Universidad de las Ciencias Informáticas

Facultad 3

"Implementación del componente Administración de Ventas para el sistema de Administración de Relaciones con el Cliente"

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

Autor: Lorenzo Antonio Risquet Rodríguez

Tutoras: Ing. Leidy Ramos González

Ing. Olga Yarisbel Rojas Grass

La Habana, Junio de 2013
"Año 55 de la Revolución"

Si los jóvenes fallan, todo fallará. Es mi más profunda convicción que la juventud cubana luchará por impedirlo. Creo en ustedes.

Fidel Castro Ruz

Declaro ser autor de la presente tesis y reconozco a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo. Para que así conste firmo la presente declaración jurada de autoría en La Habana a los ____ días del mes de junio del año 2013.

Lorenzo Antonio Risquet Rodríguez

_____

Firma del Autor

Ing. Leidy Ramos González

_____

Firma del Tutor

Ing. Olga Yarisbel Rojas Grass

_____

Firma del Tutor

Tutora: Ing. Leidy Ramos González

Correo electrónico: lramosg@uci.cu

Título de graduado: Ingeniero en Ciencias Informáticas

Situación laboral: Profesor

Categoría Docente: Asistente

Institución: Universidad de las Ciencias Informáticas (UCI).

Dirección: Carretera a San Antonio de los Baños, Km 2 1/2, Reparto Torrens, Boyeros

Tutora: Ing. Olga Yarisbel Rojas Grass

Correo electrónico: yarisbel@uci.cu

Título de graduado: Ingeniero en Ciencias Informáticas

Situación laboral: Especialista general

Categoría Docente: Instructor

Institución: Universidad de las Ciencias Informáticas (UCI).

Dirección: Carretera a San Antonio de los Baños, Km 2 1/2, Reparto Torrens, Boyeros

A mi mamá, por haberme apoyado en todos estos años, eres la mejor madre del mundo, te quiero.

A mi papá, por guiarme y aconsejarme en todo momento, un abrazo.

A mis abuelos, por inculcarme el deseo de superarme siempre, ellos también son parte de este sueño hecho realidad.

A mis hermanitos, por darme tanto amor y cariño, los quiero mucho.

A Roger y toda mi familia por estar siempre a mi lado y ayudarme en todo lo que necesité.

A mis compañeros de aula y edificio por haberme soportado todos estos años.

A mis tutoras por guiarme y ayudarme en todo momento.

A todos les agradezco.

En la actualidad las empresas han enfocado sus estrategias de negocio a los clientes, comprendiendo que gastan más recursos en buscar nuevos que reteniendo a los existentes. Los sistemas de Administración de Relaciones con el Cliente (ARC) son herramientas que les permiten a las empresas afianzar sus clientes, realizar mayores ventas y tener menores costos de mercadotecnia.

En el país el conocimiento que existe de estas herramientas es poco y su utilización no se enfoca en los clientes. Las entidades presentan problemas a la hora de gestionar sus clientes, guardando información incompleta, con datos erróneos o duplicados. Lo anterior repercute en que al vender un producto, el proceso parte casi desde cero al no poder la empresa crear un perfil del cliente

El presente trabajo tiene como objetivo realizar la implementación del proceso Administración de Ventas del sistema ARC; el cual será implementado por un equipo de trabajo del departamento SOLEM. La solución permitirá gestionar con mayor agilidad las ventas, específicamente las ofertas y los contratos. Con lo planteado anteriormente se busca lograr la satisfacción de los clientes, generando mayores ganancias para las empresas. Este componente fue probado utilizando pruebas de caja blanca y caja negra siendo aprobado por los analistas y el cliente cumpliendo con la totalidad de los objetivos propuestos.

Palabras claves: Administración de Relaciones con el Cliente, Ventas

TABLA DE CONTENIDO

INTRODUCCIÓN 1
Capítulo 1: Fundamentación Teórica 6
Introducción 6
1.1 Marco conceptual 6
1.2 Análisis de sistemas 7
1.3 Metodología, lenguajes y herramientas 12
Conclusiones parciales 18
Capítulo 2: Implementación 19
Introducción 19
2.1 Requisitos 19
2.2 Valoración del diseño propuesto por el analista 21
2.3 Artefactos de la fase de implementación 22
2.4 Estándar de codificación 26
2.5 Descripción de las clases 28
Conclusiones parciales 30
Capítulo 3: Pruebas 31
Introducción 31
3.1 Pruebas de software 31
3.2 Prueba de software que será aplicada al componente 33
Conclusiones parciales 40
CONCLUSIONES GENERALES 41
RECOMENDACIONES 42
REFERENCIAS BIBLIOGRÁFICAS 43
GLOSARIO DE TÉRMINOS 46
ANEXOS 48
Anexo 1. Interfaces del componente Gestión de Ventas 48
Anexo 2. Descripción de clases y métodos 52
Anexo 3: Caso de prueba de caja negra para validar el requisito Eliminar cliente 55

INTRODUCCIÓN

A partir de la década del 90 la evolución tecnológica ha ayudado a las empresas a convertirse en entidades con modernas infraestructuras y mejores procesos, logrando con ello una elevada eficiencia operacional. En sus inicios los sistemas informáticos se restringían a grandes empresas garantizando una enorme ventaja competitiva y valor agregado. Con el paso del tiempo, estos sistemas pasaron a controlar todas las operaciones de manera integrada, garantizando: la disponibilidad de la información para los usuarios en tiempo real, la eliminación de la barrera de la distancia trabajando con un mismo sistema en puntos distantes y la disminución de errores, por lo que se volvió una herramienta indispensable para las empresas.

Con el objetivo de controlar la información que se genera en cada departamento de las entidades surge el Sistema de Planificación de Recursos Empresariales (ERP[1]). En los ERP al integrar las áreas de negocio se conjuga todo lo necesario para el funcionamiento de los procesos de negocio de la empresa en una sola aplicación, facilita la retroalimentación tanto en la comunicación interna con los recursos humanos de una organización como en la externa con los clientes, buscando siempre aumentar la eficiencia de las empresas (1).

En un mercado competitivo y exigente caracterizado por la globalización donde diversas entidades ofrecen productos similares, es prácticamente imposible para las mismas crecer y sobrevivir sin un sistema de gestión integral, además que han comprendido que el éxito hay que buscarlo en una correcta relación con los clientes, donde los productos ya no son sólo objetos con características funcionales, sino medios para facilitarles mejores experiencias. Por esta razón muchas empresas han centrado sus estrategias de negocio en los clientes, por lo que precisan de herramientas tecnológicas que faciliten la administración de la relación con el mismo e informen sobre sus necesidades sin desperdiciar tiempo y dinero, surgiendo de esta forma la Administración de Relaciones con el Cliente (CRM[2]).

Resulta imprescindible para los CRM contar con una excelente base de datos centralizada como las ofrecidas por los sistemas ERP que interactúan entre sí consolidando todas las operaciones e informando de los gustos y necesidades de los clientes. CRM más que una herramienta, es una estrategia que brinda

[1] Siglas en inglés de Enterprise Resource Planning
[2] Siglas en inglés de Customer Relationship Management

cuando es bien implementada varias ventajas, como son mayores ventas y menores costos de mercadotecnia, soporte de ventas y servicio al cliente, además del aumento de la satisfacción del mismo.

Para contrarrestar los daños provocados en mayor medida por el bloqueo, la dirección del país ha emprendido una profunda reorganización en las entidades económicas. En la actualidad, y dando cumplimiento a lo planteado en los Lineamientos de la Política Económica y Social del Partido y la Revolución aprobados por el VI Congreso del Partido Comunista de Cuba (PCC), se estimula la producción y comercialización de alimentos a escala local y se les dan facilidades de compra de equipos a los agricultores particulares. A las empresas se les otorga más autoridad y control sobre sus actividades y sobre una parte de sus ganancias, al mismo tiempo que se les exige dar prioridad a inversiones que permitan rendir ganancias en breve plazo (2).

En resumen, los lineamientos plantean la necesidad de realizar cambios en las empresas cubanas con el objetivo de lograr un mayor desarrollo económico así como un mejor nivel de vida en la población. A pesar de los esfuerzos de la dirección del país por mejorar la economía, la situación de la nación es compleja, ya que diversas empresas no poseen indicadores ofrecidos bajo el enfoque de los sistemas, que las ayuden a establecer las relaciones entre las entradas y salidas, en sus relaciones con sus clientes, el mercado y la competencia. También presentan problemas en la gestión de los clientes, la información que se guarda de los mismos se encuentra incompleta, con datos erróneos o duplicados. Dentro de la empresa no hay una comunicación entre las áreas de mercadotecnia, ventas y atención al cliente, por lo que las oportunidades generadas por una de las áreas no son aprovechadas por las otras. Cuando se va a vender un producto, el proceso casi parte desde cero ya que no se puede crear un perfil de los usuarios. Además en algunas empresas las compras y las ventas no son realizadas por el mismo departamento lo que trae consigo demora en la atención de las solicitudes y el incumplimiento de contratos, propiciando así la insatisfacción de los clientes. En varias entidades realizan un incorrecto seguimiento de las ventas lo que trae consigo: incumplimiento de las fechas de entrega de los productos a los clientes, presentan demoras en las atenciones a las reclamaciones de los clientes insatisfechos además de no tener conocimiento de si el producto llegó en buen estado al cliente.

A partir del estudio realizado se llega a la conclusión de la necesidad de una herramienta que permita la administración de las ventas en una empresa. El departamento SOLEM perteneciente al centro CEIGE de la Universidad de las Ciencias Informáticas, luego de un estudio realizado por el equipo de arquitectura del

proyecto se definió como suite de desarrollo a OpenERP. Este es un sistema ERP que contiene un módulo de Administración de Relaciones con el Cliente (ARC), de código abierto y que brinda una serie de funcionalidades en las acciones de ventas para las empresas como son la gestión de clientes y el pedido de venta. En el caso de las entidades cubanas OpenERP no cubre todas las necesidades de las mismas. OpenERP no gestiona los contratos siendo el punto de partida del seguimiento de las ventas, por lo que a partir del análisis y diseño desarrollado por analistas del centro SOLEM, se hace necesario el desarrollo de funcionalidades que permitan la adaptación de dicha herramienta al modelo económico cubano.

Sobre la base de los elementos expuestos anteriormente se formula el siguiente **problema a resolver:** ¿cómo contribuir al seguimiento de las ventas realizadas a los clientes desde el componente de Administración de Ventas, en el sistema de Administración de Relaciones con el Cliente (ARC)?

Para ello la investigación centra su **objeto de estudio** en la administración de ventas y el **campo de acción** se enmarca en la: implementación del proceso administración de ventas en los sistemas de administración de relaciones con el cliente.

Para la solución del problema planteado se define como **objetivo general:** implementar las funcionalidades del componente de Administración de Ventas del Sistema de Administración de Relaciones con el Cliente (ARC).

Para cumplir con este objetivo se plantean una serie de **objetivos específicos:**

- Realizar el marco teórico de la investigación.
- Implementar el componente siguiendo las técnicas de programación estudiadas y el estándar de codificación.
- Validar la solución propuesta mediante las pruebas de caja negra y caja blanca.

Para dar cumplimiento a los objetivos trazados es necesario cumplir las siguientes **tareas de la investigación:**

- Elaboración el marco conceptual de la investigación.
- Análisis valorativo acerca del proceso Administración de Compra/Venta en diferentes sistemas de ARC.
- Definición de los patrones de diseño a emplear.

- Propuesta de las herramientas y tecnologías a utilizar para el desarrollo de la solución.
- Análisis crítico del diseño propuesto.
- Elaboración del mapa de componentes.
- Elaboración de los algoritmos de implementación.
- Aplicación del estándar de codificación definido.
- Implementación de las funcionalidades definidas en los requisitos del sistema.
- Realización de la descripción por funcionalidades.
- Obtención de los ficheros de código.
- Realización de las pruebas de caja negra sobre las funcionalidades.
- Realización de las pruebas de caja blanca sobre el código desarrollado.

Para la elaboración del presente trabajo de diploma se emplearon métodos científicos tanto teóricos como empíricos, entre los teóricos están:

- Histórico-Lógico: a través de este método se establece la necesaria correspondencia entre los elementos de los métodos lógico e histórico, proyectando el análisis de la evolución histórica de los fenómenos, con la proyección lógica de su comportamiento futuro. Fue utilizado para el estudio de los sistemas de Administración de Relaciones con el Cliente.
- Inductivo-Deductivo: para el estudio y análisis de la biografía.

De los empíricos se utilizó la observación para reunir información visual de la situación existente en las empresas cubanas.

Este documento se encuentra estructurado en tres capítulos que contienen toda la información referente a la investigación realizada:

En el **Capítulo 1,** denominado “Fundamentación Teórica”, se realiza una fundamentación de las tecnologías, metodologías de desarrollo y herramientas a utilizar en el desarrollo de la aplicación.

En el **Capítulo 2**, denominado “Implementación”, se realiza la descripción y el análisis de la solución obtenida con el apoyo de los artefactos del análisis y diseño propuestos por los analistas, como son los diagramas de clases del diseño y las descripciones de las funcionalidades a informatizar.

En el **Capítulo 3**, denominado “Pruebas”, se muestra los resultados de las pruebas de caja blanca y de caja negra realizadas respectivamente al software y la interfaz de la aplicación, con el objetivo de garantizar la calidad de la solución propuesta.

