Universidad de las Ciencias Informáticas

Facultad 15

Diseño e implementación del módulo Banco del Sistema Integral de Gestión CEDRUX

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

Autor:

Taimé Ramos Arias

Pedro Antonio Torres Salas

Tutores:

Ing. Yoan Arlet Carrascoso Puebla

Ing. Lianet Guevara Álvarez

Ciudad de la Habana, junio de 2010.

"Año 52 de la Revolución"

DECLARACIÓN DE AUTORÍA

Declaramos que somos los únicos autores del trabajo titulado: Diseño e implementación del módulo Banco del Sistema Integral de Gestión CEDRUX; y autorizamos a la Universidad de las Ciencias Informáticas los derechos patrimoniales del mismo, con carácter exclusivo.

Para que así conste firmamos la presente a los ____ días del mes de junio del año 2010.

_____

Taimé Ramos Arias
Autor

_____

Pedro Antonio Torres Salas
Autor

_____

Ing. Yoan Arlet Carrascoso Puebla
Tutor

_____

Ing. Lianet Guevara Álvarez
Tutor

"…aquí está una de las tareas de la juventud: empujar, dirigir con el ejemplo la producción del hombre de mañana. Y en esta producción, en esta dirección, está comprendida la producción de sí mismos…"

Che

RESUMEN

En la actualidad, las organizaciones se ven en la necesidad de transmitir información de alta calidad que esté disponible en el momento oportuno y agilizar los procesos económicos basándose en las tecnologías y la información para mejorar la toma de decisiones. En Cuba, muchas empresas optan por el uso de sistemas de gestión empresarial; sin embargo, estos sistemas no responden totalmente a las funcionalidades requeridas por las entidades para el óptimo control de sus procesos empresariales.

Dentro de estos procesos se encuentra la gestión bancaria, que incluye el control de las cuentas bancarias de la entidad así como la emisión de instrumentos bancarios y las operaciones generadas a partir de los estados de cuenta y las conciliaciones.

El presente trabajo tiene como objetivo realizar el diseño e implementación del módulo Banco del subsistema Finanzas del Sistema Integral de Gestión CEDRUX, partiendo del estudio de las tendencias modernas y buenas prácticas en el desarrollo de aplicaciones web. Durante la investigación se describen las etapas del diseño para el módulo siguiendo el modelo de desarrollo orientado a componentes establecido por el equipo de arquitectura y se definen las herramientas a emplear para implementar los componentes.

El sistema garantizará el buen funcionamiento de los procesos bancarios en las entidades cubanas, cumpliendo con las legislaciones financieras actuales y constituyendo un aporte decisivo a la informatización y economía del país.

Palabras clave: CEDRUX, sistemas de gestión empresarial, gestión bancaria.

ÍNDICE DE CONTENIDO

INTRODUCCIÓN

El campo de las Tecnologías de la Información y las Comunicaciones (TIC) ha alcanzado un auge mundial sin precedentes, influyendo en las estrategias de negocios y en la forma en que las entidades realizan sus procesos. Actualmente, las empresas requieren de sistemas que les proporcionen control y centralización de la información con el fin de tomar mejores decisiones. Por esta razón, surgen los Sistemas de Planificación de Recursos Empresariales (ERP: Enterprises Resources Planning), para lograr alcanzar la optimización de procesos en respaldo a estrategias competitivas, como eslabón fundamental de la integración de la información en las diferentes áreas o departamentos de las entidades.

Hasta el momento, en Cuba existe gran variedad de paquetes de software certificados que facilitan los procesos empresariales: VERSAT-Sarasola, RODAS XXI, SISCONT5, CONDOR, SAP y otros; sin embargo, no se ha logrado obtener una herramienta estándar que beneficie a las entidades cubanas e incluya funcionalidades necesarias afines a las particularidades de la economía del país. Además, la implantación de estas soluciones puede resultar muy costosa tanto en tiempo como en dinero, incluso una vez establecido el sistema, los costos de los cambios son muy altos, reduciendo la flexibilidad y las estrategias de control. La mayoría de estos software son propietarios, lo que implica gastos en pagos de licencias anuales, y no se pueden ejecutar sobre diversos sistemas operativos.

A partir de este enfoque, el país se traza la tarea de desarrollar un sistema de gestión empresarial acorde a las necesidades reales de las empresas cubanas. La Universidad de las Ciencias Informáticas en coordinación con el Ministerio de Finanzas y Precios, las entidades DESOFT y TEICO Villa Clara, desarrollarán el Sistema Integral de Gestión CEDRUX con el objetivo de satisfacer todos los requerimientos funcionales y técnicos que estén a la altura de las mejores soluciones de este tipo a nivel mundial. La aplicación debe vencer las barreras de software existentes para migrar hacia las nuevas posibilidades que brinda el Software Libre. De manera general, el sistema constituye una solución completa a todas las áreas de las entidades que permite optimizar los procesos empresariales garantizando la eficiencia organizacional del país.

Precisamente, entre las actividades del negocio imprescindibles a automatizar en el sistema se encuentra la gestión bancaria. En ocasiones, este proceso se realiza en las entidades cubanas de forma insuficiente; las empresas efectúan movimientos de entrada y salida de dinero sin tener efectivo que las respalde y

violan las legislaciones financieras establecidas. Por tanto, para evitar estas imprecisiones contables, se necesita mejorar el control de las operaciones bancarias que permita gestionar operaciones automáticas, cuentas bancarias, pagos anticipados o por cheques y el análisis del flujo del capital contable de la entidad.

Actualmente, para la gestión de estos procesos financieros, las entidades cubanas tienden al uso de aplicaciones extranjeras que no poseen óptima correspondencia y adaptabilidad a las nuevas legislaciones nacionales emitidas. Las soluciones bancarias de estas aplicaciones se caracterizan por tener poca flexibilidad ante la dualidad monetaria, la multimoneda y la multientidad, lo que impide llevar registros independientes de cálculos complejos, de conversiones de monedas y el control administrativo de las entidades subordinadas. El deficiente soporte técnico afecta la fiabilidad y el intercambio ordenado de información para el proceso de toma de decisiones respecto a temas financieros e informáticos. Los sistemas no se basan en los principios de independencia tecnológica ni permiten tomar decisiones administrativas con rapidez. Además, se dificulta la integración de los procesos contables producto de la heterogeneidad de sistemas existentes que realizan las operaciones de diversas maneras. En algunas empresas no se dispone de dichos sistemas, las actividades todavía se llevan de forma manual utilizando como soporte el papel, lo cual provoca pérdida de datos, baja confidencialidad de los mismos, lentitud y elevada probabilidad de cometer errores al realizar los procesos.

Debido a estos inconvenientes, se precisa optimizar la gestión bancaria en las entidades del país garantizando la calidad de las operaciones y el cumplimiento de las políticas y especificaciones normativas y legislativas; derivando en la mejora continúa de una solución bien estructurada que asegure la integración con otras áreas del negocio financiero.

A partir de la situación problémica planteada, se define el siguiente **problema a resolver**: Los sistemas que controlan la gestión bancaria empresarial en Cuba, no cumplen en su totalidad con las funcionalidades necesarias y las recientes legislaciones financieras, lo que imposibilita la toma de decisiones estratégicas de las entidades cubanas.

