrollo de Software.* s.l. : Addison Wesley, 2004. ISBN 84-7829-036-2.

Maturana, Sergio. 2009. *Gescoop.* 2009.

Openbravo, S.L.U. 2010. OpenBravo ERP. [En línea] 2010. [Citado: Marzo 05, 2010.] http://www.openbravo.com/.

PHP Group. 2010. PHP Hipertext Preprocesor. [En línea] Marzo 17, 2010. [Citado: Marzo 17, 2010.] http://www.php.net/.

php.net. 2010. *PHP Development.* s.l. : Philip Olson, 2010.

PostgreSQL Fundation. 2009. PostgreSQL - Batabase Server. [En línea] 12 15, 2009. http://www.postgresql.org/.

Potencier, Fabien and Zaninotto, François. 2009. *Symfony, guía definitiva.* 2009.

Potencier, Fabien. 2010. *El Tutorial Jobbet.* 2010.

Prat Montesino, Noraisi. 2009. *Diseño de una Arquitectura de Software para el Sistema de Facturación y Cobro de la Empresa de Gas Manufacturado.* 2009.

Rivero Varona, Helimay. 2009. *ANÁLISIS Y DISEÑO DEL SISTEMA DE FACTURACIÓN Y COBRO.* Ciudad de la Habana : s.n., 2009.

Sánchez, Alain and Rodríguez, Darlenis. 2007. *Sistema de gestión para el Programa Nacional de Grupos Electrogenos DENYO.* Ciudad de la Habana : s.n., 2007.

Sensio Labs. 2010. Symfony | Web PHP Framework. [En línea] Marzo 10, 2010. [Citado: Marzo 12, 2010.] http://www.symfony-project.org/.

Visual Paradigm Design Group. 2009. Visual Paradigm. [En línea] Noviembre 12, 2009. [Citado: 01 18, 2010.] http://www.visual-paradigm.com/.

Zend GDE. 2010. Zend Studio Integration Development Enviroment. [En línea] Febrero 22, 2010. [Citado: Febrero 25, 2010.] http://www.zend.com/.

Bibliografía Consultada

Alfaro, F.M. *Tecnología Cliente - Servidor.*

Angel Alvarez, Miguel. 2004. Que es la Programación Orientada a Objeto. [En línea] 24 de Julio de 2004. http://www.desarrolloweb.com/articulos/499.php.

Angel Alvarez, Miguel y López, Daniel, Gutiérrez,Manu. 2007. *PHP 5: Programación Orientada a Objeto.* 2007.

ASF. 2010. Apache - Web Server. [En línea] 05 de 01 de 2010. http://www.apache.org/.

Autoridad Aeronáutica Civil. Implementación del sistema de facturación y cobro FYC4. [En línea] http://www.aeronautica.gob.pa/index.php?option=com_content&task=view&id=522&Itemid=2.

Ben-Kiki, Oren, Evans, Clark y Ingerson, Brian. 2006. *YAML Ain't Markup Language.* 2006.

CommTrack. Sistema de Facturación CommTrack. [En línea] http://www.techmarksoftware.com/media/tsi_pdf/cbs4_espanol.pdf..

Connalen, J. 1999. *Building Web Applications with UML.* s.l. : Adison Wesley, 1999.

Corbyn, Chris. 2010. SwiftMailer. [En línea] 12 de Febrero de 2010. http://www.swiftmailer.org/.

ExtJS development Group. 2010. ExtJS Javascript Framework and RIA Plataform. [En línea] 03 de Marzo de 2010. http://www.extjs.com/.

Frederick, Shea, Ramsay, Colin y Blades, Steve 'Cutter'. 2008. *Learning ExtJS.* s.l. : PACKT, 2008. ISBN 978-1-847195 .

—. 2008. *Learning ExtJS.* s.l. : PACKT, 2008. ISBN 978-1-847195 .

Garrido, J. S. C. 2004. *Arquitectura y diseño de sistemas web modernos.* 2004.

Jacobson, Ivar, Booch, Grady y Rumbaugh, James. 2004. *El Proceso Unificado de Desarrollo de Software.* s.l. : Addison Wesley, 2004. ISBN 84-7829-036-2.

Lellelid, Hans. 2005. *Propel - Guía de usuario.* 2005.

Maturana, Sergio. 2009. *Gescoop.* 2009.