Capítulo 1: Fundamentación Teórica

Introducción

En este capítulo se mencionan una serie de conceptos necesarios para el entendimiento de la investigación. Se realizó un estudio de sistemas ARC que son usados actualmente nacional e internacionalmente. Se argumenta y justifica el uso de los lenguajes, herramientas y las tecnologías para el desarrollo de la aplicación.

1.1 Marco conceptual

Ventas

Del latín vendita, venta es la acción y efecto de vender (traspasar la propiedad de algo a otra persona tras el pago de un precio convenido). El término se usa tanto para nombrar a la operación en sí misma como a la cantidad de objetos que se venden (3).

El Diccionario de la Real Academia Española, define a la venta como "la acción y efecto de vender. Cantidad de cosas que se venden. Contrato en virtud del cual se transfiere a dominio ajeno un artículo propio por el precio pactado" (4).

Para el autor, venta es un proceso en el cual el vendedor ofrece un producto que es adquirido por el comprador pagando un precio considerado entre ambas partes.

Administración de Relaciones con el Cliente

El CRM consiste en una estrategia de la organización, la cual centra sus esfuerzos en el conocimiento de sus clientes, detectando sus necesidades, aumentando su grado de satisfacción, incrementando su fidelidad a la empresa e incrementando la rentabilidad o beneficios del cliente a la empresa, mediante el análisis de las informaciones extraídas por los clientes desde los diferentes canales o medios de comunicación (5).

El CRM se refiere a aquellas aplicaciones que las empresas pueden utilizar para administrar todos los aspectos de sus encuentros con los clientes. Un sistema CRM puede incluir todo, desde tecnología para la

recolección de datos en las llamadas telefónicas del área de ventas, hasta sitos web de autoservicio donde los clientes pueden aprender acerca de los productos y de su compra, o el análisis de los clientes y los sistemas de administración de campaña (5).

Para el autor, CRM es un sistema que le permite a las empresas mejorar las relaciones con sus clientes. Para lograr esto, la herramienta tecnológica brinda una serie de datos de los clientes. Los datos son analizados y usados por la empresa para introducir en el mercado productos que satisfagan las necesidades de sus clientes.

ERP (Planificación de Recursos Empresariales)

ERP se define como un grupo de módulos conectados a una única base de datos. El ERP es un paquete de software que permite administrar todos los procesos operativos de una empresa, integrando varias funciones de gestión en un único sistema (6).

Para el autor, ERP es un software que integra a los distintos departamentos de una empresa controlando la información que en estos se genera. Esta información ayuda a las empresas a tener una mejor planificación, análisis y gestión de sus recursos.

1.2 Análisis de sistemas

En este epígrafe se realiza un estudio de sistemas ARC que son utilizados tanto nacional como internacionalmente.

SugarCRM se adapta fácilmente a cualquier entorno empresarial ofreciendo una alternativa más flexible y rentable que las aplicaciones propietarias. La arquitectura de código abierto de SugarCRM permite a las empresas una personalización muy sencilla que permite integrar los procesos de negocio a fin de construir y mantener relaciones estables con los usuarios. La filosofía empresarial del CRM es el código abierto así que incluye todas aquellas tecnologías necesarias para que la aplicación funcione, como son PHP, MySQL y el servidor web Apache (7).

Acciones de ventas:

- Gestión de oportunidades.
- Gestión de clientes.

- Gestión de clientes potenciales.
- Gestión de pronósticos de ventas.
- Gestión de contratos (8).

SAP Customer Relationship Management (SAP CRM), incluido en el SAP Business Suite es una solución capaz de viabilizar procesos de negocios totalmente confiables, amplios y centrados en el cliente, proporcionando un beneficio real de cada cliente. El software SAP CRM se aplica a procesos dirigidos a los mercados verticales para sustentar departamentos que interaccionan directamente con los clientes, específicamente en las áreas de marketing, ventas y servicios. SAP CRM asegura una visibilidad de 360 grados de todos los puntos de contacto de los clientes y de los canales de interacción, incluyendo internet, centros de interacción y socios del canal. La solución CRM ofrece poderosos recursos analíticos (9).

Acciones de ventas:

- Gestión de cuentas y contactos.
- Gestión de actividades.
- Gestión de oportunidades.
- Gestión de territorios.
- Gestión de pedido de venta.
- Gestión de contratos.

Vtiger CRM es una aplicación para la administración de relaciones con clientes, cien por ciento código libre. Controla el ciclo de vida de la relación con el cliente con infinidad de tareas, no limitándose a ser un directorio de usuarios sino que relaciona esta información con otras áreas como las ventas, atención al cliente, marketing o productos. Surgió como un derivado de SugarCRM y actualmente cuenta con una gran comunidad alrededor de todo el mundo con el objetivo de convertirse en la aplicación más abierta y mejor implementada para cualquier tipo de empresa. (9)

Acciones de ventas:

- Gestión de contactos.
- Gestión de oportunidad de venta.

- Gestión de pedido de venta.
- Gestión de facturas.
- Gestión de tarifas.

Oracle Siebel CRM se cuenta entre las plataformas CRM líderes del mundo y se implementa en empresas grandes y medianas para optimizar todos los factores relativos a las relaciones con los clientes: en departamentos de marketing y ventas, para centros de contactos, gestión de relaciones con socios, gestión de presupuestos y pedidos. Proporciona una plataforma escalable, flexible, así como una herramienta de análisis de largo alcance que permite a las organizaciones cambiar, analizar y monitorizar los procesos orientados al cliente, así como simplificar la gestión de los datos recibidos de dichos clientes (7).

Acciones de ventas:

- Administración de cuentas.
- Gestión de oportunidades.
- Metodologías de ventas.
- Previsión de ventas.
- Gestión de pedidos (10).

Microsoft Dynamics CRM permite aumentar las ventas y la satisfacción del cliente final, siendo muy fácil de usar, personalizar y mantener. Este software ofrece un amplio espectro de características, estando al alcance de cualquier tipo de empresa, permitiendo a la compañía cumplir con sus necesidades específicas. (7)

Acciones de ventas:

- Gestiona tipo de órdenes.
- Gestiona tipo de operación.
- Genera reportes personalizados para cada etapa.
- Gestiona las liquidaciones de comisiones.

- Gestiona los pedidos de venta.
- Gestiona los descuentos de ventas.
- Gestiona las solicitudes de ventas (11).

OpenERP es un sistema de gestión empresarial (ERP) y de relación con el cliente (CRM) de código abierto. OpenERP cubre las necesidades de las áreas de contabilidad, ventas, compras, y almacén e inventario, entre otras. También soporta múltiples monedas, múltiples compañías y múltiples contabilidades; además incorpora funcionalidades de gestión de documentos para agilizar la colaboración entre departamentos y equipos en la empresa; y permite trabajar remotamente mediante una interfaz web desde una computadora conectada a Internet (12).

Características:

- Se integra con distintos software de oficina. Dispone de funcionalidad para la generación de impresos vía PDF, HTML, y permite exportar datos a otros programas como OpenOffice o MS Office (Excel, Word).
- La arquitectura del sistema es cliente – servidor, lo que permite que todos los usuarios trabajen sobre el mismo repositorio de datos. Esto tiene la ventaja de que toda la información está disponible y sincronizada en todo momento además de que descarga la mayor parte del trabajo de procesamiento de datos de las máquinas cliente (donde trabajan efectivamente los usuarios). Dispone de interfaces XML-RPC y SOAP.
- Dentro de la construcción misma del software se hace un uso intensivo de flujos de trabajo que se pueden integrar con sus distintos módulos (13).

Acciones de ventas:

- Gestionar iniciativa de venta.
- Gestión de pedido de venta.
- Gestión de cliente.
- Convertir iniciativa en oportunidad.

Tabla 1. Comparación entre los CRM

Sistemas	OpenERP	Vtiger CRM	Sugar CRM	Microsoft Dynamics	SAP AG	Oracle Siebel
Multiplataforma	Sí	Sí	Sí	No	No	No
Costo (Bajo)	Sí	Sí	Sí	No	No	No
Soporte	No	No	No	Sí	Sí	Sí
Pedidos de Ventas	Sí	Sí	No	Sí	Sí	Sí
Oportunidades	Sí	Sí	Sí	No	Sí	Sí
Contratos	No	Sí	Sí	No	Sí	No
Clientes	Si	No	Sí	No	No	No

Después del estudio de estos sistemas se llegó a la conclusión de que para el país no sería factible el uso de algunos de ellos; es el caso de SAP, Microsoft Dynamics y Siebel, ya que son propietarios y aunque posean empresas que le den soporte y ventajas en el área de la Administración de Ventas, el costo por el software, la licencia y el mantenimiento sería muy alto. Al ser construidos bajo herramientas y software propietarios no es posible acceder al código de las funcionalidades, por lo que es difícil adaptarlos al modelo económico cubano.

El resto de los sistemas están desarrollados sobre software libre y aunque no tengan empresas que le den soporte, su costo inicial es bajo. Al pertenecer a la familia de código abierto es posible modificarlos y adaptarlos a las necesidades de las entidades del país. Los arquitectos del proyecto seleccionaron al OpenERP como suite de desarrollo ya que brinda flexibilidad y simplicidad a la hora de realizar modificaciones y adaptaciones, ajustándolo a las necesidades, se integra con herramientas de negocio, utiliza un flujo de trabajo flexible y dinámico, se le puede agregar funciones y módulos e integrarlos a los ya existentes, además de soportar plataformas heterogéneas. Esta elección cumple con la política cubana de migrar al software libre, buscando la independencia tecnológica.

1.3 Metodología, lenguajes y herramientas

A continuación se profundiza en la metodología, tecnologías y herramientas definidas por los arquitectos del proyecto para el desarrollo de la aplicación.

1.3.1 Modelo de desarrollo

Para el desarrollo de las funcionalidades sobre la suite se utilizará el Modelo de desarrollo del Centro de Informatización de la Gestión de Entidades (CEIGE) v1.1. El mismo detalla el ciclo de vida de sus proyectos con la incorporación de los distintos subprocesos dictados por el Nivel II del Modelo de Madurez de la Capacidad Integrado (CMMI[3]), certificación obtenida por el centro en julio de 2011 y reconocida por el Instituto de Ingeniería de Software (SEI[4]) como aval de la calidad de su proceso de desarrollo de software.

Descripción general de la disciplina Implementación

A partir de los resultados del análisis y diseño se implementa el sistema en términos de componentes, es decir, ficheros de código fuente, scripts, ejecutables y similares. Al reutilizar componentes software ya implementados se lleva a cabo el desarrollo necesario para ajustar a los requisitos actuales y posteriormente realizar la integración de los componentes. En la siguiente figura se muestran las actividades por las que pasa la disciplina implementación (14).

[3] Siglas en inglés de Capability Maturity Model Integration.
[4] Siglas en inglés de Software Engineering Institute

Figura 1. Actividades de la Implementación

1.3.2 Lenguajes

Python es un lenguaje de guión, independiente de plataforma y orientado a objetos, preparado para realizar cualquier tipo de programa, desde aplicaciones Windows a servidores de red o incluso, páginas web. Es un lenguaje interpretado, lo que significa que no se necesita compilar el código fuente para poder ejecutarlo, lo que ofrece ventajas como la rapidez de desarrollo e inconvenientes como una menor velocidad (15).

Características

- Lenguaje de alto nivel.
- Implementado en C (Jython, IronPython, Pypy).
- Multiplataforma.
- Multiparadigma: programación OO, estructurada o funcional (16).

Ventajas

- La cantidad de librerías que contiene, tipos de datos y funciones incorporadas en el propio lenguaje, que ayudan a realizar muchas tareas habituales sin necesidad de tener que programarlas desde cero.
- La sencillez y velocidad con la que se crean los programas. Un programa en Python puede tener de 3 a 5 líneas de código menos que su equivalente en Java o C.
- La cantidad de plataformas en que se puede desarrollar, como Unix, Windows, OS/2, Mac, Amiga y otros.

Además, Python es gratuito, incluso para propósitos empresariales (16).

XML es un formato basado en texto, específicamente diseñado para almacenar y transmitir datos. Un documento XML se compone de elementos XML, cada uno de los cuales consta de una etiqueta de inicio, de una etiqueta de fin y de los datos comprendidos entre ambas etiquetas. XML no es realmente un lenguaje en particular, sino una manera de definir lenguajes para diferentes necesidades, de ahí que se le denomine metalenguaje (16).

Ventajas

- Fácilmente procesable tanto por humanos como por software.
- Separa radicalmente la información o el contenido de su presentación o formato.
- Diseñado para ser utilizado en cualquier lenguaje o alfabeto.
- Su análisis sintáctico es fácil debido a las estrictas reglas que rigen la composición de un documento.
- Estructura Jerárquica.
- El número de marcas es ilimitado (16).

Herramientas

Eclipse 3.3.0 es un Entorno de Desarrollo Integrado (IDE, por sus siglas en inglés) que en un principio fue diseñado y desarrollado por IBM y que luego fue lanzada a la comunidad de software libre, y que se

distribuye mediante una licencia de código abierto, la Eclipse Public License (EPL). La plataforma de Eclipse integra herramientas de desarrollo, con una arquitectura abierta y basada en plugins. Además, proporciona soporte a todo tipo de proyectos que abarcan desde el ciclo de vida del desarrollo de aplicaciones, incluyendo soporte para modelado.