En tal sentido, el **objeto de estudio** está centrado en los procesos contables de la gestión bancaria. Delimitando como **campo de acción** el diseño e implementación de los procesos contables de la gestión bancaria.

Se plantea como **objetivo general** para solucionar el problema: Realizar el diseño e implementación del módulo Banco del Sistema Integral de Gestión CEDRUX para gestionar las actividades bancarias de las entidades, que se adapte a las legislaciones financieras actuales y cumpla con los requerimientos del proceso de informatización en el país.

Para dar cumplimiento al objetivo propuesto, se proponen los siguientes **objetivos específicos:**

- Analizar procesos financieros, herramientas, tecnologías y sistemas informáticos vinculados a la gestión bancaria en las entidades.
- Definir la solución de diseño e implementación de los componentes del módulo cumpliendo con los requerimientos especificados en el análisis.
- Validar las funcionalidades del módulo.

Se definen como **tareas investigativas**:

- Realizar el marco teórico conceptual de la investigación a partir del análisis de los procesos bancarios de una entidad, los logros y las limitaciones en los enfoques existentes.
- Investigar los patrones de diseño, estándares de codificación, herramientas y tecnologías definidos por la Línea de Arquitectura en el Marco de Trabajo.
- Analizar los artefactos generados durante la etapa de análisis.
- Diseñar e implementar las clases por componentes del módulo.
- Evaluar el diseño propuesto y la implementación del módulo.

Se plantea como **idea a defender** que: Con el diseño e implementación del módulo Banco del Sistema Integral de Gestión CEDRUX se espera obtener un producto funcional que controle todos los procesos contables de la gestión bancaria empresarial y cumpla con las resoluciones financieras vigentes.

En cumplimiento a las distintas tareas antes mencionadas, se pusieron en práctica los siguientes **métodos de investigación:**

Métodos teóricos:

- **Analítico – sintético:** Posibilitando procesar toda la información enfocada hacia la investigación de los procesos bancarios, permitiendo organizar y simplificar el análisis de todo el volumen de datos a recopilar en fracciones factibles.

- **Histórico – lógico**: Para conocer los antecedentes y tendencias actuales en los ERP, procesos financieros, las plataformas y lenguajes de implementación.

Métodos empíricos:

- **Experimento**: Favoreciendo el desarrollo de pruebas para la verificación de las funcionalidades implementadas, con el fin de detectar errores y comprobar el correcto funcionamiento del negocio.
- **Observación**: Para percibir y planificar como quedaría concebido el sistema.

Como **posible resultado** se espera obtener el módulo Banco del Sistema Integral de Gestión CEDRUX totalmente funcional, que cumpla con los requerimientos descritos, e integrado con otros subsistemas que son necesarios para su correcto funcionamiento y flujo de información; permitiendo realizar todas las operaciones contables de forma ágil.

La investigación se estructura de la siguiente manera: Introducción, Tres Capítulos, Conclusiones, Recomendaciones, Bibliografía y Glosario de Términos.

Capítulo 1: Se expone el estudio del estado del arte, donde se realiza la fundamentación teórica de los procesos financieros bancarios. Al mismo tiempo, se describe el objeto de estudio y los procesos fundamentales de la gestión bancaria. También se mencionan y describen algunos sistemas contables existentes. Finalmente, se justifican las tendencias y tecnologías utilizadas para el modelado de la aplicación y otras necesarias para su futura implementación.

Capítulo 2: Se realiza el análisis de la arquitectura de software definida en el proyecto, a partir del uso de patrones de diseño y arquitectónicos. Se valoran los artefactos obtenidos durante análisis que fueron entregados por los analistas, definiendo los componentes que agrupan las funcionalidades necesarias. Se especifican las principales estructuras de datos y estándares de codificación usados y se describen los algoritmos no triviales implementados, así como las principales clases que se utilizaron.

Capitulo 3: Se evalúa el diseño propuesto empleando métricas de software. Se aplican pruebas resaltando la importancia de las mismas en el desarrollo del proyecto. Se aborda acerca de las pruebas de menor escala tales como pruebas de caja blanca y caja negra y las particularidades de cada una de ellas, analizando los resultados obtenidos. Se realiza un análisis de los algoritmos implementados en el módulo, con el objetivo de demostrar la eficiencia de los mismos.

CAPÍTULO I: Fundamentación Teórica

1.1 Introducción

El desarrollo acelerado y el ambiente competitivo del mundo empresarial actual hacen imposible que las entidades puedan controlar óptimamente sus actividades financieras. De ahí que la mayoría de las empresas utilicen tecnologías que permitan agilizar el trabajo de manera eficiente y con el personal necesario.

Por tanto, en este capítulo se realiza el estudio de los procesos financieros bancarios que se desarrollan en las entidades cubanas, así como los sistemas existentes hasta el momento en el país que efectúan estas actividades. Además, se definen las herramientas y las metodologías de desarrollo a emplear en el l diseño y la implementación del sistema, acorde a las peculiaridades de la economía cubana y al proceso evolutivo de las tecnologías que enmarca el desarrollo de software.

1.2 Procesos de Gestión Bancaria

Los procesos contables y financieros definen las actividades que realiza una entidad para registrar hechos y transacciones que ocurren en un momento determinado. Estos, a su vez, contienen la gestión bancaria que se encarga del control de la actividad bancaria de la entidad y realiza como proceso fundamental, el registro y control de cheques evitando con ello la desviación de recursos. Incluye la gestión y seguimiento de las cuentas bancarias, los talonarios de instrumentos bancarios asociados a cada una de ellas y los instrumentos bancarios emitidos para realizar las diferentes operaciones de pago, ya sea liquidaciones de obligaciones, pagos anticipados y reembolsos. Garantiza que las operaciones bancarias automáticas, de cobros y pagos, permitan el reconocimiento de cobros anticipados y liquidación de derechos.

Además, la gestión bancaria realiza el análisis del flujo del capital contable de la entidad y la contabilización de los procesos mediante los estados de cuenta y las conciliaciones bancarias, emitiendo diferentes reportes que posibilitan la toma de decisiones; y genera solicitudes de transferencias y documentos de pago.

Esencialmente, los principales procesos bancarios en una empresa permiten:

- Gestionar las cuentas bancarias por tipo y monedas.

- Gestionar por cada cuenta bancaria los talonarios de los diferentes instrumentos de pagos (cheques, letras de cambio, pagaré, etc.)
- Emitir y registrar instrumentos de pago.
- Procesar los estados de cuenta de banco recibidos teniendo en cuenta las operaciones de cobros y pagos automáticos, así como las registradas con anterioridad por la entidad.
- Gestionar las conciliaciones bancarias por cada una de las cuentas bancarias permitiendo escoger el método a utilizar.
- Emitir reportes por moneda como son: registro de cheques emitidos, registro de cobros, registro de pagos, confirmación de saldos, registro de disponibilidad, etc.
- Realizar cierre diario, de período contable y de ejercicio para cada cuenta bancaria. (Mariaelena, 2010)

Fig. 1: Principales procesos de gestión bancaria.