Openbravo, S.L.U. 2010. OpenBravo ERP. [En línea] 2010. http://www.openbravo.com/.

PHP Group. 2010. PHP Hipertext Preprocesor. [En línea] 17 de Marzo de 2010. http://www.php.net/.

php.net. 2010. *PHP Development.* s.l. : Philip Olson, 2010.

PostgreSQL Fundation. 2009. PostgreSQL - Batabase Server. [En línea] 15 de 12 de 2009. http://www.postgresql.org/.

Potencier, Fabien. 2010. *El Tutorial Jobbet.* 2010.

Potencier, Fabien y Zaninotto, François. 2009. *Symfony, guía definitiva.* 2009.

Prat Montesino, Noraisi. 2009. *Diseño de una Arquitectura de Software para el Sistema de Facturación y Cobro de la Empresa de Gas Manufacturado.* 2009.

Rivero Varona, Helimay. 2009. *ANÁLISIS Y DISEÑO DEL SISTEMA DE FACTURACIÓN Y COBRO.* Ciudad de la Habana : s.n., 2009.

Sánchez, Alain y Rodríguez, Darlenis. 2007. *Sistema de gestión para el Programa Nacional de Grupos Electrogenos DENYO.* Ciudad de la Habana : s.n., 2007.

Sensio Lab. 2009. *Askeet - symfony.* 2009.

—. 2009. *More with symfony.* 2009.

—. 2009. *Practical symfony "Jobeet" + Propel.* 2009.

—. 2009. *symfony Forms in Action.* 2009.

—. 2009. *The symfony Cookbook.* 2009.

—. 2009. *The symfony Reference Book.* 2009.

Sensio Labs. 2010. Symfony | Web PHP Framework. [En línea] 10 de Marzo de 2010. http://www.symfony-project.org/.

Visual Paradigm Design Group. 2009. Visual Paradigm. [En línea] 12 de Noviembre de 2009. http://www.visual-paradigm.com/.

Wordware Publishing, Inc. 2001. *Advanced Javascript.* 2001. ISBN 1-55622-852-X.

Zend GDE. 2010. Zend Studio Integration Development Enviroment. [En línea] 22 de Febrero de 2010. http://www.zend.com/.

Glosario de Términos

Analista del Sistema: Persona encargada de procesar los datos enviados hacia la UEB Comercial, también realiza el proceso de facturación de los clientes.

Aplicación Web: Aplicación publicada en la Internet, a la cual se accede mediante los protocolos http(s), la característica fundamental de esta es que solo se encuentra publicada en un servidor al que se accede mediante un cliente.

Carga del TPL: Proceso mediante el cual la Tramitadora carga los datos de los clientes en el TPL.

Casa Comercial: Filiales de la UEB Comercial, ubicada en cada municipio de Ciudad de la Habana, son las encargadas de realizar la lectura y cobro del gas correspondiente a cada cliente.

Chequera: Reporte que es entregado al cliente como notificación de la facturación de su consumo.

Clave de Facturación: Clave asociada a cada cliente por la cual es facturado, cada clave tiene una tarifa escalonada distinta, son 10 claves (0-9).

Clave de Lectura: Claves utilizadas para representar las distintas situaciones en que se puede encontrar el Metro – Contador, son un total de 10 claves (0-9).

Cliente Metrado: Cliente que realizó un contrato con la UEB Comercial y tiene instalado un Metro – Contador, el proceso de Facturación se le realiza según el consumo mensual del cliente.

Cliente no Metrado: Cliente que realizó un contrato con la UEB Comercial, pero todavía no tiene instalado un Metro – Contador. El proceso de Facturación se realiza según un valor fijo en dependencia de la cantidad de integrantes del núcleo familiar.

Corrección de Errores: Proceso mediante el cual son corregidos los datos descargados del TPL, cuenta con 5 condiciones de corrección.

Criterios de Facturación: Sinónimo de Clave de Facturación.

Descarga del TPL: Proceso mediante el cual la Tramitadora descarga del TPL los datos de las lecturas de los clientes.

Folio: Término usado para agrupar varios clientes que por su dirección no pueden ser agrupados en un Libro.

Lector – Cobrador: Trabajador de la empresa, encargado de recoger las lecturas de los clientes, entregar las notificaciones de cobro, etc.

Libro: Término usado para agrupar varios clientes que viven en una misma calle.