La arquitectura de plugins permite integrar diversos lenguajes sobre un mismo IDE e introducir otras aplicaciones accesorias. Conservan el registro de las versiones, generan y mantienen la documentación de cada etapa del proyecto. Modifica e inspecciona valores de variables. También avisa de los errores cometidos mediante una ventana secundaria, y depura el código que resida en una máquina remota. Eclipse permite agrupar código escrito y mostrarlo visualmente como una estructura empaquetada para que sea fácil poder seguir un código escrito (17).

PyDev es un plugin para Python en el IDE Eclipse, que puede utilizarse en el desarrollo de Python, Jython, IronPython (18).

Características de PyDev 2.5

- Integración de Django.
- Análisis de código.
- Depurador.
- Refactorización.
- Navegador de tokens.
- Consola interactiva (18).

Servidor de aplicaciones OpenObject 6.0.3

Es un marco totalmente modular, incluyendo mapeo objeto relacional (ORM) en la parte superior de PostgreSQL, un motor de flujo de trabajo, un sistema de informes pdf, un sistema de vistas (formularios, listas, árboles, tablero de instrumentos, calendario), así como un motor de migración automatizada (19).

Servidor de Base de Datos: PostgreSQL 9.1

PostgreSQL es un sistema de gestión de bases de datos objeto-relacional, distribuido bajo licencia BSD y con su código fuente disponible libremente. Es el sistema de gestión de bases de datos de código abierto

más potente del mercado. PostgreSQL utiliza un modelo cliente-servidor y usa multiprocesos en vez de multihilos para garantizar la estabilidad del sistema. Un fallo en uno de los procesos no afectará el resto y el sistema continuará funcionando (20).

Características:

- Integridad referencial.
- Replicación asincrónica/sincrónica.
- Copias de seguridad.
- Unicode.
- Juegos de caracteres internacionales.
- Regionalización por columna.
- Control de Concurrencia para Múltiples Versiones.
- Múltiples métodos de autentificación.
- Acceso encriptado vía SSL.
- Completa documentación.
- Licencia BSD.
- Disponible para Linux y UNIX en todas sus variantes (20).

1.3.3 Patrones de diseño

Los patrones GRASP describen los principios fundamentales de la asignación de responsabilidades a objetos, expresados en formas de patrones. A continuación se da la definición formal de cada uno de ellos.

Experto: asignar una responsabilidad al experto en información; la clase que tiene la información necesaria para llevar a cabo la responsabilidad.

Controlador: asignar la responsabilidad de gestionar un mensaje de un evento del sistema a una clase que represente el sistema global, dispositivo o subsistema (controlador de fachada), ó represente un escenario de caso de uso en el que tiene lugar el evento del sistema (controlador de caso de uso o de sesión)

Alta cohesión: asignar responsabilidades de manera que la información que almacena una clase sea coherente y esté relacionada con la clase.

Bajo acoplamiento: diseñar con el objetivo de tener las clases lo menos ligadas entre sí que se pueda. De tal forma que en caso de producirse una modificación en alguna de ellas, se tenga la mínima repercusión posible en el resto de clases, potenciando la reutilización, y disminuyendo la dependencia entre las clases (21).

1.3.4 Patrón arquitectónico

El patrón Modelo Vista Controlador (MVC) es una arquitectura de diseño software para separar los componentes de aplicación en tres niveles, interfaz de usuario, lógica de control y lógica de negocio. La finalidad del modelo es mejorar la reusabilidad por medio del desacople entre la vista y el modelo (22).

Ventajas del patrón

- Es posible tener diferentes vistas para un mismo modelo.
- Es posible construir nuevas vistas sin necesidad de modificar el modelo subyacente.
- Proporciona un mecanismo de configuración a componentes complejos (el modelo puede verse como una representación estructurada del estado de la interacción) (22).

En el caso del sistema el uso de este patrón se evidencia en que la base de datos es el modelo, los objetos de OpenERP (clases u objetos de Python) que proporcionan los módulos son el controlador, y los archivos XML que hay dentro de los módulos definen las vistas (23).

Pruebas de calidad

Al desarrollar un nuevo software o sistema de información, la primera etapa de pruebas a considerar es la etapa de pruebas unitarias o también llamada pruebas de caja blanca (White Box), estás pruebas también son llamadas pruebas modulares ya que permiten determinar si un módulo del programa está listo y correctamente terminado, estas pruebas no se deben confundir con las pruebas informales que realiza el

programador mientras está desarrollando el módulo. El objetivo fundamental de las pruebas unitarias es asegurar el correcto funcionamiento de las interfaces, o flujo de datos entre componentes (24).

Se denominan pruebas funcionales, a las pruebas de software que tienen por objetivo probar que los sistemas desarrollados cumplan con las funciones específicas para los cuales han sido creados, también se les llaman pruebas de comportamiento o pruebas de caja negra, ya que los analistas de pruebas, no enfocan su atención a como se generan las respuestas del sistema, básicamente el enfoque de este tipo de prueba se basa en el análisis de los datos de entrada y en los de salida, esto generalmente se define en los casos de prueba preparados antes del inicio de las pruebas (25).

El objetivo principal de aplicarle pruebas de calidad a las funcionalidades que serán creadas es que el producto final cumpla con las especificaciones definidas por el cliente. Con esto se lograría que el cliente este satisfecho y por consiguiente un aumento de la competitividad de la empresa.

Conclusiones parciales

En este capítulo se abordaron los elementos teóricos que sustentan la solución del problema, partiendo del proceso de Ventas en una empresa utilizando como partida los sistemas de Administración de Relaciones con el Cliente.

Con el estudio realizado sobre los diferentes sistemas que poseen un proceso de Ventas se pudo concluir que estos presentan ventajas y desventajas de acuerdo a los parámetros medidos, siendo la suite de desarrollo OpenERP la más completa y apropiada para la personalización en el centro CEIGE.

Se le implementará un componente con algunas funcionalidades necesarias y que esta suite no tenía concebida y que su uso es fundamental en el centro. Las herramientas y tecnologías seleccionadas permitirán implementar y validar la solución que se propone elaborar en los capítulos posteriores.

Capítulo 2: Implementación

Introducción

En este capítulo se describen los elementos tenidos en cuenta para la implementación de los componentes de la solución. Se exponen los requisitos funcionales detectados por los analistas como punto de partida para conseguir un adecuado desarrollo de la aplicación. Se muestra como queda concebida la arquitectura a través de las posibilidades que proporciona el marco de trabajo OpenObject y cómo influye la misma en la programación de las funcionalidades implementadas.

2.1 Requisitos

Según el Glosario Estándar de Ingeniería de Software del Instituto de Ingenieros Eléctricos y Electrónicos (IEEE) un requisito se puede definir como:

- Una condición o necesidad de un usuario para resolver un problema o alcanzar un objetivo.
- Una condición o capacidad que debe estar presente en un sistema o componentes de sistema para satisfacer un contrato, estándar, especificación u otro documento formal.
- Una representación documentada de una condición o capacidad como en los casos anteriores (26).

Requisitos Funcionales

Son declaraciones de los servicios que debe proporcionar el sistema, de la manera en que éste debe reaccionar a entradas particulares y de cómo se debe comportar en situaciones particulares. En algunos casos, los requerimientos funcionales de los sistemas también pueden declarar explícitamente lo que el sistema no debe hacer. Estos requerimientos dependen del tipo de software que se desarrolle, de los posibles usuarios del software y del enfoque general tomado por la organización al redactar requerimientos (27).

A continuación la tabla 2 incluye el listado de los requisitos funcionales a implementar en este trabajo.

Tabla 2. Listado de Requisitos Funcionales

Gestionar oferta	Adicionar oferta
	Eliminar oferta
	Buscar oferta
	Modificar oferta
	Listar oferta
	Consultar oferta
Gestionar pedido de venta	Adicionar pedido de venta
	Eliminar pedido de venta
	Buscar pedido de venta
	Modificar pedido de venta
	Listar pedido de venta
	Consultar pedido de venta
Gestionar cliente	Adicionar cliente
	Eliminar cliente
	Buscar cliente
	Modificar cliente
	Listar cliente
	Consultar cliente
Gestionar contrato	Adicionar contrato
	Eliminar contrato
	Buscar contrato
	Listar contrato
	Modificar contrato
	Consultar contrato

Requisitos No Funcionales

Para el presente trabajo se tuvieron en cuenta los requisitos no funcionales definidos en el documento de arquitectura de software, situado en el repositorio del departamento Soluciones Empresariales.

2.2 Valoración del diseño propuesto por el analista

El objetivo de la fase de implementación es tomar el diseño propuesto e implementarlo en archivos de código fuente, librerías de clases y ejecutables. Para llevar a cabo ese objetivo es necesario revisar los artefactos generados en la fase de análisis y diseño ya que pueden existir elementos que se hayan definido de forma incorrecta.

En el caso del modelo de datos propuesto por los analistas fue necesario transformarlo, existían dos clases nomencladoras, Nom_Tipo_de_contrato y Nom_Tipo_Oferta con los mismos atributos, código, descripción y tipo (servicio ó producto) para dos clases controladoras Tipo_contrato y Oferta, se decidió dejar una sola clase nomencladora y relacionar la misma con las clases implicadas. Además en dicho modelo algunas clases no poseían llave primaria siendo esto un error. Para que el componente de Servicios pudiera interactuar con el componente de Ventas necesitaba los atributos id_producto de tipo char, dicho atributo hace relación al identificador de la clase producto y el atributo devolución de tipo booleano, el mismo se refiere a si en el contrato, el producto puede ser o no devuelto. La clase Pedido se decidió eliminar, pasando el atributo estado (solicitado ó procesado) referente al estado del pedido, para la clase Pedido_venta. Entre la clase Cliente y la clase Oferta se decidió establecer una relación mucho a mucho, algo que no estaba definido anteriormente, además de la necesidad de incluir a la clase Contrato los siguientes atributos:

Tabla 3. Listado de Atributos

Nombre	Tipo	Descripción
cant_producto	float	Cantidad real de productos que existe
producto_cliente	float	Cantidad de productos que compra el cliente
precio_venta	float	Precio que tiene el producto
precio_venta_final	float	Precio total de la venta del producto

El uso por parte del analista del patrón GRASP le imprime mayor claridad a los programadores en el proceso de trance del diseño a la implementación, potenciando la reutilización, flexibilidad y organización del código. Los utilizados en el presente trabajo son los siguientes:

Experto: se pone en práctica con el uso de clases que tienen responsabilidades específicas, por ejemplo las clases controladoras. A la hora de definir, eliminar y modificar un contrato, la clase Contrato es la responsable de efectuar estas operaciones, asumiendo toda la lógica que implica estas funciones.

Controlador: las clases Contrato.py, Cliente.py son ejemplos del uso de este patrón en la solución del sistema, las mismas manejaran los eventos dentro del componente.

Bajo acoplamiento: en el modelo de datos se definieron un conjunto de clases persistentes, entre las cuales se establecieron las relaciones necesarias de manera que fueran más independientes. De esta forma se aumenta la reusabilidad, se reduce la dependencia entre las clases generando un menor impacto a la hora de realizar algún cambio.

Alta cohesión: este patrón fue utilizado en el diseño del componente de manera general; ya que las clases se agruparon en dependencia de los requerimientos, por ejemplo, la clase Oferta gestiona las ofertas, la clase Pedido_venta gestiona todo lo relacionado a los pedidos realizados por los clientes y así sucesivamente con el resto de las clases que conforman el componente.

Creador: este patrón se evidencia en las clases OSV encargadas de implementar el ORM en el componente, es decir la misma crea los objetos para la transferencia de información con la base de datos.

El uso del patrón Modelo-Vista-Controlador quedó evidenciado de la siguiente forma: el modelo estará constituido por una base de datos, no obstante lo que se ve a nivel de desarrollador es el ORM. Todas las clases en el componente derivan de la clase OSV.OSV y a la vez esta implementa el ORM. En las mismas se indican los atributos que se necesitan, de que tipo son y él se encarga del mapeo, haciendo totalmente transparente el acceso a la base de datos. El controlador es completamente código python, ejemplo la clase Contrato.py, Cliente.py, estas dan sustento a las ventanas y en las mismas se definen las funcionalidades necesarias para cumplir con los requisitos definidos. La vista son los archivos .XML, ejemplo cliente_view.xml, en la misma se definen el orden que van a tener los atributos así como los tipos de vista a mostrar, los tipos de vistas utilizados fueron los form (formularios) y los tree (árboles ó lista).

2.3 Artefactos de la fase de implementación

2.3.1 Modelado de datos

El modelo de datos va a describir el comportamiento de los elementos físicos persistentes que dan soporte a la información que almacenará el componente, con sus atributos correspondientes. Con el objetivo de garantizar la permanencia de los datos se modeló y normalizó el modelo de entidad relación del componente para la gestión del proceso Venta.