1.3 Sistemas de Gestión Bancaria existentes

En el país existen varias soluciones nacionales y además se utilizan o han sido certificados módulos contables y financieros de ERP desarrollados por empresas extranjeras. Las organizaciones cubanas han realizado la implantación de varios sistemas de gestión, de manera integrada o en paralelo, pero siempre

utilizando las sinergias que se producen de la existencia de procesos comunes. Estos software, en su mayoría, están enfocados a un sector específico o han sido desarrollados en el marco de otra economía.

A continuación se realiza el análisis de algunos de los sistemas de gestión empresarial utilizados actualmente.

1.3.1 Software Financieros-Contables Cubanos

SISCONT-5

SISCONT-5 es un sistema desarrollado por la empresa de Tecnologías de la Información, Automática y las Comunicaciones (TECNOMATICA), que se aviene a las definiciones y conceptos del Ministerio de la Industria Básica aunque por las acciones contables financieras que permite puede ser utilizado en otras entidades nacionales.

SISCONT-5 permite trabajar en forma monousuario, multiusuario y por Internet, dependiendo el tipo de hardware que el cliente adquiera bajo el sistema operativo Windows XP o superior. Se actualiza totalmente gratis desde la página web de SISCONT y en cuanto a la seguridad posibilita configurar los permisos de cada usuario, de esta forma controla que el personal acceda únicamente a las funciones autorizadas. Es un software contable, multiempresa, doble moneda. Agiliza y facilita el ingreso de documentos, cuenta con asistentes de una sola pantalla y un motor de base de datos para recibir documentos en línea desde cualquier sistema.

SISCONT-5 contiene varios módulos integrados con diversas soluciones para su contabilidad y finanzas, entre ellos el módulo de tesorería que permite controlar y programar las cuentas por pagar, actualizar en línea el Flujo de Caja y emitir cheques y cuentas contables en forma directa. También ingresa un asiento contable actualizando automáticamente todos los libros de diario, mayor, libro banco, compras, ventas. (SISCONT, 2009)

SABIC

SABIC (Sistema automatizado para la Banca Internacional de Comercio) es un sistema diseñado y desarrollado por la Dirección de Sistemas Automatizados del Banco Central de Cuba.

Entre sus principales características están: la contabilización en tiempo real (que permite mantener actualizados los ficheros contables) y la contabilización multimoneda (que permite registrar los activos y

pasivos en las monedas orígenes sin tener que realizar en el momento del registro las conversiones de monedas). Además, las operaciones contables se pueden realizar a través de transacciones tipificadas generando los asientos contables de forma automática.

SABIC comprende las regulaciones para la aplicación de las normas y proceso electrónico de datos de operaciones tales como remesas, transferencias, cuentas bancarias, ingresos, pagos, etc. Entre otras cuestiones, el sistema apoya el aumento de la eficacia y eficiencia en los servicios bancarios, la seguridad de la información contable, la interoperabilidad entre las sucursales del banco, la actualización de sus bases de datos en tiempo real, etc. (Tamargo, 2009)

RODAS XXI

RODAS XXI es un Sistema Integral Económico Administrativo, creado por la empresa de Tecnologías de la Información y Servicios Telemáticos Avanzados (CITMATEL), que posibilita automatizar el funcionamiento de una empresa o unidad presupuestada.

Es un software multiempresa, trabaja con doble moneda y permite el intercambio automático de los comprobantes generados por cada módulo con el de Contabilidad.

Cuenta con seis módulos: Finanzas, Contabilidad, Activos Fijos, Nóminas, Inventario y Facturación. Estos módulos pueden emplearse integrados en su totalidad o forma independiente. Particularmente, el subsistema Finanzas permite tener el registro de cheques tanto emitidos, como recibidos, así como llevar el registro de otros instrumentos de pago y efectuar las operaciones de cobros y de pagos. Además, permite tener un control de dietas, reembolso, vales para pagos menores y el registro de ingresos; y posibilita llevar los submayores bancarios, realizar la conciliación bancaria y tener todo el control de caja con la posibilidad de realizar el arqueo correspondiente. (CITMATEL, 2010)

CONDOR

CONDOR es una solución tecnológica integral orientada a la gestión de grandes y medianas empresas y desarrollada por la empresa cubana de Servicios Informáticos, Consultorías y Sistemas (SICS) que se subordina al Ministerio del Transporte. Es un poderoso y flexible ERP que se adapta fácilmente a las necesidades de las organizaciones. Soporta funcionalidades como la multiempresa, multimoneda, y operación en tiempo real.

CONDOR ofrece soluciones óptimas adaptadas a las normas y principios de la Contabilidad General aceptados en Cuba, conforme con las normas y procedimientos establecidos por los Ministerios de Finanzas y Precios y de Informática y Comunicaciones.

Está compuesto por varios módulos: Contabilidad General, Nómina/Pre-nómina, Activos Fijos, Inventario, entre otros, permitiendo el intercambio entre ellos automáticamente y por opciones. Dentro del módulo de Tesorería se encuentran tareas como la administración de cartera de cheques, conciliación bancaria, impresión de cheques, pagos por ventanilla y por acreditación, y transferencias electrónicas de fondos. (Productos, 2008)

VERSAT-Sarasola

VERSAT-Sarasola fue desarrollado en 1998 por TEICO Villa Clara, empresa del Ministerio del Azúcar encargada de la Informática y las Comunicaciones, siendo el primer sistema de contabilidad cubano certificado y uno de los más completos existentes en el país.

Automatiza las actividades de planificación, control y análisis económico de cualquier tipo de entidad, abarcando la administración, contabilidad, medios de rotación, activos fijos, finanzas, cajas y costos. Está implementado con modernas tecnologías y trabaja en red con alta seguridad.

Está constituido por 12 módulos que incluyen configuración y seguridad, contabilidad general y de gastos, costos y procesos, análisis económico empresarial y control de activos fijos. Además, interviene finanzas y cajas, planificación y presupuestos, control de inventarios, de productos terminados, pago de salario, paquete de gestión, contratación y facturación.

VERSAT permite trabajar con diferentes monedas (multimoneda) para revalorizar los Estados Financieros en una fecha y con una tasa de cambio determinada. Además, posibilita procesar los documentos primarios en los diferentes subsistemas en doble moneda y por tanto obtener los resultados que se deseen en cada una de las monedas utilizadas.

El Sarasola fue seleccionado por el Ministerio de Finanzas y Precios (MFP) como la herramienta informática más adecuada para implantarse en varios Centros de Gestión Contables del país. Al evitarse la importación del sistema foráneo, se ahorró un millón 186 mil dólares que costaban las licencias, y la labor de consultoría, atenciones y otros gastos. (Lic. Miguel P Cabrera González, 2004)

1.3.2 Software Financieros-Contables Extranjeros

ASSETS-NS

ASSETS-NS es comercializado por la firma panameña D´MARCO S.A. y distribuido en Cuba en el año 1997 por INFOMASTER, entidad informática perteneciente a la Empresa Nacional de Producción y Servicios a la Educación Superior del MES. Es un sistema flexible, amigable, y funciona en ambiente multiusuario incluidas estaciones remotas.