Metro – Contador: Dispositivo instalado en la casa de los clientes usado para controlar el consumo de gas mensual.

Modelo de Incidencias: Modelo utilizado por los Lectores – Cobradores, en el que reflejan los errores cometidos en el proceso de recogida de las lecturas de los Metros – Contadores.

Núcleo Familiar: Término usado para definir los integrantes de una casa, donde se ha realizado un contrato con la UEB Comercial.

PHP: Pre-procesador de hipertexto.

Ruta: Término usado para denotar los Libros relacionados con un Lector – Cobrador.

SISMET: Sistema Metrado, es el sistema que se usa en la actualidad en las Casas Comerciales y la UEB Comercial para realizar los procesos de facturación y cobro.

Sistema web: Sinónimo de Aplicación Web.

Talonarios: Importe generado como contraparte de la chequera.

Tarifa Escalonada: Tarifa basada en rango de valores, mediante los cuales se calcula el monto a pagar por un cliente según su consumo de gas mensual.

TPL: Terminal Portátil de Lectura. Dispositivo usado por los Lectores – Cobradores para recoger los datos de las lecturas de los Metros – Contadores de los clientes.

Tramitadora: Trabajadora de la Casa Comercial encargada de interactuar con el sistema, cargar y descargar los datos del TPL.

UEB Comercial: Casa Matriz de la Empresa de Gas Manufacturado de Ciudad de la Habana, es la encargada de realizar la facturación de todos los clientes y la impresión de las chequeras y talonarios.

UEB: Unidad Eléctrica Básica.

Anexo I
Ejemplo del Estándar de Codificación de Symfony

```
<?php
class sfFoo
{
  public function bar()
  {
    sfCoffee::make();
  }
}
```

Ejemplo 1 Uso del Tab

```
<?php
if ($myVar == getRequestValue($name))    // correcto
if ( $myVar == getRequestValue($name) )  // incorrecto
```

Ejemplo 2 Uso del espacio

```
# Bien: function makeCoffee()
# Mal: function MakeCoffee()
# Mal: function make_coffee()
```

Ejemplo 3 Uso de la notación camelCase

```php
<?php
function makeCoffee()
{
  if (false !== isSleeping() && false !== hasEnoughCafeineForToday())
  {
    canMakeCoffee();

    return 1;
  }
  else
  {
    cantMakeCoffee();
  }

  return null;
}
```

Ejemplo 4 Uso del return

```
// espacio en blanco, esto es un comentario
```

Ejemplo 5 Declaración del comentario de línea

```php
<?php
$string = 'something';
$newString = "$string is awesome!";  // mal, no impresiona
$newString = $string.' is awesome!'; // mejor
$newString = sprintf('%s is awesome', $string); // para la validación de excepciones
```

Ejemplo 6 Ejemplo de cómo evaluar un string

```php
<?php
if (null !== $coffee)
{
  echo “Tengo café”;
}
```

Ejemplo 7 Comprobación de valor null de una variable

```php
<?php
if (“Hola Mundo” === $variable)
```

Ejemplo 8 Comparación de un string con una variable

```php
<?php
public function notify(sfEvent $event)
{
  // Implementación de la función
}
```

Ejemplo 9 Declaración de métodos y funciones

```php
<?php
/**
 *  Notifica la ocurrencia de un evento
 ** @param  sfEvent  $event  instancia de la clase sfEvent
 *
 * @return sfEvent  instancia del tipo sfEvent
 */
public function event(sfEvent $event)
```