A continuación se muestra dicho modelo:

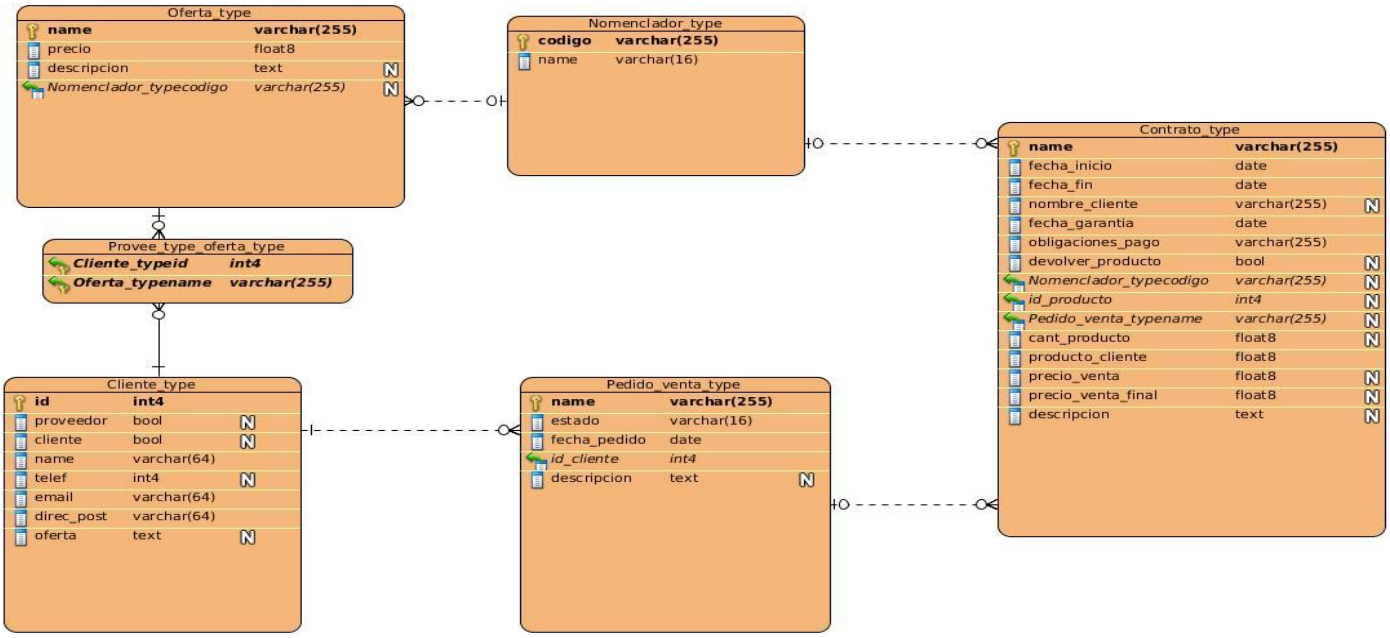

Figura 2. Modelo de Entidad-Relación

2.3.2 Modelo de Componentes

Un diagrama de componentes representa cómo un sistema de software es dividido en componentes, mostrando además las dependencias entre estos. Este tipo de diagrama contiene componentes, interfaces y sus relaciones, puede contener también paquetes que se utilizan para agrupar elementos del modelo. Teniendo en cuenta la arquitectura definida por la dirección del proyecto y el modelo de desarrollo a utilizar se definen como estructura básica a los componentes (28).

La figura 3 muestra el diagrama de componente propuesto para el desarrollo de la solución:

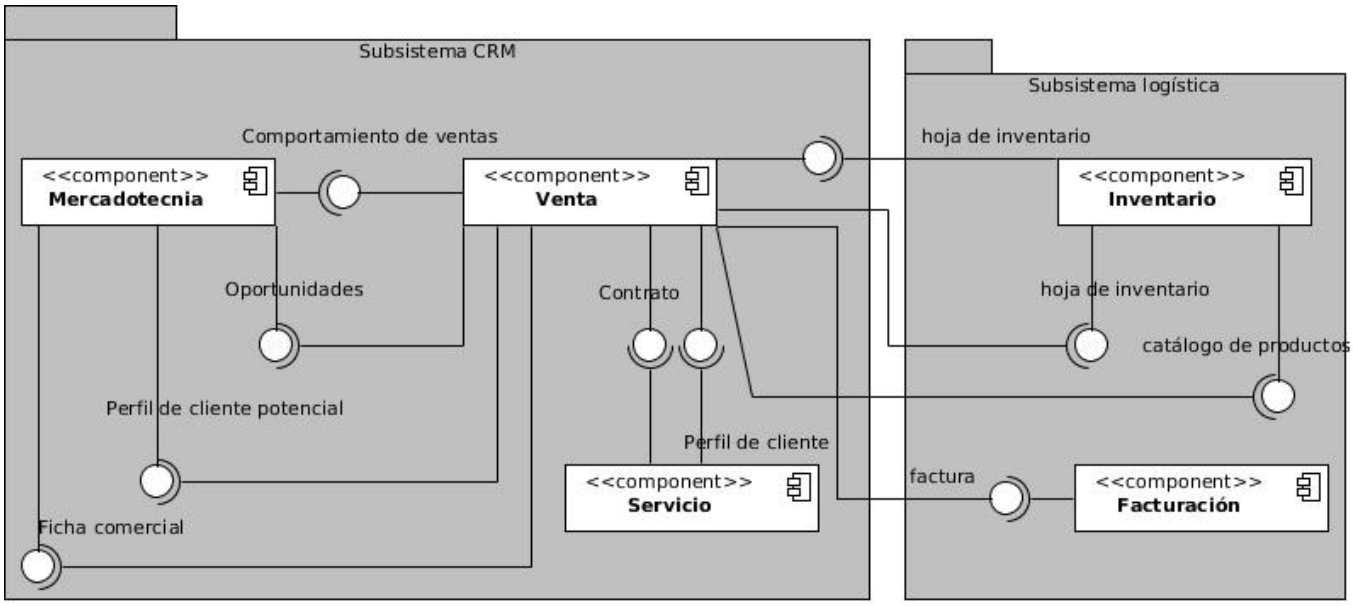

Figura 3. Diagrama de componentes

A continuación, una explicación más detallada de la funcionalidad de cada uno de estos componentes a partir de los servicios que brindan y reciben:

Venta: se encarga de registrar las iniciativas, oportunidades y pedidos de venta.

Mercadotecnia: este componente se encarga de realizar un estudio del mercado, en el mismo se ejecutaran actividades que ayuden a la empresa a gestionar las relaciones con los clientes.

Servicio: este componente permite brindar prestaciones a los clientes como, reparaciones, quejas y devoluciones para lograr una mayor satisfacción de los mismos.

2.3.3 Diagrama de Despliegue

En los diagramas de despliegue se visualiza las relaciones físicas de los distintos nodos que componen un sistema, el reparto de los componentes sobre dichos nodos. Este diagrama también muestra la configuración en funcionamiento del sistema, incluyendo su hardware y su software. Está compuesto por nodos, dispositivos y conectores. El propósito del modelo de despliegue es capturar la configuración de los elementos de procesamiento y las conexiones entre estos elementos en el sistema (29).

A continuación se presenta el modelo de despliegue elaborado para el componente:

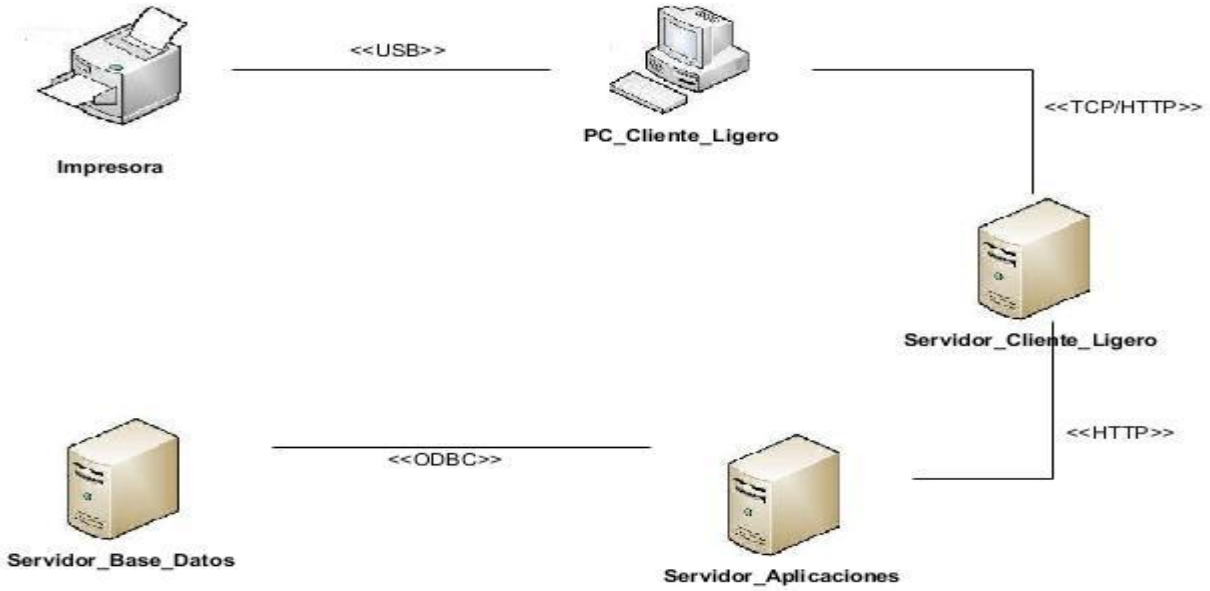

Figura 4. Diagrama de Despliegue

2.3.3.1 Descripción de elementos e interfaces de comunicación

<<Nombre tipo de conexión>>: Características físicas de la conexión

<<ODBC>>: es un estándar de acceso a bases de datos, el objetivo de ODBC acceder a cualquier dato desde cualquier aplicación, sin importar qué sistema de gestión de bases de datos (DBMS) almacene los datos. ODBC logra esto al insertar una capa intermedia denominada nivel de interfaz de cliente SQL, entre la aplicación y el DBMS, el propósito de esta capa es traducir las consultas de datos de la aplicación en comandos que el DBMS entienda (30).

<<HTTP>>: el protocolo de transferencia de hipertexto es un protocolo que define la sintaxis y la semántica que utilizan los elementos de software de la arquitectura web (clientes, servidores, proxies) para comunicarse. Es un protocolo orientado a transacciones, sin estado, es decir, que no guarda ninguna información sobre conexiones anteriores y sigue el esquema petición-respuesta entre un cliente y un servidor, además opera en la capa de presentación (31).

<<FTP>>: el protocolo de transferencia de archivos es utilizado para la transferencia de archivos entre sistemas conectados a una red, basado en la arquitectura cliente-servidor. Desde un equipo cliente se

puede conectar a un servidor para descargar archivos desde él o para enviarle archivos, independientemente del sistema operativo utilizado en cada equipo (31).

<<TCP>>: el Protocolo de Control de Transmisión permite a dos anfitriones establecer una conexión e intercambiar datos. El TCP garantiza la entrega de datos, es decir, que los datos no se pierdan durante la transmisión y también garantiza que los paquetes sean entregados en el mismo orden en el cual fueron enviados (33).

2.3.3.2 Descripción de los nodos

PC_Cliente_Ligero: accede a la aplicación a través de un computador, donde es ejecutada mediante el navegador Mozilla Firefox v12.0 o superior, sobre cualquier sistema operativo. Se conecta al Servidor_Cliente_Ligero mediante los protocolos HTTP y TCP.

Servidor_Cliente_Ligero: en este se encuentran los recursos utilizados por la PC_Cliente_Ligero y permite la conexión de la misma con el Servidor_Aplicaciones, se conecta al servidor de aplicaciones mediante el protocolo HTTP.

Servidor_Aplicaciones: como servidor de aplicaciones se utiliza Apache 2.2.4 con la librería adicional Python 2.7, es el encargado de atender las solicitudes del nodo PC_Cliente_Ligero, radicando en el mismo la lógica de negocio del sistema.

Servidor_Base_Datos: como servidor de base de datos se encuentra PostgreSQL v9.1, este responde a las peticiones realizadas por el Servidor_Aplicaciones de almacenamiento y recuperación de datos, a través del protocolo ODBC.

2.4 Estándar de codificación

Indentación

Usa 4 espacios por cada nivel de indentación.

¿Tabuladores o espacios?

Nunca mezclar tabuladores y espacios.

Tamaño máximo de línea

El máximo de caracteres por líneas es de 79.

Comentarios

Los comentarios deberían ser frases completas. Si un comentario es una frase o sentencia, la primera palabra debería estar en mayúsculas. Antes de declarar una clase o una función se escribe una breve descripción que exponga el propósito de la misma.

```
##############################################################################
# Clase: Cliente                                                             #
# Descripción: Clase controladora, gestiona los clientes y proveedores.      #
# Autor: Lorenzo Antonio Risquet Rodríguez                                   #
# Versión: 1.0                                                               #
##############################################################################
```

Figura 5. Comentario de la clase Cliente

Ejemplo de comentario de una función:

```
#     Método para autocompletar el id del nomenclador.
```

Figura 6. Comentario de la función def _autompletar_codigo

Nombres de Paquetes y Módulos

Los módulos deberían tener nombres cortos formados en su totalidad por letras minúsculas y terminarán con la palabra module**,** en caso de ser un nombre compuesto se puede utilizar guiones bajos en el nombre del módulo si mejora la legibilidad. Ejemplo: **contrato_module**

Nombres de Clases

Los nombres de clases usan la convención CapWords. Ejemplo: **class Contrato ():**

Nombres de Excepciones

Dado que las excepciones son clases, se aplica la convención relativa al nombrado de clases.

Nombres de Funciones

Los nombres de funciones estarán en letras minúsculas, con palabras separadas mediante guiones bajos según sea necesario para mejorar la legibilidad. Ejemplo: **def check_contrato (self, cr, uid, ids, context= None):**

Nomenclatura de las variables

Las variables serán nombradas comenzando en minúscula, en caso de que el nombre de estas sea compuesto, se utilizará guión bajo para separarlas. Ejemplo: **primer_apellido.**

2.5 Descripción de las clases

Las clases controladoras coordinan las actividades de los objetos que implementan las funcionalidades, definen el flujo de control y las transacciones entre los objetos.