Es un Sistema de Gestión Integral estándar y parametrizado capaz de adaptarse a las exigencias de cada entidad en particular; que permite el control de los procesos de Compras, Ventas, Producción, Taller, Inventario, Finanzas, Contabilidad, Presupuesto, Activos Fijos, Útiles y Herramientas y Recursos Humanos.

Permite el control por Fondos de efectivo; pueden existir distintos tipos de fondos de acuerdo con los requerimientos de cada Empresa. Se procesan los documentos recibidos en los procesos de Cobros y Cobros Directos y los Pagos Menores realizados desde Caja. Controla los pagos de Anticipos para gastos de viajes y otros conceptos, la liquidación y justificación de los mismos. Genera, automáticamente, los asientos de diario a la contabilidad por cada una de las transacciones contempladas en el sistema.

Permite realizar Arqueo de Caja, que consiste en el análisis de las transacciones que afectan el fondo en un período, con el objetivo de verificar si el saldo según transacciones, corresponde con lo que se encuentra físicamente en Caja en dinero efectivo, cheques o vales. Se puede además realizar el Reembolso de los Fondos. (ASSETS, 2004)

SAP ERP

SAP ERP es un producto desarrollado por la multinacional del software SAP (Sistemas, Aplicaciones y Productos), creada en 1972 en Alemania.

SAP ERP permite llevar la gestión de varias compañías, en distintas monedas, con base en más de un país. Brinda un sinnúmero de funcionalidades que permiten analizar el negocio de una empresa, optimizar finanzas, gestionar recursos humanos, operaciones y servicios corporativos.

El software ERP de SAP comprende cuatro soluciones independientes que brindan soporte a procesos de negocio clave a través de su sistema ERP específico: SAP ERP Financials, SAP ERP Human Capital Management, SAP ERP Operations y SAP ERP Corporate Services.

SAP ERP Financials permite mantener el control y la responsabilidad de las finanzas y proporciona las herramientas necesarias a la empresa para mejorar el retorno sobre la inversión y conseguir un crecimiento sostenible. Ofrece control e integración en la compañía de toda la información financiera y empresarial esencial para la toma de decisiones estratégicas y operativas. Mejora la gestión de los controles internos, simplifica el análisis financiero y permite llevar a cabo procesos de negocio adaptables.

La aplicación de SAP para bancos, ofrece un entorno sólido para manejar todos los aspectos de organización, brindando una variedad de beneficios empresariales. La gestión bancaria necesita controlar los costos, consolidar las operaciones y el negocio, y administrar las operaciones globales al mismo tiempo que se adaptan a las nuevas tecnologías. (CBM, 2010)

No obstante, el beneficio de SAP ERP es intangible, ya que se tiene que valorar cuanto le cuesta a la empresa tener o no la información. Además, su infraestructura puede ser muy alta si no se cumple desde un principio con los requerimientos del SAP. Se necesita una persona al 100% con el conocimiento de todo el negocio para que pueda hacerse cargo de administrar el SAP y dar solución a los problemas de forma rápida. Requiere de mantenimiento, actualización y fuertes inversiones en módulos complementarios. Es muy costoso, muchas empresas no pueden pagar los precios aunque necesitan de las prestaciones de un ERP. (Global, 2010)

Openbravo

Openbravo es un sistema de planificación de recursos empresariales (ERP) desarrollado por la compañía Tecnicia (actualmente conocida como Openbravo); que está preparado para implantarse en entornos multinacionales y multicliente.

Se encuentra dirigido a pequeñas y medianas empresas (pymes). Utiliza tecnologías modernas, pero sólidas y suficientemente probadas, para cumplir los requerimientos estrictos de rendimiento y escalabilidad de cualquier entorno empresarial. Constituye la solución profesional líder en el mercado de los ERP en código libre y entorno web, sin embargo, no dispone de clientes de utilización que no sean a

través de navegador Web. Debido a esta característica de software libre, el cliente tiene el control completo de la solución, sin depender de contratos o licencias.

Openbravo ofrece un sistema ERP totalmente integrado, adaptado a las necesidades de cada empresa, independientemente de su tamaño o su sector de actividad. Brinda una amplia gama de funcionalidades que abarcan los procesos de negocio, tales como: gestión de las relaciones con el cliente, facturación, aprovisionamiento, inventario, gestión de proyectos, gestión económica financiera e inteligencia de negocio. También ayuda a las empresas a administrar sus operaciones diarias, optimizar los procesos de negocios, lograr mayor satisfacción del cliente e incrementar la rentabilidad empresarial.

Dentro de la gestión económico-financiera Openbravo gestiona el plan de cuentas, las cuentas contables, los impuestos, las cuentas a pagar, las cuentas a cobrar y la contabilidad bancaria; incluye la realización de liquidaciones y emisión de informes de banco. (Openbravo, 2010)

Valoración crítica de los sistemas estudiados.

Partiendo del estudio y análisis de los sistemas contables existentes que incluyen de una forma u otra la gestión bancaria entre sus procesos, se arriba a la conclusión de que no existe un sistema altamente configurable que integre toda la información de la empresa y se adapte a las necesidades propias de cada economía, teniendo en cuenta las resoluciones financieras vigentes que cumplan con los datos de uso obligatorio de los documentos primarios, permitan registrar las transacciones en moneda original y su conversión a moneda base.

Lo cierto es, que el actual contexto se ha distinguido por el uso de sistemas con muchos beneficios como SAP ERP y Openbravo, ambos extranjeros, que comprenden los procesos bancarios entre sus módulos; pero estos sistemas requieren de mantenimiento y actualización que implica grandes inversiones para el país. Openbravo está implementado sobre plataforma Java, lo que exige altos requerimientos de hardware para su adecuado funcionamiento. Además de las limitaciones que constituye el elevado precio de un producto extranjero y el pago de sus licencias, está la dependencia que se crea a un suministrador externo, en quienes se seguiría invirtiendo hasta lograr adaptar el sistema a las necesidades de la economía cubana, aun cuando estos sean productos de reconocido prestigio.

Entre los sistemas cubanos analizados, la mayoría fueron diseñados inicialmente para un sector específico de la economía del país y no cubren en su totalidad las funcionalidades de la gestión bancaria.

Entre ellos el SISCONT, orientado al sector industrial; RODAS XXI al Centro de Investigación Tecnología y Medio Ambiente (CITMA) y CONDOR al Ministerio de Transporte respectivamente.

En resumen, la complejidad que caracteriza al sistema financiero ha hecho de la globalización una realidad que desborda las fronteras de muchos sistemas nacionales y extranjeros. De ahí la importancia de construir un módulo de Banco en el Sistema Integral de Gestión CEDRUX, con la marcada labor de proporcionar servicios y automatizar el control de todos los procesos bancarios para una entidad, cumpliendo con las legislaciones establecidas.

1.4 Modelo de desarrollo

Para el desarrollo del módulo se siguieron los lineamientos arquitectónicos establecidos por la dirección del proyecto, que plantean el uso de un modelo estandarizado, y la definición clara y precisa de las responsabilidades de cada uno de los roles que se ven involucrados en el desarrollo de la solución.