Ejemplo 10 Documentación php

Anexo II
Casos de Uso del Sistema de Facturación y Cobro

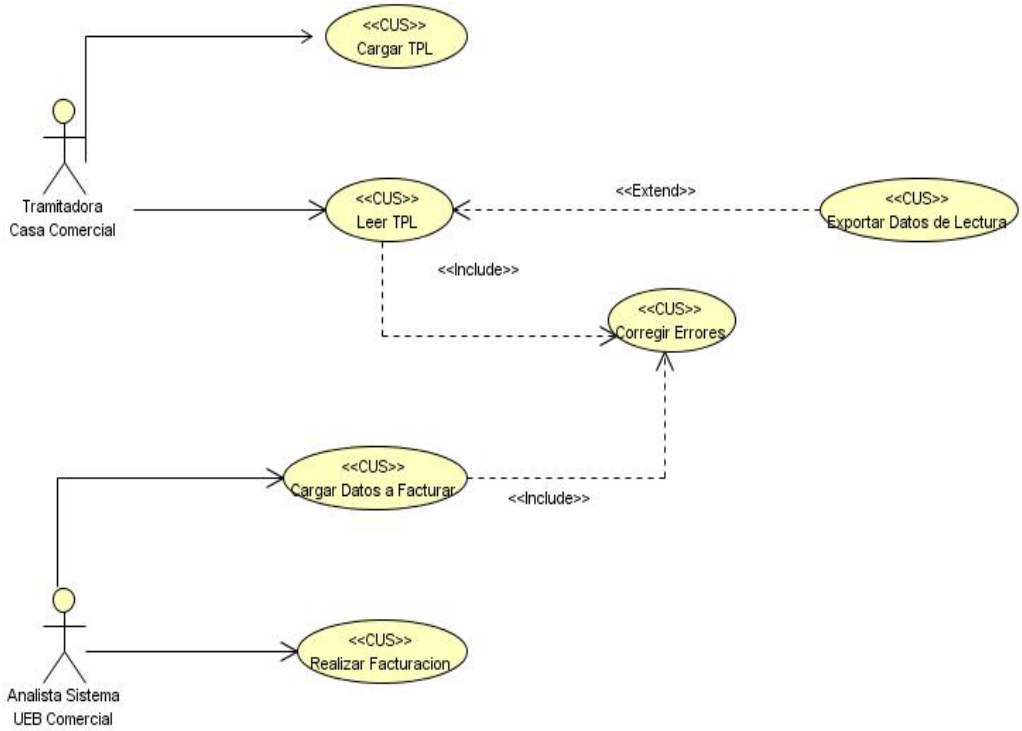

Figura 23 Casos de Uso: Módulo de Facturación

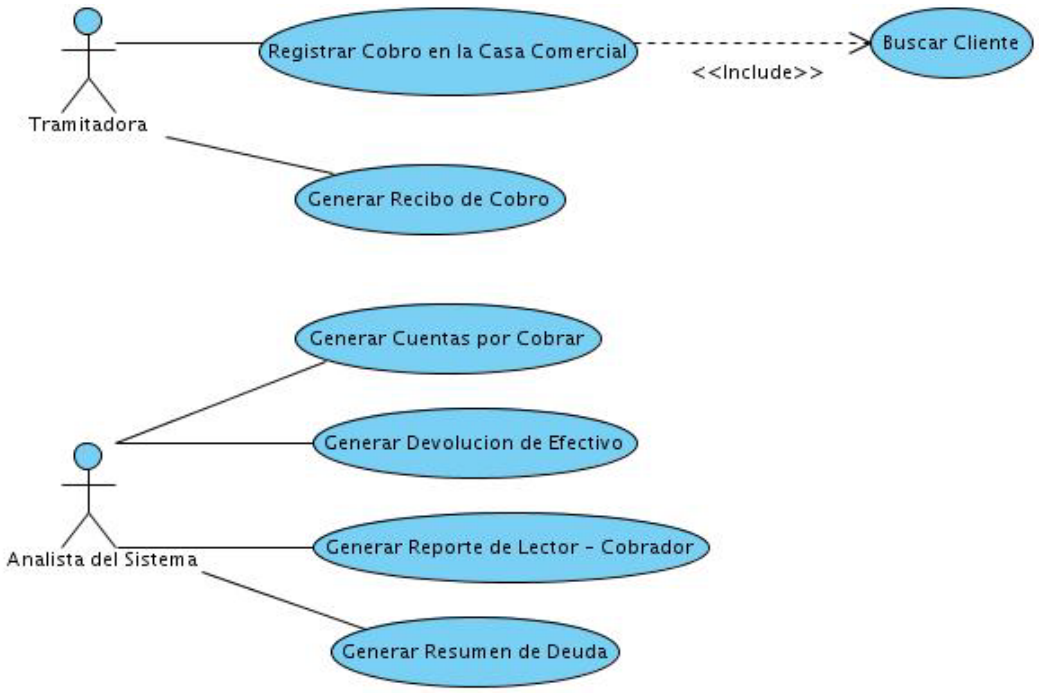

Figura 24 Casos de Uso: Modulo de Cobro