Tabla 4. Descripción de la clase Oferta

Nombre: Oferta_type		
Descripción: La tabla va a recoger los datos de una oferta		
Atributo	**Tipo**	**Descripción**
name	char	Identificador de la oferta
precio	float	Precio de la oferta
descripcion	char	Descripción de la oferta

Tabla 5. Descripción de la clase Cliente

Nombre: Cliente_type

Descripción: La tabla va a recoger los datos de un cliente		
Atributo	**Tipo**	**Descripción**
id	Integer	Identificador de la clase cliente
proveedor	booleano	Si el cliente es proveedor
cliente	booleano	Activado por defecto
name	char	Nombre del cliente
telef	integer	Teléfono del cliente
email	char	Correo electrónico del cliente
direc_post	char	Dirección postal del cliente
oferta	text	Campo donde el cliente redacta lo que está buscando u ofreciendo

Tabla 6. Descripción de la clase Contrato

Nombre: Contrato_type		
Descripción: La tabla va a recoger los datos de un contrato		
Atributo	**Tipo**	**Descripción**
name	char	Identificados de la clase contrato
fecha_inicio	date	Fecha que fue creada el contrato
fecha_fin	date	Fecha en que finaliza el contrato
nombre_cliente	char	Nombre del cliente

fecha_garantia	date	Fecha de garantía
obligaciones_pago	char	Obligaciones de pago
devolver_producto	booleano	Saber si el producto admite devolución o no
cant_producto	float	Cantidad real de productos que existe
producto_cliente	float	Cantidad de productos que compra el cliente
precio_venta	float	Precio que tiene el producto
precio_venta_final	float	Precio total de la venta del producto
descripcion	text	Descripción del contrato

El resto de las descripciones se pueden consultar en los anexos: ver Anexo 2.

Conclusiones parciales

En este capítulo se realizó una valoración del análisis y diseño propuesto por los analistas del proyecto. Se explicaron los componentes utilizados para el desarrollo de la aplicación, lográndose un mejor entendimiento de la solución propuesta al mostrar la integración entre ellos. El modelo de datos aportó una visión de cómo están relacionadas y cuáles son las clases persistentes del componente. En general se logró implementar los requerimientos definidos por los analistas para el componente administración de ventas para el sistema CRM. Luego de haber concluido la implementación de la aplicación se pueden realizar las pruebas sobre el componente.

Capítulo 3: Pruebas

Introducción

La calidad de los productos, sean informáticos o no, es una actividad necesaria en las empresas que los desarrollan si se quiere sobresalir en un mercado que es cada día más competitivo y exigente. Debido a la imposibilidad humana de trabajar y comunicarse de forma perfecta, en la informática, el desarrollo de software ha de ir acompañado de una actividad que garantice la calidad. La prueba del software es un elemento crítico para la garantía de calidad del software y representa una revisión final de las especificaciones, del diseño y de la codificación.

Calidad se refiere a:

- El grado en que el producto satisface los requerimientos funcionales y no funcionales explícitamente establecidos.
- El nivel al que se siguieron los estándares de desarrollo explícitos y documentados.
- Que el producto muestre las características implícitas que se espera de todo software desarrollado profesionalmente (34).

3.1 Pruebas de software

Consiste en realizar ensayos de funcionamiento de las aplicaciones en entornos controlados, a fin de detectar los posibles defectos presentes antes de que el producto se ponga en funcionamiento y pueda originar cualquier tipo de fallo. Si las pruebas no son eficaces, pueden permitir que software defectuoso llegue al cliente causando graves problemas. Un proceso de prueba será exitoso cuando encuentre errores (35).

Niveles de Prueba

Cuando se va a evaluar un sistema se debe comenzar por los componentes más simples e ir avanzando hasta que se pruebe el sistema completo. Existen distintos niveles dentro de las pruebas de software, los mismos son los siguientes:

- Prueba Unitaria: son pruebas individuales a las unidades separadas de un sistema de software. Su objetivo es detectar errores en los datos, lógica y algoritmos utilizados en el desarrollo del software.
- Pruebas de Integración: los componentes individuales son combinados con otros componentes para asegurar que la comunicación, enlaces y los datos compartidos ocurran apropiadamente. Su objetivo es detectar errores en las interfaces y las relaciones entre los componentes del sistema.
- Pruebas del Sistema: son usualmente conducidas para asegurar que todos los módulos trabajan como sistema sin error. Es similar a la prueba de integración pero con un alcance mucho más amplio. Su objetivo es detectar fallas en el cubrimiento de los requerimientos.
- Pruebas de Aceptación: son realizadas principalmente por los usuarios con el apoyo del equipo del proyecto. El propósito es confirmar que el sistema está completo, que desarrolla puntualmente las necesidades de la organización y que es aceptado por los usuarios finales. Su objetivo es detectar fallas en la implementación del sistema (36).

Métodos de prueba

Tienen el objetivo de diseñar pruebas que descubran diferentes tipos de errores con menor tiempo y esfuerzo (37).

Entre los mecanismos de prueba de software se encuentran:

Caja Negra: pruebas funcionales sin acceso al código fuente de las aplicaciones, se trabaja con entradas y salidas.

Caja Blanca: pruebas con acceso al código fuente (datos y lógica). Se trabaja con entradas, salidas y el conocimiento interno (36).

Objetivo

Comprobar en qué medida cumple el software desarrollado con las expectativas del cliente en cuanto a los requisitos definidos. A partir de esto los objetivos principales de la realización de una prueba son:

- Detectar un error.

- Reducir costos de mantenimiento, mediante el diagnóstico oportuno de los componentes del sistema.
- Disminuir la penosa y costosa labor de soporte a usuarios insatisfechos, consecuencia de liberar un producto inmaduro. Esto hace que mejore la imagen de la organización desarrolladora y la credibilidad en ella.
- Obtener información concreta acerca de fallas, que pueda usarse como apoyo en la mejora de procesos (38).

3.2 Prueba de software que será aplicada al componente

3.2.1 Descripción de las pruebas para el nivel de Unidad

La prueba de unidad centra el proceso de verificación en la menor unidad del diseño del software: el módulo. Usando la descripción del diseño procedimental como guía, se prueban los caminos de control importantes, con el fin de descubrir errores dentro del límite del módulo (39).

3.2.2 Descripción y aplicación de la Prueba de Caja Blanca o Estructural

A este tipo de técnica se le conoce también como Técnicas de Caja Transparente o de Cristal. Es un método de diseño de casos de prueba se realiza atendiendo al comportamiento interno y la estructura del programa, examinando su lógica interna, sin considerar los aspectos de rendimiento.

Las pruebas de caja blanca intentan garantizar que:

- Se ejecutan al menos una vez todos los caminos independientes de cada módulo.
- Se utilizan las decisiones en su parte verdadera y en su parte falsa.
- Se ejecuten todos los bucles en sus límites.
- Se utilizan todas las estructuras de datos internas.

A continuación se citan algunas de las técnicas de prueba de caja blanca:

- Prueba de Condición.
- Prueba de Flujo de Datos.
- Prueba de Bucles.
- Prueba del Camino Básico.

La técnica que se utilizó fue la Prueba del Camino Básico, la misma permite obtener una medida de la complejidad de un diseño procedimental, y utilizar esta medida como guía para la definición de una serie de caminos básicos de ejecución, diseñando casos de prueba que garanticen que cada camino se ejecuta al menos una vez.

Para aplicar dicha técnica se debe introducir la notación para la representación del flujo de control, este puede representarse por un Grafo de Flujo en el cual:

- Cada nodo del grafo corresponde a una o más sentencias de código fuente.
- Todo segmento de código de cualquier programa se puede traducir a un Grafo de Flujo.
- Todo segmento de código de cualquier programa se puede traducir a un Grafo de Flujo.

Un grafo de flujo está formado por 3 componentes fundamentales que ayudan a su elaboración y comprensión, estos brindan información para confirmar que el trabajo se está haciendo adecuadamente.
Componentes del grafo de flujo:

- **Nodo:** son los círculos representados en el grafo de flujo, el cual representa una o más secuencias del procedimiento, donde un nodo corresponde a una secuencia de procesos o a una sentencia de decisión. Los nodos que no están asociados se utilizan al inicio y final del grafo.
- **Aristas:** son constituidas por las flechas del grafo, son iguales a las representadas en un diagrama de flujo y constituyen el flujo de control del procedimiento. Las aristas terminan en un nodo, aun cuando el nodo no representa la sentencia de un procedimiento.
- **Regiones:** son las áreas delimitadas por las aristas y nodos donde se incluye el área exterior del grafo, como una región más. Las regiones se enumeran siendo la cantidad de regiones equivalente a la cantidad de caminos independientes del conjunto básico de un procedimiento.

Para realizar la prueba del camino básico, es necesario calcular la complejidad ciclomática del algoritmo a analizar, esta es una medida que proporciona una idea de la complejidad lógica de un programa.
A continuación se enumeran las sentencias de código del procedimiento realizado al método get_producto (self, cr, uid, ids, id_producto), el cual tiene como función insertar en los campos cant_producto y precio_venta los valores concernientes a cantidad de productos y precio de venta del id_producto que se pase por parámetro.

```
def get_producto(self, cr, uid, ids, id_producto):
    if not id_producto: 1
        return {'value': {'cant_producto': False, 'precio_venta': False}} 2
    else: 3
        cr.execute('''SELECT 4
           producto_type.cant_real
       FROM
           public.contrato_type,
           public.producto_type
      WHERE
           %d = producto_type.id''' % (id_producto,))
        c_producto = int(cr.fetchone()[0]) 5

        cr.execute('''SELECT 6
             producto_type.precio_v
         FROM
             public.producto_type,
             public.contrato_type
         WHERE
         %d = producto_type.id;''' % (id_producto,))
        p_venta = int(cr.fetchone()[0])  7
        return {'value': {'cant_producto': c_producto, 'precio_venta': p_venta  }} 8
```

Figura 7. Código del algoritmo get_producto(self, cr, uid, ids, id_producto)

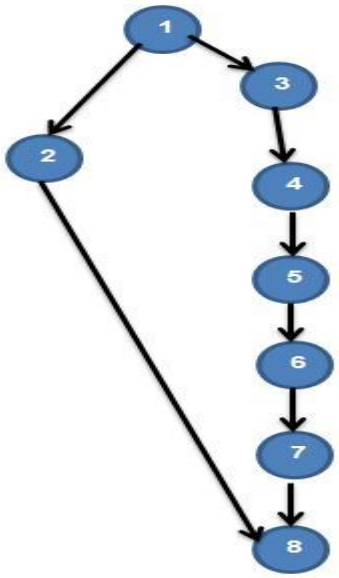

Figura 8. Grafo de flujo asociado al algoritmo get_producto(self, cr, uid, ids, id_producto)

Una vez construido el grafo de flujo se procede a calcular la complejidad ciclomática del algoritmo, la misma se haya mediante tres vías o fórmulas de manera tal que quede justificado el resultado, siendo el mismo en cada caso:

1. V (G) = (Aristas – Nodos) + 2
V (G) = (8-8) + 2
V (G) = 2

2. V (G) = P + 1
Siendo “P” la cantidad total de nodos predicados (son los nodos de los cuales parten dos o más aristas).
V (G) = 1+ 1
V (G) = 2

3. V (G) = R
Siendo “R” la cantidad total de regiones, se incluye el área exterior del grafo, contando como una región más.
V (G) = 2

El resultado del cálculo de la complejidad ciclomática del código fue 2, lo que significa que existen dos posibles caminos por donde el flujo puede circular, este valor representa el límite mínimo del número total de casos de pruebas para el procedimiento tratado.

Tabla 7. Caminos independientes

Número	Camino Básico
1	1-2-8
2	1-3-4-5-6-7-8

Después de haber identificado los caminos básicos se procede a realizar al menos un caso de prueba por cada camino básico. Para esto es necesario cumplir con lo siguiente:

- Descripción: se hace la entrada de datos necesaria, validando que ningún parámetro obligatorio pase nulo al procedimiento o se entre algún dato erróneo.

- Condición de ejecución: se especifica cada parámetro para que cumpla una condición deseada, para ver el funcionamiento del procedimiento.
- Entrada: se muestran los parámetros que entran al procedimiento.
- Resultados esperado: se expone resultado que se espera que devuelva el procedimiento.

Tabla 8. Posible representación de casos de prueba para pruebas estructurales

Número de Camino	Descripción	Entrada	Resultados Esperados
1	No se pasa la variable id_producto y se crea una tupla vacía.	Id_producto=vacío	No se completa los campos cantidad de producto(cant_prod) y precio de venta (precio_venta).
2	Se pasa la variable id_producto, se realizan las consultas a la base de datos y se obtienen los valores de cant_prod y precio_venta.	Id_producto=12	Se completan los campos correspondientes a la cant_prod y al precio_venta.

Luego de aplicar los distintos casos de prueba, se comprobó que el flujo de trabajo esta correcto cumpliendo con lo planteado.

3.2.3 Descripción y aplicación de la Prueba de Caja Negra o Funcional

A este tipo de prueba también se le conoce como Prueba de Caja Opaca o Inducida por los Datos. Las mismas se aplican a la interfaz del software, examinando el aspecto funcional del sistema.
Las pruebas de caja negra permiten encontrar:

- Funciones incorrectas o ausentes.
- Errores de interfaz.
- Errores en estructuras de datos o en accesos a las bases de datos externas.
- Errores de rendimiento.