El modelo de desarrollo de software propuesto se estableció con el objetivo de producir software de alta calidad, basando en la retroalimentación continua entre el cliente y el equipo de desarrollo; siendo factible adaptar las características de las metodologías ágiles y robustas en la construcción de un nuevo modelo debido a la necesidad de producción de un sistema ERP para el país.

Este modelo de desarrollo se caracteriza por:

Centrado en la arquitectura

La arquitectura determina la línea base y los elementos de software estructurales a partir de la arquitectura de negocio. Interviene en la gestión de cambios y diseña la evolución e integración del producto. La arquitectura orienta las prioridades del desarrollo y resuelve las necesidades tecnológicas y de soporte para el desarrollo.

Orientado a componentes

Las iteraciones son orientadas por el nivel de significación arquitectónica de los componentes, que constituyen abstracciones de los procesos de negocio y requisitos asociados a modelar; y establecen al componente como unidad de medición y ordenamiento de las iteraciones.

Iterativo e incremental

Las iteraciones son planificadas y coordinadas con el equipo de arquitectura, los clientes y la alta gerencia. Cada iteración constituye el desarrollo de componentes, que son integrados permitiendo la evolución incremental del producto.

Ágil y adaptable al cambio

El desarrollo de las partes formaliza las características principales de la solución, priorizando los talleres y las comunicaciones entre las personas. Los clientes y funcionales están involucrados en el proyecto y poseen parte de la responsabilidad del éxito del mismo. Los cambios son conciliados semanalmente, discutidos y aprobados. (GESTIÓN, 2009)

1.5 Arquitectura de Software

La Arquitectura de Software es la organización fundamental de un sistema encarnada en sus componentes, las relaciones entre ellos y el ambiente y los principios que orientan su diseño y evolución. (Garlan, 1995)

El buen diseño de la arquitectura brinda una estructura que soporta soluciones a cada tipo de problema durante el desarrollo. Define la interacción y organización entre los distintos componentes del software, asegura que los requerimientos más importantes puedan ser evaluados e implementados. También permite flexibilidad en el sistema facilitando la ejecución de futuros cambios; y promueve la reutilización de componentes existentes como librerías de clases y aplicaciones de terceros.

La dirección de desarrollo del sistema CEDRUX establece elementos arquitectónicos que forman la arquitectura de software del proyecto, definiendo aplicar el estilo de Arquitectura basada en componentes siguiendo los principios de la Arquitectura Orientada a Servicios (SOA) y a través del patrón Modelo-Vista-Controlador.

1.5.1 Arquitectura basada en Componentes

Por las características que presenta el dominio del negocio y las tendencias de desarrollo de otros sistemas ERP, se decide adoptar para el desarrollo horizontal del sistema el estilo arquitectónico orientado a componentes.

Un componente es un fragmento reemplazable de un sistema de software, una unidad de composición con interfaces especificadas contractualmente, que satisface una o varias funcionalidades dentro del contexto de una arquitectura bien definida y puede ser ensamblado con otros fragmentos por medio de una interfaz.

Todas las funcionalidades modeladas en las fases de negocio y requerimientos quedan expresadas o contenidas en al menos un componente, y las distintas interacciones entre estos originan funcionalmente la existencia de módulos y subsistemas. (Fernández, 2010)

1.5.2 Patrón MVC

El patrón Modelo Vista Controlador MVC separa los datos de la aplicación, la interfaz de usuario, y la lógica de control en tres componentes distintos, permitiendo mayor independencia, mantenimiento y reutilización.

- ✓ **Modelo:** representa los datos del programa y controla todas sus transformaciones.
- ✓ **Vista:** genera la presentación visual de los datos representados por el Modelo y muestra los datos al usuario.
- ✓ **Controlador:** maneja las entradas del usuario, actuando sobre los datos representados por el Modelo.

1.5.3 Arquitectura basada en los principios de SOA

La arquitectura SOA define un conjunto de servicios tanto de negocio como tecnológicos que interactuando entre ellos, proporcionan la lógica necesaria para construir aplicaciones de una manera rápida y cumpliendo siempre con los principios de la Orientación a Servicios. Además SOA proporciona una serie de guías y recomendaciones para conseguir los objetivos que se impone una organización a la hora de desarrollar aplicaciones.

Este estilo está orientado al logro de la integración de aplicaciones independientes de forma que desde la red pueda accederse a sus funcionalidades, que serán tratadas como servicios. Se basa en contratos, donde el proveedor establece las reglas de comunicación, el transporte, y los datos de estrada y salida que serán intercambiados por ambas partes.

1.6 Patrones de diseño

Con el fin de diseñar aplicaciones de alta calidad se utilizan los patrones de diseño que promuevan la reutilización y agilicen el proceso de desarrollo de software.

Un patrón de diseño es una solución estándar para un problema común de programación, es una descripción de clases y objetos comunicándose entre sí, adaptada para resolver un problema de diseño general en un contexto particular. Los patrones de diseño proponen una forma de reutilizar la experiencia de los desarrolladores, para ello clasifican y describen formas de solucionar problemas que ocurren de forma frecuente en el desarrollo.

En principio, se aplican sólo en la fase de diseño, aunque se ha comenzado a definir y aplicar patrones en otras etapas del proceso de desarrollo, desde la concepción arquitectónica inicial hasta la implementación del código. A continuación se explican los patrones de diseños seleccionados y su aplicación en el desarrollo del módulo.

Patrones generales de software para asignar responsabilidades (GRASP)

Experto:

Se encarga de asignar la responsabilidad al experto en la información: la clase que cuenta con la información necesaria para cumplir la responsabilidad. Permite conservar el encapsulamiento, ya que los objetos se valen de su propia información para hacer lo que se les pide, lo que provee un bajo nivel de acoplamiento. Promueve clases sencillas y cohesivas que son más fáciles de mantener y comprender.

Creador:

Consiste en asignar a un Objeto la responsabilidad de crear otro Objeto. Un objeto es responsable de crear una nueva instancia de alguna clase si: agrega o contiene objetos de ella, registra las instancias de sus objetos o tiene los datos de inicialización que serán enviados a ella cuando el objeto sea creado.

Controlador:

El patrón ofrece una guía para tomar decisiones sobre los eventos de entrada, asignando la responsabilidad del manejo de mensajes de los eventos del sistema a una clase controladora, ya que los elementos de interfaz y sus controladores de eventos, no deben ser responsables de controlar los eventos del sistema.

Bajo acoplamiento:

El acoplamiento es una medida de la fuerza con que una clase está conectada a otras clases. Este patrón da soporte a una mínima dependencia y a un aumento de la reutilización; una clase con bajo acoplamiento no depende de "muchas otras" clases para realizar sus tareas, permitiendo que se pueda reutilizar con mayor facilidad y flexibilidad.

Alta cohesión:

La cohesión es una medida de cuán relacionadas y enfocadas están las responsabilidades de una clase. Una alta cohesión caracteriza a las clases con responsabilidades estrechamente relacionadas que no realizan un trabajo enorme. Fomenta la reutilización, mejorando la claridad y facilidad del diseño.

Patrones Gang of Four (GOF)

Fachada:

Es un patrón estructural que establece un objeto fachada proporcionando una interfaz única y simplificada para los servicios más generales del subsistema. Permite reducir la complejidad de diseño del sistema, minimizando la comunicación y dependencias.