- Errores de inicialización y terminación.

Algunas técnicas utilizadas en la prueba de caja negra son:

- **Partición de equivalencia:** técnica que divide el campo de entrada de un programa en clases de datos de los que se pueden derivar casos de prueba. La partición equivalente se dirige a una definición de casos de prueba que descubran clases de errores, reduciendo así el número total de casos de prueba que hay que desarrollar.
- **Análisis de valores límite:** la experiencia muestra que los casos de prueba que exploran las condiciones límite producen mejor resultado que aquellos que no lo hacen. Las condiciones límite son aquellas que se hallan en los márgenes de la clase de equivalencia, tanto de entrada como de salida. Por ello, se ha desarrollado el análisis de valores límite como técnica de prueba. Esta técnica lleva a elegir los casos de prueba que ejerciten los valores límite.
- **Grafos de causa-efecto:** es una técnica que permite al encargado de la prueba validar complejos conjuntos de acciones y condiciones.

A continuación se aplica la prueba de partición de equivalencia, como parte de la realización de la prueba de caja negra sobre la interfaz que responde al requisito funcional Gestionar cliente. La partición de equivalencia consiste en dividir el dominio de entrada de un programa en un conjunto finito de variables de equivalencia. Regularmente, una condición de entrada es un valor numérico específico, un rango de valores, un conjunto de valores relacionados o una condición lógica. Las variables de equivalencia se dividen en dos grupos, las válidas, que significa las entradas válidas al programa, y las no válidas, que como su nombre lo indica son las entradas erróneas al componente. Ver Anexo 3

Resultados de la prueba de caja negra

Para esta prueba se revisaron 24 requisitos del componente, realizándose una descripción de cada escenario de prueba. Durante la validación de las funcionalidades implementadas se realizaron 3 iteraciones cuyo resultado es el siguiente:

En la primera iteración se encontraron un total de 17 No Conformidades (NC):

- Gestionar Cliente 4 NC.
- Gestionar Oferta 3 NC.
- Gestionar Pedido de venta 4 NC.

- Gestionar Contrato 6 NC.

En la segunda iteración se encontraron un total de 6 (NC):

- Gestionar Cliente 2 NC.
- Gestionar Oferta 1 NC.
- Gestionar Pedido de venta 1 NC.
- Gestionar Contrato 2 NC.

En la tercera iteración no se encontraron (NC):

- Gestionar Cliente 0 NC.
- Gestionar Oferta 0 NC.
- Gestionar Pedido de venta 0 NC.
- Gestionar Contrato 0 NC.

En la siguiente gráfica se muestran las no conformidades detectadas durante la realización de las pruebas de caja negra. En las mismas se encontraron mayormente errores de faltas de ortografías en las interfaces, problemas con las validaciones de los campos y en los mensajes de información, los mismos se encontraban mal elaborados.

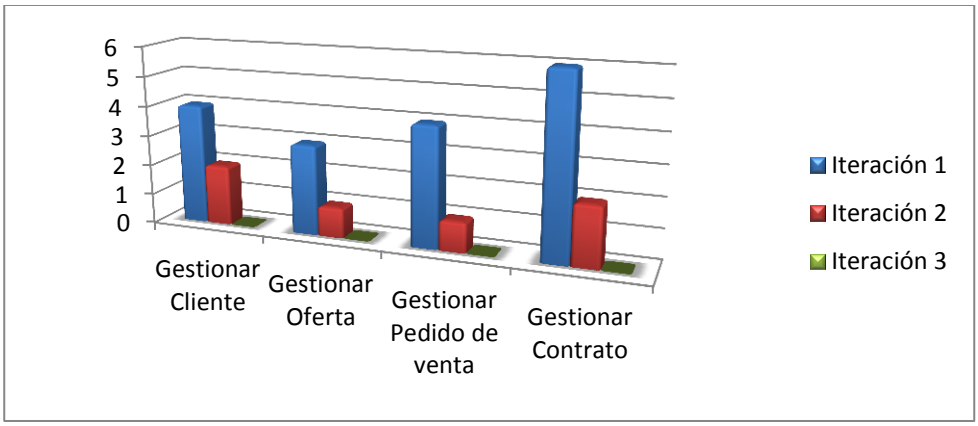

Figura 9. Resultado de las iteraciones de la prueba de caja negra

Conclusiones parciales

En este capítulo se abordaron los métodos de pruebas utilizados en la validación de la solución propuesta. Los resultados de las mismas ayudaron a identificar errores que habían pasado inadvertidos en la etapa de implementación. Las pruebas de caja blanca permitieron conocer la complejidad ciclomática de cada funcionalidad, mientras que las pruebas de caja negra probaron las interfaces que se utilizarán para la Administración de Ventas en una empresa.

CONCLUSIONES GENERALES

La investigación realizada, los resultados y las validaciones obtenidas permitieron el cumplimiento de los objetivos propuestos, arribando a las siguientes conclusiones:

- Con el estudio de diferentes herramientas CRM se pudo concluir que estas presentan ventajas y desventajas de acuerdo a los parámetros medidos, siendo el OpenERP el sistema adecuado para realizar la implementación del componente Administración de Ventas.
- El componente fue implementado utilizando herramientas, lenguajes y tecnologías distribuidas bajo licencias de software libre en correspondencia con el principio de independencia tecnológica en el cual está basado actualmente el desarrollo de software en Cuba.
- Se valoró críticamente el diseño propuesto y fue necesario la inclusión de nuevas clases y atributos.
- El componente elaborado fue expuesto a diferentes pruebas, las cuales arrojaron resultados satisfactorios y el cumplimiento de los objetivos propuestos.

RECOMENDACIONES

Para el completamiento del proceso Administración de Ventas se recomienda:

- Implementar la integración con el componente de Facturación de Cedrux.
- Utilizar este documento para investigaciones futuras y para el trabajo en el proyecto.
- Migrar el componente realizado a la versión 7.0.

REFERENCIAS BIBLIOGRÁFICAS

1. [En línea] [Citado el: 24 de 10 de 2012.] http://e-ngenium.blogspot.com/2011/11/historia-de-los-sistemas-erp.html.

2. Cubadebate. [En línea] [Citado el: 28 de 10 de 2012.]

http://www.cubadebate.cu/especiales/2012/09/21/el-bloqueo-es-el-principal-obstaculo-para-que-cuba-desarrolle-a-plenitud-sus-potencialidades.

3. [En línea] [Citado el: 14 de 11 de 2012.] http://definicion.de/venta/.

4. Thompson Ivan. [En línea] [Citado el: 16 de 11 de 2012.] http://www.rae.es/.

5. Kotler Philip. [En línea] [Citado el: 16 de 11 de 2012.]

http://www.webandmacros.com/crm.htm.

6. [En línea] [Citado el: 19 de 11 de 2012.] www.elegirerp.com/ERP-Definicion.php.

7. Carmona, Laura Díaz. Sistema CRM de código abierto: SUGARCRM . Madrid : s.n., 2011.

8. [En línea] [Citado el: 19 de 11 de 2012.] http://www.sugarcrm.com.

9. [En línea] [Citado el: 19 de 11 de 2012.] http://www.informatica-hoy.com.ar/software-crm/SAP-CRM.php.

10. [En línea] [Citado el: 28 de Enero de 2013.] http://fadata.bg/es/CRM.

11. Microsoft Dynamics. [En línea] 2011. [Citado el: 21 de 11 de 2011.]

http://www.microsoft.com/latam/dynamics/crm/datasheet/default.aspx.

12. [En línea] [Citado el: 23 de 11 de 2012.]

http://www.excalesolutions.com/servicios/desarrollo-implementacion-formacion-openerp.

13. [En línea] [Citado el: 23 de 11 de 2012.]

http://www.cybercia.com/index.php?option=com_content&view=article&id=52&Itemid=71#1.2..

14. Obregón, Ing. William González. Modelo de desarrollo de software. Habana : s.n., 2012.

15. Alvarez, Miguel Angel. Lenguaje de programación de propósito general, orientado a objetos, que también puede utilizarse para el desarrollo web. [En línea] [Citado el: 21 de Enero de 2013.] http://www.desarrolloweb.com/articulos/1325.php.

16. Montalvo, Marlene Melián. XML el nuevo lenguaje universal. 2010.

17. Bermejo Laura Sanz, Gómez Enrique Monreal. [En línea] [Citado el: 19 de 11 de 2012.]

http://kybele.escet.urjc.es/documentos/HC/Exposiciones/EclipseIDE.pdf.

18. [En línea] [Citado el: 2013 de Enero de 25.] http://www.softpedia.es/programa-Pydev-192265.html.

19. Borrás, Jose Ramón Terol. Implantación de OpenERP y programación de un conector con básculas MAPAL. Valencia : s.n., 2010.

20. PostgreSQL-es . [En línea] 02 de 10 de 2010. [Citado el: 25 de Enero de 2013.] http://www.postgresql.org.es/sobre_postgresql.

21. Visconti Marcello, Astudillo Hernán. Fundamentos de Ingeniería de Software. Valparaíso : Universidad Técnica Federico Santa María, 2004.

22. Sebastián, Juan. [En línea] 13 de Noviembre de 2010. [Citado el: 21 de Enero de 2013.] http://www.comusoft.com/modelo-vista-controlador-definicion-y-caracteristicas.

23. Pinckaers Fabien, Gardiner Geoff, Van Vossel Els. Open Object Developer Book. 2010.

24. [En línea] [Citado el: 2013 de Febrero de 2.]

http://www.calidadysoftware.com/testing/pruebas_unitarias1.php.

25. [En línea] [Citado el: 2013 de febrero de 2.]
http://www.calidadysoftware.com/testing/pruebas_funcionales.php.

26. Prácticas del Software. [En línea] 16 de 11 de 2012. [Citado el: 28 de 3 de 2013.]
http://www.practicasdesoftware.com.ar/tag/ieee/.

27. Sosa, Angel Gabriel Olivera. Requisitos funcionales y no funcionales. Campeche : s.n., 2010.

28. Acosta Johan David Cortes, Huertas Walter Díaz, Wbeimar Wilder Díaz Santa. Diagrama de Componentes. 2009.

29. Huallpara, Hugo Michael Marca y Limachi, Nancy Susana Quisbert. diagrama de despliegue. 2012.

30. FileMaker, Compañía. Guía ODBC y JDBC. 2010.

31. Guijarro, Álvaro Primo. Protocolo HTTP. 2012.

32. Quinodóz, Carolina. Protocolo FTP . 2010.

33. www.masadelante.com. [En línea] [Citado el: 1 de abril de 2013.]
http://www.masadelante.com/faqs/tcp-ip.

34. [En línea] [Citado el: 13 de 5 de 2013.] http://sg.com.mx/content/view/467.

35. [En línea] 02 de 2009. [Citado el: 11 de 5 de 2013.] http://informaticaitc.blogspot.com/2009/02/importancia-de-la-calidad.html.

36. it-Mentor. Pruebas de Software. 2013.

37. Galeon.com. [En línea] [Citado el: 13 de 5 de 2013.] http://www.galeon.com/neoprogramadores/met_test.htm.

38. [En línea] [Citado el: 13 de 5 de 2013.] http://sg.com.mx/content/view/467.

39. Angelfire. [En línea] [Citado el: 12 de 5 de 2013.] http://www.angelfire.com/my/jimena/ingsoft/guia10.htm.

GLOSARIO DE TÉRMINOS

CapWords: es un estilo donde la primera letra de cada palabra comienza con mayúscula y el resto con minúscula, es una variante del estilo CamelCasing.

DBMS: Sistema de Gestión de Bases de Datos es un conjunto de programas que permiten el almacenamiento, modificación y extracción de la información en una base de datos, además de proporcionar herramientas para añadir, borrar modificar y analizar los datos. Los usuarios pueden acceder a la información usando herramientas específicas de interrogación y de generación de informes, o bien mediante aplicaciones al efecto.

HTML (siglas del inglés Hypertext Markup Language): hace referencia al lenguaje de marcado predominante para la elaboración de páginas web que se utiliza para describir y traducir la estructura y la información en forma de texto, así como para complementar el texto con objetos tales como imágenes.

IDE (siglas del inglés Integrated Development Environment): es un programa informático compuesto por un conjunto de herramientas de programación. Puede dedicarse en exclusiva a un solo lenguaje de programación o bien puede utilizarse para varios.

Licencia BSD: es la licencia de software otorgada principalmente para los sistemas BSD (Berkeley Software Distribution). Es una licencia de software libre permisiva. Esta licencia tiene menos restricciones en comparación con otras como la GPL estando muy cercana al dominio público. La licencia BSD al contrario que la GPL permite el uso del código fuente en software no libre.

Licencia GPL: la Licencia Pública General de GNU (GNU GPL, por sus siglas en inglés) es una licencia libre y gratuita con derecho de copia para software y otros tipos de obras.

Pydev: es un entorno de desarrollo integrado para Python basado en Eclipse.

PDF (siglas del inglés portable document format): es un formato de almacenamiento de documentos digitales independiente de plataformas de software o hardware.

Proxies: en una red informática, es un programa o dispositivo que realiza una acción en representación de otro.

SOAP (siglas del inglés Simple Object Access Protocol): es un protocolo estándar que define cómo dos objetos en diferentes procesos pueden comunicarse por medio de intercambio de datos XML.

SQL (por sus siglas en inglés structured query language): es un lenguaje declarativo de acceso a bases de datos relacionales que permite especificar diversos tipos de operaciones en ellas.

SSL (siglas del inglés Secure Sockets Layer**):** es un protocolo criptográfico que proporciona comunicaciones seguras por una red, comúnmente Internet.

Unicode: es un estándar de codificación que especifica un nombre e identificador numérico único para cada carácter o símbolo.

XML-RPC: es un protocolo de llamada a procedimiento remoto que usa XML para codificar los datos y HTTP como protocolo de transmisión de mensajes.

ANEXOS

Anexo 1. Interfaces del componente Gestión de Ventas

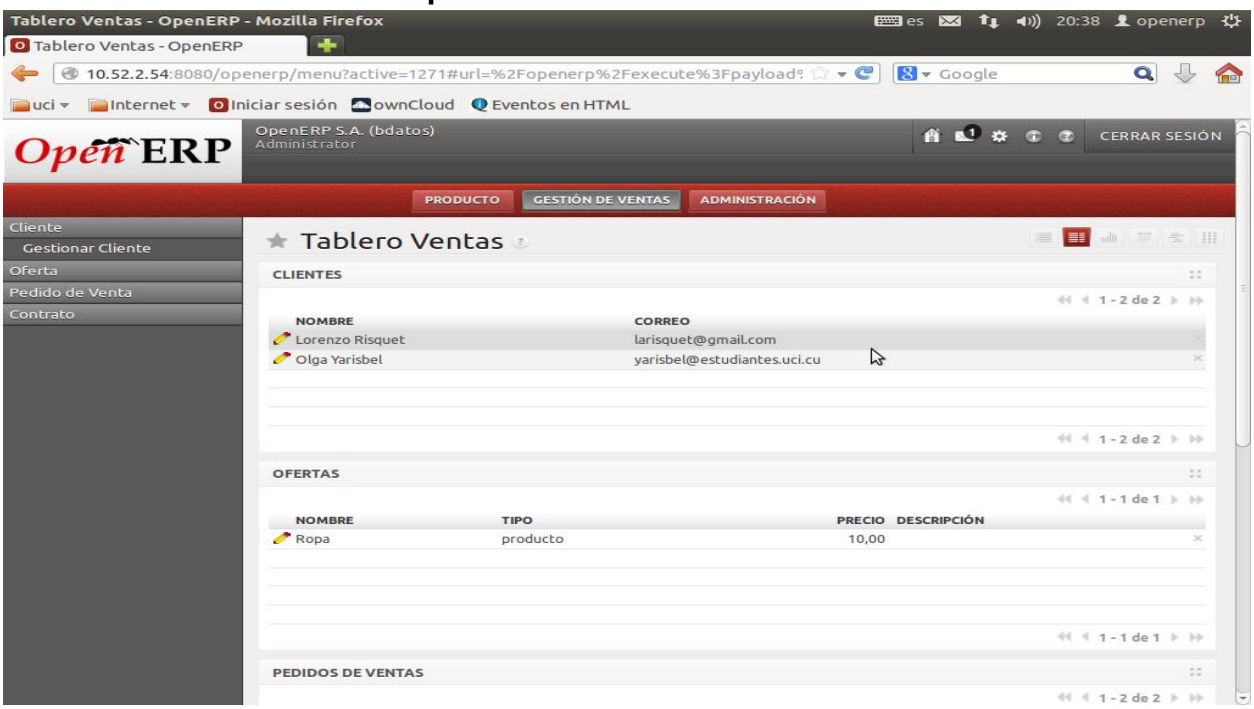

Figura 10. Interfaz Principal del componente Gestión de Ventas

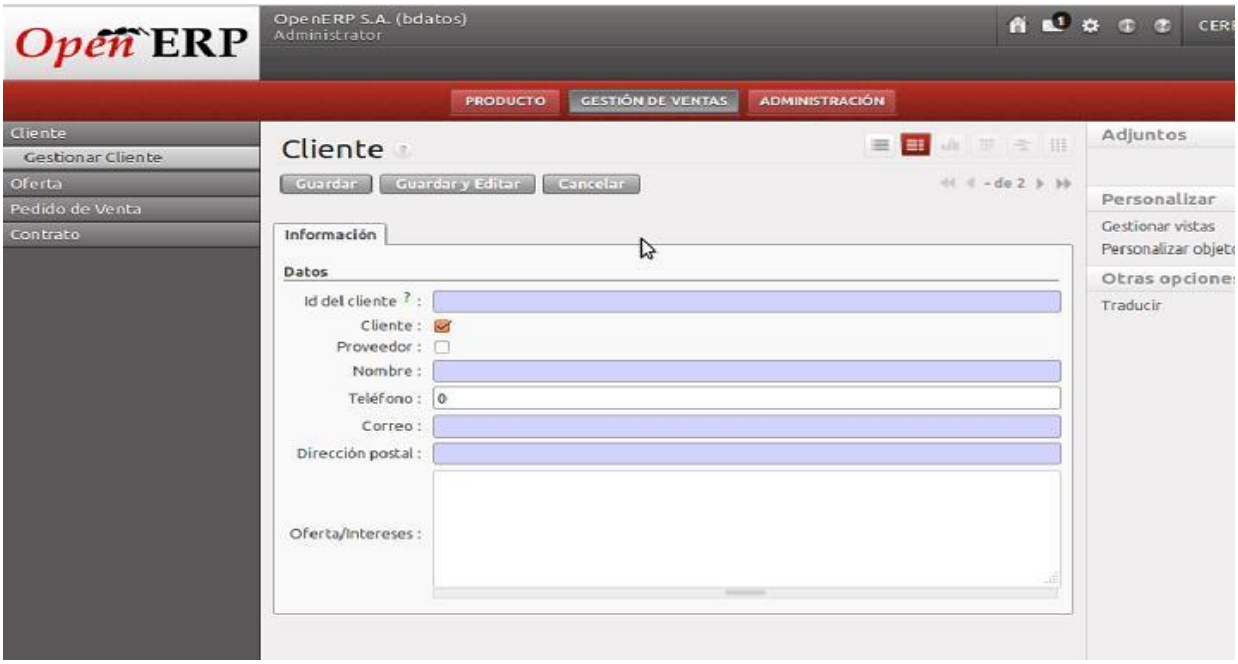

Figura 11. Interfaz del Gestionar Cliente, vista formulario

Figura 12. Interfaz del Gestionar Cliente, vista lista

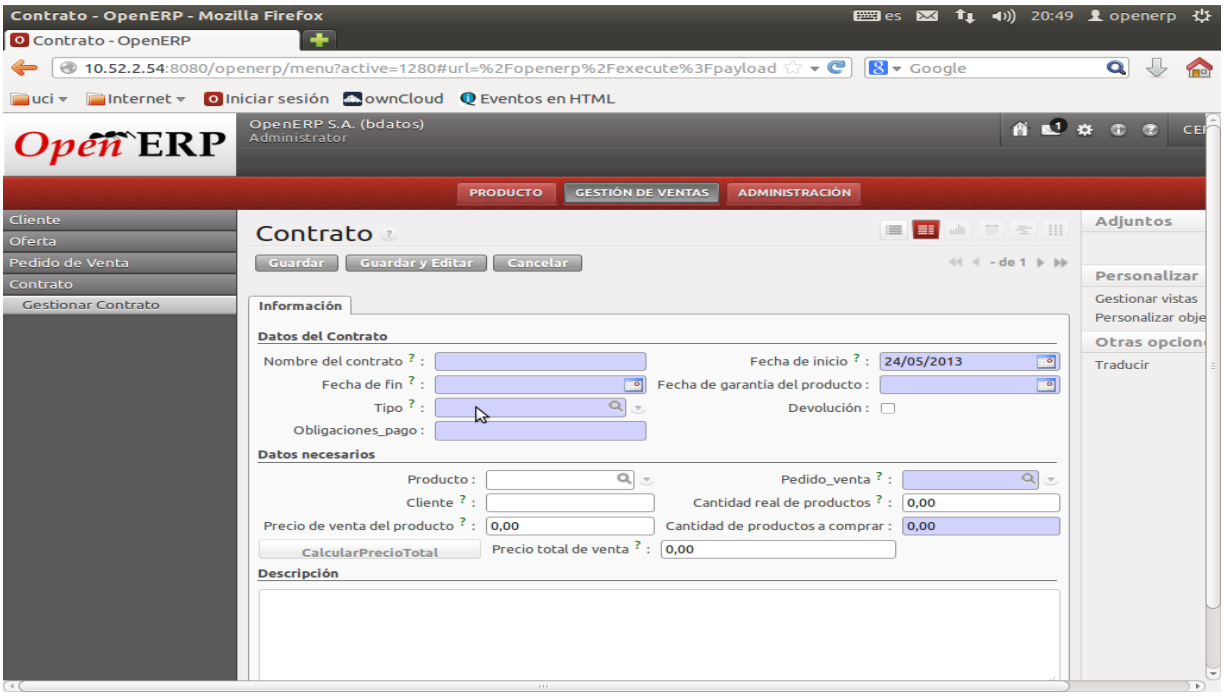

Figura 13. Interfaz del Gestionar Contrato, vista formulario

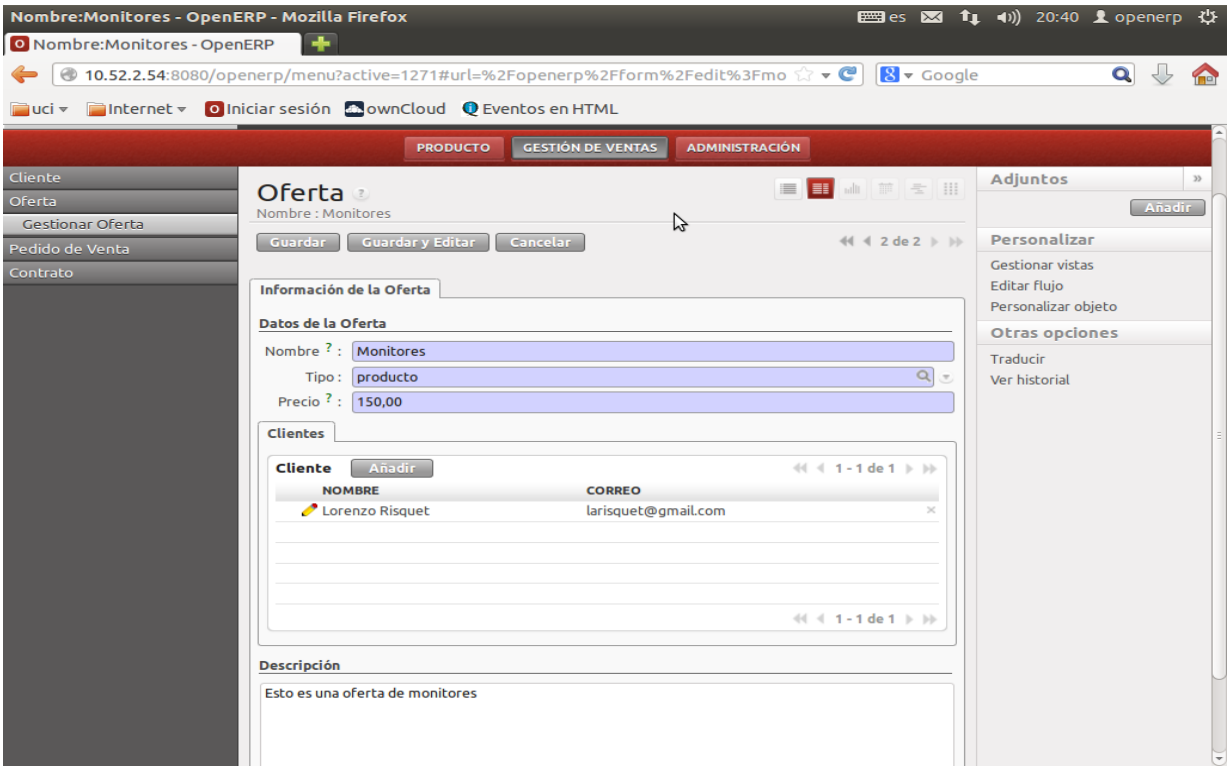

Figura 14. Interfaz del Gestionar Oferta, vista formulario

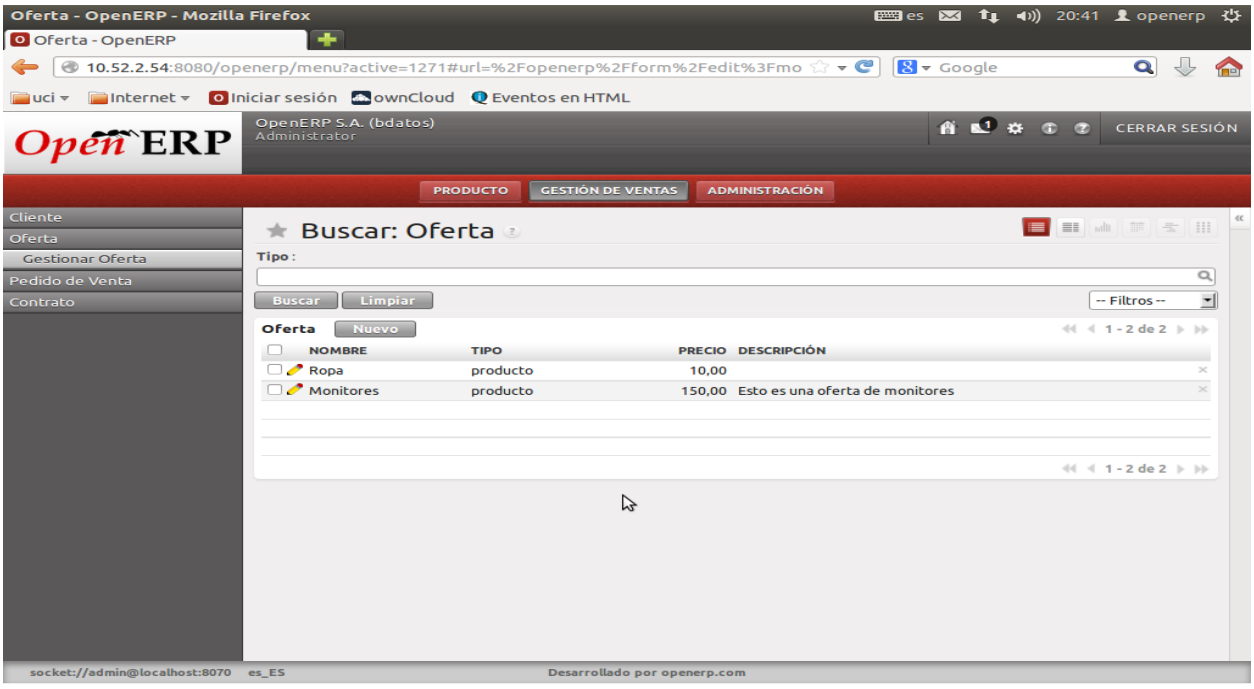

Figura 15. Interfaz del Gestionar Oferta, vista lista

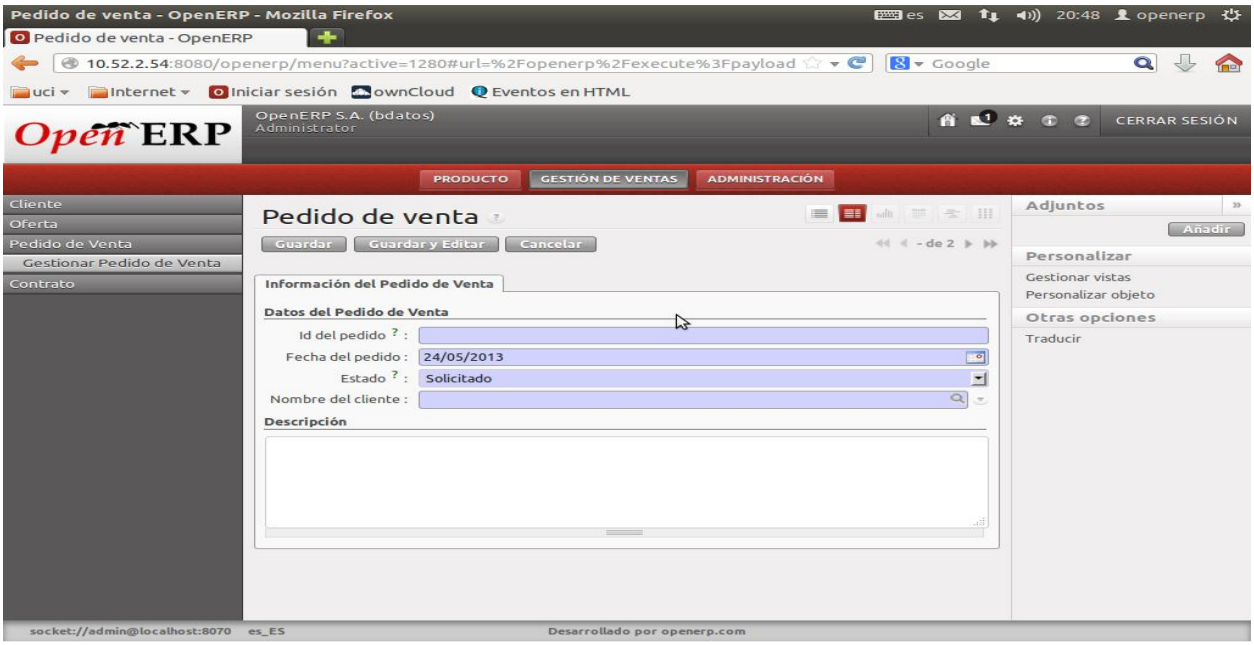

Figura 16. Interfaz del Gestionar Pedido de venta, vista formulario

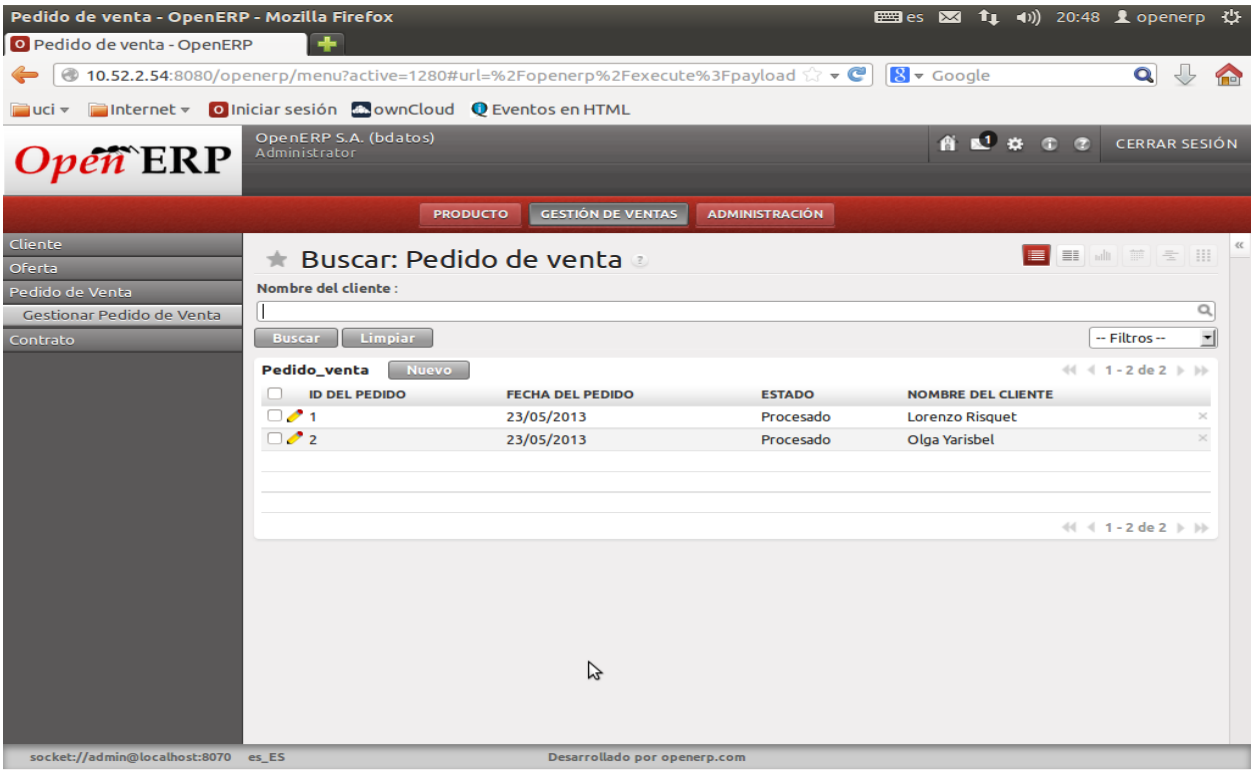

Figura 17. Interfaz del Gestionar Pedido de venta, vista lista.

Anexo 2. Descripción de clases y métodos

Tabla 9. Descripción de la clase Nomenclador

Nombre: Nomenclador_type		
Descripción: La tabla va a recoger los datos del nomenclador tipo de oferta y contrato		
Atributo	**Tipo**	**Descripción**
codigo	char	Identificador de la clase Nomenclador
tipo	char	Tipo de nomenclador, puede ser servicio ó producto

Tabla 10. Descripción de la clase Pedido_venta

Nombre: Pedido_venta_type		
Descripción: La tabla va a recoger los datos del pedido de venta		
Atributo	**Tipo**	**Descripción**
name	char	Identificador de la clase Pedido_venta
fecha_pedido	date	Fecha del pedido de venta
estado	char	Estado del pedido de venta, puede ser solicitado ó procesado
descripción	text	Descripción del pedido de venta

Tabla 11. Descripción de las funcionalidades de la clase Cliente

Clase Cliente	
Nombre de las funcionalidades	**Descripción**
def check_idcliente(self, cr, uid, ids, context=None):	Método para chequear el tipo de datos que se inserte en el atributo Id del cliente
def chekear_email(self, cr, uid, ids, context=None):	Método para chequear el tipo de datos, y la estructura que se inserte en el atributo email
def check_nombre(self, cr, uid, ids, context=None):	Método para chequear el tipo de datos, y la estructura que se inserte en el atributo Nombre

Tabla 12. Descripción de las funcionalidades de la clase Nomencladora

Clase Nomenclador	
Nombre de las funcionalidades	**Descripción**
def _autompletar_codigo(self, cr, uid, ids, context=None):	Método para autocompletar el id del nomenclador

Tabla 13. Descripción de las funcionalidades de la clase Oferta

Clase Oferta	
Nombre de las funcionalidades	**Descripción**
def check_id_oferta(self, cr, uid, ids, context=None):	Método para chequear el tipo de datos del atributo nombre

Tabla 14. Descripción de las funcionalidades de la clase Pedido de venta

Clase Pedido_venta	
Nombre de las funcionalidades	**Descripción**
def check_id_pedido_venta(self, cr, uid, ids, context=None):	Método para chequear el tipo de datos del atributo Id del pedido

Tabla 15. Descripción de las funcionalidades de la clase Contrato