1.7 Métricas de software para el diseño

Las métricas de software se definen como la aplicación continua de mediciones basadas en técnicas para el proceso de desarrollo del software, sus productos y el suministro de información relevante a tiempo. Facilitan una base para que el análisis, diseño, codificación y prueba puedan ser conducidos objetivamente y valorados cuantitativamente, brindando la información necesaria para la toma de decisiones técnicas.

Las métricas abarcan atributos de calidad relacionados con el desarrollo del software como:

- ✓ *Responsabilidad (Cohesión):* Responsabilidad asignada a una clase en un marco de modelado de un dominio o concepto de la problemática propuesta.
- ✓ *Complejidad del diseño:* Complejidad que posee una estructura de diseño de clases.
- ✓ *Complejidad de implementación:* Grado de dificultad que tiene la implementación en un diseño de clases determinado.

- ✓ *Complejidad del mantenimiento:* Grado de esfuerzo necesario a realizar para desarrollar una mejora, arreglo o rectificación de algún error de un diseño de software. Puede influir indirecta, pero fuertemente en los costes y la planificación del proyecto.
- ✓ *Reutilización:* Grado de reutilización presente en una clase o estructura de clase dentro de un diseño de software.
- ✓ *Acoplamiento:* Grado de dependencia o interconexión de una clase o estructura de clase con otras; está muy ligado a la característica de Reutilización.
- ✓ *Cantidad de pruebas:* Número o el grado de esfuerzo para realizar las pruebas de calidad (unidad) del producto (componente, módulo, clase, conjunto de clases, etc.) diseñado.
- ✓ *Nivel de Cohesión:* Grado de especialización de las clases concebidas para modelar un dominio o concepto específico.
- ✓ *Abstracción del diseño:* Capacidad de modelar lo más cercano posible a la realidad un concepto o dominio determinado.

1.8 Pruebas de software

Las pruebas de software se definen como la actividad en la cual un sistema o uno de sus componentes se ejecuta en circunstancias previamente especificadas (configuración de la prueba), registrándose los resultados. Se realiza un proceso de evaluación en el que los resultados obtenidos se comparan con los resultados esperados para localizar fallos en el software y se especifican mediante la realización de casos de pruebas como productos de desarrollo de software que ayudan a validar y verificar el sistema como un todo.

1.8.1 Métodos de pruebas

Existen fundamentalmente dos enfoques de prueba que permiten lograr mayor fiabilidad en el software y proporcionan distintos criterios para generar casos de prueba que provoquen fallos en el programa:

- ➢ Pruebas de Caja Negra (o Pruebas Funcionales)
- ➢ Pruebas de Caja Blanca (o Pruebas Estructurales)

Estas técnicas se pueden aplicar en cualquiera de los niveles de pruebas (unitarias, integración, aceptación, sistema) con diferentes grados de abstracción en la definición de los casos de prueba.

1.8.1.1 Pruebas de Caja Negra

Las pruebas de Caja Negra se centran principalmente en los requisitos funcionales del software. Se realizan sobre la interfaz, sin considerar la estructura interna del componente que se prueba, simulando una "Caja Negra" cuyo comportamiento sólo puede ser determinado por sus entradas y las salidas obtenidas a partir de ellas. Estas pruebas permiten detectar incorrectas o incompletas funcionalidades en el software, problemas de rendimiento, errores de accesos y de estructuras de datos externas.

Fig. 2: Pruebas de Caja Negra.

Para desarrollar las pruebas de caja negra existen varias técnicas:

- ✓ *Técnica de la Partición de Equivalencia*: Divide el campo de entrada en clases de datos que tienden a ejercitar determinadas funciones del software.
- ✓ *Técnica del Análisis de Valores Límites*: Prueba la habilidad del programa para manejar datos que se encuentran en los límites aceptables.
- ✓ *Técnica de Grafos de Causa-Efecto*: Permite al encargado de la prueba validar complejos conjuntos de acciones y condiciones.

1.8.1.2 Pruebas de Caja Blanca

Las pruebas de Caja Blanca se basan en un minucioso examen de los detalles procedimentales del código a evaluar. Requieren del conocimiento de la estructura interna del programa y son derivadas a partir de las especificaciones internas de diseño o el código. Además, permiten comprobar los caminos lógicos del software proponiendo casos de prueba que ejerciten conjuntos específicos de condiciones y/o

bucles. El objetivo de esta técnica es diseñar casos de prueba para que se ejecuten, al menos una vez, todas las sentencias del programa.

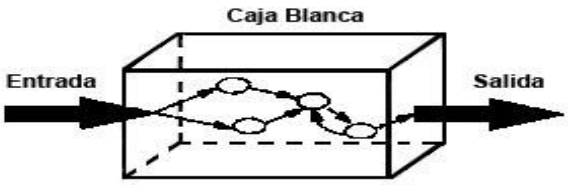

Fig. 3: Pruebas de Caja Blanca.

Entre los métodos de prueba de caja blanca se encuentran:

- ✓ *Prueba de Condición*: Ejercita las condiciones lógicas contenidas en el módulo de un programa.
- ✓ *Prueba de Flujo de Datos*: Selecciona caminos de prueba de un programa de acuerdo con la ubicación de las definiciones y los usos de las variables del programa.
- ✓ *Prueba de Bucles*: Se centra exclusivamente en la validez de las construcciones de bucles.
- ✓ *Prueba del Camino Básico*: Permite obtener una medida de la complejidad lógica de un diseño procedimental y usar esa medida como guía para la definición de un conjunto básico de caminos de ejecución.

1.9 Lenguajes de modelado

UML

UML (Unified Modeling Language) es el lenguaje de modelado unificado para la especificación, visualización, construcción y documentación de los artefactos de un sistema de software. Dispone un conjunto de notaciones y diagramas estándares para modelar sistemas orientados a objetos, describiendo la semántica esencial de lo que estos significan.

Además, se puede utilizar para modelar distintos tipos de sistemas: software, hardware, y organizaciones del mundo real. UML intenta solucionar el problema de propiedad de código que se da con los desarrolladores, al implementar un lenguaje de modelado común para todos mediante una documentación que cualquier desarrollador con conocimientos de UML será capaz de entender. Su utilización es independiente del lenguaje de programación y de las características de los proyectos, ya que UML ha sido

diseñado para modelar cualquier tipo de proyectos, tanto informáticos como de arquitectura, o de cualquier otra rama. (Systems, 2008)

1.10 Lenguajes de programación

1.10.1 En el servidor

Los lenguajes de programación del lado servidor son aquellos lenguajes que son reconocidos, ejecutados e interpretados por el propio servidor y que se envían al cliente en un formato comprensible para él. Estos se ejecutan en el servidor web, justo antes de que se envíen páginas al cliente. Estas páginas pueden realizar accesos a bases de datos, conexiones en red y otras tareas para crear la página final que verá el cliente. (Torre, 2006)

PHP 5.2

PHP, acrónimo de Hypertext Preprocessor, es el lenguaje de programación interpretado de alto nivel embebido en páginas HTML. Es gratuito e independiente de plataforma, rápido, con una gran librería de funciones y mucha documentación. Permite rápidamente a los desarrolladores la generación dinámica de páginas. Es uno de los lenguajes de programación más populares, debido a la gran fluidez y rapidez de sus scripts; siendo realmente fácil de utilizar por ventajas como su gratuidad y seguridad.

Como producto de código abierto, PHP goza de la ayuda de un gran grupo de programadores, permitiendo que los fallos de funcionamiento se reparen rápidamente. Una de sus características más potentes es su compatibilidad con las bases de datos más comunes, como MySQL, PostgreSQL, Oracle, Informix, ODBC, entre otras. Y ofrece la integración con varias bibliotecas externas.

PHP es la opción natural para los programadores en máquinas con Linux que ejecutan servidores web con Apache, pero funciona igualmente bien en cualquier otra plataforma de UNIX o de Windows, con el software de Netscape o del web server de Microsoft. (Henst, 2001)

1.10.2 En el cliente

Los lenguajes de programación del lado cliente son aquellos que pueden ser directamente interpretados por el navegador y no necesitan un pre-tratamiento, ya que el navegador es quien soporta la carga de procesamiento.

JavaScript

JavaScript es un lenguaje de programación que permite a los desarrolladores crear acciones en sus páginas web. Gran parte de su programación está centrada en describir objetos, escribir funciones que respondan a movimientos del mouse, aperturas, utilización de teclas, cargas de páginas, entre otros.

Permite la programación de pequeños scripts y de programas más grandes orientados a objetos, con funciones, estructuras de datos complejas, etc. Además, pone a disposición del programador todos los elementos que forman la página web, para poder acceder a ellos y modificarlos dinámicamente.

Gracias a su compatibilidad con la mayoría de los navegadores modernos, es el lenguaje de programación del lado del cliente más utilizado, es soportado por Internet Explorer, Netscape, Opera, Mozilla Firefox, entre otros.

HTML

HTML es el Lenguaje de Marcas de Hipertexto (Hypertext Markup Language) que permite describir hipertextos, textos presentados de forma estructurada y agradable, con enlaces que conducen a otros documentos o fuentes de información relacionadas, y con inserciones multimedia como gráficos y sonido.

Una de sus características esenciales es la universalidad, prácticamente cualquier ordenador, independientemente del sistema operativo, puede leer o interpretar una página web, convirtiéndose en uno de los formatos más populares y predominantes en la construcción de páginas web.

XML

XML conocido como lenguaje universal de marcado para documentos estructurados y datos en la Web, constituye un grupo de reglas y convenciones sintácticas que ofrece un formato para la descripción de datos estructurados y que se pueden utilizar para construir grupos de elementos de marcación propios.

Se desarrolló para proporcionar una flexibilidad y consistencia que no se podían alcanzar con HTML; no solo es un lenguaje de marcado, sino también un metalenguaje cuya particularidad más importante es que no posee etiquetas prefijadas con anterioridad, permitiendo describir otros lenguajes de marcado y definir lenguajes de presentación propios en dependencia del contenido del documento.

La meta fundamental del XML es hacer la cooperación y la interoperabilidad más fáciles entre módulos que pertenecen a diferentes aplicaciones, e incluso a diferentes organizaciones. Permite la definición,

transmisión, validación e interpretación de datos; garantizando que los datos estructurados sean uniformes e independientes de aplicaciones o fabricantes, y el intercambio de cualquier tipo de información, sin que ocasione problemas de tipo "contenido" o de tipo "presentación". (Función, 2006)

AJAX

AJAX, no es exactamente un lenguaje, su nombre viene dado por las siglas de Asynchronous JavaScript And XML (JavaScript asíncrono y XML), es un término que describe un nuevo acercamiento a usar un conjunto de tecnologías existentes juntas: HTML o XHTML, hojas de estilo (Cascading Style Sheets o css), JavaScript, DOM (Document Object Model), XML, y el objeto XMLHttpRequest. Es una técnica de desarrollo web para crear aplicaciones interactivas, válidas para múltiples plataformas y utilizables en muchos sistemas operativos y navegadores; permitiendo realizar cambios sobre las páginas sin necesidad de recargarlas, y aumentando la interactividad, velocidad y usabilidad en las aplicaciones. (Función, 2006)

1.11 Frameworks

El término framework se refiere a una estructura de software compuesta de componentes personalizables e intercambiables para el desarrollo de una aplicación. Entre los objetivos principales que se persigue con el uso de frameworks están: acelerar el proceso de desarrollo, reutilizar código ya existente y promover buenas prácticas de desarrollo como el uso de patrones.

Zend Framework 1.9.7

Zend Framework se trata de un framework de código abierto para desarrollo de aplicaciones y servicios Web con PHP5. Utiliza código 100% orientado a objetos y brinda soluciones para construir sitios web modernos, robustos y seguros.

La estructura de los componentes de Zend Framework es única; cada componente está construido con una baja dependencia de otros componentes. Esta arquitectura, débilmente acoplada, permite a los desarrolladores utilizar los componentes por separado.

Aunque se pueden utilizar de forma individual, los componentes de la biblioteca estándar de Zend Framework conforman un potente y extensible framework de aplicaciones web al combinarse. Además, ofrecen gran rendimiento y robusta implementación MVC, abstracción de base de datos fácil de usar, y un componente de formularios que implementa: la prestación de formularios HTML, validación y filtrado para

que los desarrolladores puedan consolidar todas las operaciones usando de una manera sencilla la interfaz orientada a objetos. (Framework, 2010)

Zend_Ext Framework 1.5.4

Zend_Ext framework es una extensión de Zend Framework diseñada para PHP que utiliza el patrón MVC como base de su funcionamiento. Es fácilmente integrable a las aplicaciones, debido a que contiene diferentes clases de gran utilidad como la búsqueda dinámica de ficheros a incluir o utilizar.

Zend_Ext cuenta con un importante mecanismo de manejo de controladores y vistas, por lo que se propone tenerlo en cuenta para el diseño de estos dos componentes de la arquitectura. Posee un controlador vertical para el control de las acciones realizadas por las vistas hacia el controlador y un motor de reglas para las validaciones en el servidor. Tiene incorporado el ORM Doctrine Framework para trabajo en la capa de abstracción a base de datos y el ExtJs Framework para el desarrollo de las vistas.

El framework garantiza la comunicación entre diferentes módulos y componentes mediante un mecanismo que permite a otros módulos o componentes realizar acciones de control que se requieran para el conjunto de sucesos que tengan que ocurrir. Este mecanismo se basa en el patrón Inversión de Control. (Ver epígrafe 2.6)

Doctrine 1.2.1

Doctrine es un potente y completo sistema para el desarrollo de aplicaciones PHP 5.2 o superior que utilicen bases de datos. Implementa el patrón ORM (Object Relational Mapping) para desarrollar el dominio de una aplicación; cuenta con una capa de abstracción para el acceso a bases de datos y un lenguaje de consulta propio que abstrae del gestor que se esté utilizando. Sus funcionalidades permiten exportar una base de datos existente a sus clases correspondientes y también convierten clases (convenientemente creadas siguiendo las pautas del ORM) a tablas de una base de datos. Uno de sus rasgos importantes es la habilidad de escribir opcionalmente las preguntas de la base de datos orientado a objeto; lo que proporciona una alternativa poderosa a los diseñadores de SQL manteniendo un máximo de flexibilidad sin requerir la duplicación del código innecesario. (Gómez, 2009)

ExtJs 2.2

ExtJs es una librería JavaScript que permite construir aplicaciones complejas. Incluye componentes UI del alto performance y personalizables, modelo de componentes extensibles, un API fácil de usar y licencias open source y comerciales.