Clase Contrato	
Nombre de las funcionalidades	**Descripción**
def check_nombre(self, cr, uid, ids, context=None):	Método para chequear el tipo de datos, y la estructura que se inserte en el atributo Nombre
def check_fechas(self, cr, uid, ids, context=None):	Método que verifica que la fecha del fin del contrato es mayor que la fecha de inicio del contrato
def check_garantia(self, cr, uid, ids, context=None):	Método que verifica que la fecha de garantía del contrato es mayor que la fecha de fin del contrato
def get_cliente(self, cr, uid, ids, id_pedido):	Método que autocompleta el campo nombre del cliente de acuerdo al Id del pedido de venta que se inserte
def get_producto(self, cr, uid, ids, id_producto):	Método que autocompleta los campos cantidad de productos y precio del producto de acuerdo al Id del producto que se inserte
def get_precio(self, cr, uid, ids, context=None):	Método que calcula el precio de venta final de los productos que se adquieren en el contrato

Anexo 3: Caso de prueba de caja negra para validar el requisito Eliminar cliente

Tabla16. Escenario de prueba.

Nombre del requisito	Descripción general	Escenarios de pruebas	Flujo del escenario
1: Eliminar Cliente.	El componente debe permitir eliminar cliente.	EP 1.1: Eliminar cliente.	– Se muestra un mensaje confirmando si realmente se desea eliminar el cliente. – Se presiona el botón **Aceptar**.
		EP 1.2: Cancelar.	– Se muestra un mensaje confirmando si realmente se desea eliminar el cliente. – Se presiona el botón Cancelar.

Tabla 17. Descripción de variable

No	Nombre de campo	Tipo	Válido	Inválido	Inválido	Inválido	Inválido
1	id	Integer	Integer	Letras	Vacío		
2	proveedor	booleano	booleano	NA	NA		
3	cliente	booleano	booleano	NA	NA		

4	name	char	char	Números	Vacío
5	telef	integer	Números	Letras, caracteres especiales	Vacío
6	email	char	Letras, (.,@), números	Caracteres especiales	Vacío
7	direc_post	char	char	Caracteres especiales	Vacío
8	oferta	text	text	NA	NA

Tabla 18. Juegos de datos a probar

Id del escenario	Escenario	Respuesta del componente	Resultado de la prueba
EP 1.1	Eliminar cliente.	El componente muestra el mensaje de confirmación: "¿Realmente desea eliminar este registro?". El componente elimina el registro	NA
EP 1.2	Cancelar.	El componente cierra la interfaz sin realizar ninguna operación.	NA

ANEXO 4: Aval de aprobación del departamento SOLEM

Universidad de las Ciencias Informáticas

SISTEMA DE ADMINISTRACIÓN DE RELACIONES CON EL CLIENTE

Por el presente damos constancia de que todas las funcionalidades del componente Ventas para el Sistema de Administración de Relaciones con el Cliente fueron implementadas cumpliendo todas las necesidades plasmadas en los requisitos funcionales.

Este componente concebido para garantizar la gestión de las ventas, está desarrollado sobre tecnologías libres, por tanto responde a las políticas de software libre dictadas en la Universidad de las Ciencias Informáticas. Durante la ejecución de la tesis con título "Implementación del componente Administración de Ventas para el sistema de Administración de Relaciones con el Cliente" se actualizaron los artefactos ya realizados que sufrieron algún tipo de modificación.

La utilización de este componente permite la gestión de las ofertas, los contratos, los clientes y los pedidos de ventas.

Y para constancia de ello firman el presente aval a los 5 días del mes de Junio del 2013:

Ing. Olga Yarisbel Rojas Grass
Jefe de línea

Centro de Informatización de la Gestión de Entidades
CEIGE

Ing. Erich Mario Gómez Pérez
Jefe de departamento