Su sistema de licenciamiento no contempla la licencia LGPL; o se tiene código 100% GPL o se debe pagar por su licencia de desarrollo.

Es soportado por varios navegadores web como Internet Explorer, Firefox, Safari y Opera. (Corzo, 2008)

1.12 Herramientas y tecnologías de desarrollo

1.12.1 Herramientas CASE

CASE (Computer Aided Software Engineering), siglas de Ingeniería de Software Asistida por Computación, define la aplicación de métodos y técnicas para comprender las capacidades de las computadoras, por medio de programas, de procedimientos y su respectiva documentación.

Las herramientas CASE tienen gran importancia ya que permiten al administrador de un proyecto informático, llevarlo adelante de forma eficaz y eficiente. Para el desarrollo de proyectos se considera imprescindible la utilización de estas herramientas porque posibilitan organizar y manejar la información del mismo.

Visual Paradigm 6.4

Visual Paradigm es una herramienta CASE que soporta el ciclo de vida completo del desarrollo de software: análisis y diseño orientados a objetos, construcción, pruebas y despliegue.

Su mayor éxito consiste en la capacidad de ejecutarse sobre diferentes sistemas operativos que le confiere la característica de ser multiplataforma. Utiliza UML como lenguaje de modelado ofreciendo soluciones de software que permiten a las organizaciones desarrollar las aplicaciones de calidad de forma rápida y barata. Presenta un ambiente gráfico agradable para el usuario. Permite configurar las líneas de redacción, el modelado de base de datos, el modelado de requerimientos, el modelado del proceso de negocio, la interoperabilidad, la generación de documentación y la generación de código base para diferentes lenguajes de programación como Java, C# y PHP.

Visual Paradigm soporta UML 2.1 completo y BPMN, permitiendo realizar ingeniería tanto directa como inversa; es colaborativa, porque soporta múltiples usuarios trabajando sobre el mismo proyecto; genera la documentación del proyecto automáticamente en varios formatos como Web o pdf, y permite el control de versiones. (Manager, 2007)

1.12.2 Herramientas de desarrollo colaborativo

Control de versiones

El control de versiones es la gestión de los diversos cambios que se realizan sobre los elementos de algún producto o una configuración del mismo. Los sistemas de control de versiones facilitan la administración de las distintas versiones de cada producto desarrollado, así como las posibles especializaciones realizadas. El principal objetivo es permitir editar de forma colaborativa y compartir información. Aunque un sistema de control de versiones puede realizarse de forma manual, es muy aconsejable disponer de herramientas que faciliten esta gestión.

Subversion TortoiseSVN 1.4.5

Subversion (SVN) es un software libre desarrollado para el control de versiones. Es un sistema centralizado para compartir información que permite realizar modificaciones atómicas y gestionar archivos, directorios y sus cambios a través del tiempo, lo que facilita las tareas administrativas. Su capacidad para que varias personas puedan modificar y administrar el mismo conjunto de datos desde sus respectivas ubicaciones fomenta la colaboración.

TortoiseSVN es el cliente gratuito para el sistema de control de versiones Subversion, con código abierto y software libre bajo la licencia GNU GPL. Está disponible en 28 idiomas diferentes y puede ser usado sin un entorno de desarrollo. (Briano, 2008)

1.12.3 IDE

IDE: Integrated Development Environment (Entorno de Desarrollo Integrado), constituye un editor de código para depurar y facilitar las diferentes tareas necesarias en el desarrollo de cualquier tipo de aplicación que pueda funcionar con diferentes lenguajes de programación.

A continuación se listan los IDEs usados para PHP con el objetivo de seleccionar el entorno de desarrollo que mejor se adapte a las necesidades del proyecto.

Zend Studio

Zend Studio es un editor de texto para páginas PHP que proporciona buen número de ayudas desde la creación y gestión de proyectos hasta la depuración del código. Es posiblemente uno de los mejores IDE del momento y uno de los mayores impulsores de PHP orientado a desarrollar aplicaciones web.

Actualmente, está disponible el nuevo Zend Studio Neon que a diferencia de las versiones anteriores ya no se trata de un IDE desarrollado en Java (excesiva lentitud y consumo de memoria en algunos casos), sino basado en Eclipse. (TuFunción, 2007)

Spket

Spket es un plugin para Eclipse y Aptana que provee un conjunto de utilidades para la edición de JavaScript, sobre todo para la edición de clases que extienden del framework JavaScript ExtJs o que usan la librería. Proporciona un editor de código JavaScript muy parecido al editor Java de Eclipse, incluyendo el autocompletado de código, resaltado de texto, muestra de errores, etc.

Spket IDE es una excelente aplicación que ofrece la posibilidad de editar en lenguaje de programación JavaScript para la creación de utilidades menores. Dentro de sus características, se destacan el autocompletado de comandos, diferenciación por colores de la sintaxis, etc. Cuenta con un funcionamiento totalmente sencillo para todo aquel programador profesional o aficionado y posee una interfaz gráfica verdaderamente eficiente y completa para la edición de aplicaciones. (MP3es, 2009)

1.12.4 Servidores de aplicaciones

Un servidor de aplicaciones es un software que ayuda al servidor Web a procesar las páginas que contienen scripts o etiquetas del lado del servidor.

Apache 2.0

Apache es un servidor Web de tecnología open source sólido, y el más usado por los servidores en todo Internet. Su robustez y estabilidad hacen que cada vez millones de servidores reiteren su confianza en este programa. Al ser una tecnología gratuita de código fuente abierto permite realizar modificaciones en el código fuente.

Es un servidor que corre en una multitud de Sistemas Operativos, que lo hace prácticamente universal. Tiene una alta configuración en la creación y gestión de logs y permite personalizar la respuesta ante los posibles errores que se puedan dar en el servidor.

Apache es un servidor altamente configurable de diseño modular que trabaja con gran cantidad de lenguajes de script como Perl, PHP y otros, teniendo todo el soporte que se necesita para tener páginas dinámicas. (CiberAula, 2006)

1.12.5 Servidores de BD

Los servidores de bases de datos surgen por la necesidad que tienen las empresas de manejar grandes y complejos volúmenes de datos, al tiempo que requieren compartir la información con un conjunto de clientes de una manera segura. Ante este enfoque, un sistema gestor de bases de datos (SGBD) deberá ofrecer soluciones de forma fiable, rentable y de alto rendimiento.

Los SGBD proporcionan herramientas de apoyo a la toma de decisiones ("datawarehouse") proporcionando una plataforma de transacciones "on-line" (OLTP) que hacen que la información esté siempre actualizada y consistente.

Ofrecen además, las herramientas de administración completas que simplifican la tarea de la configuración, seguridad, creación y gestión de bases de datos; y facilitan los mecanismos de integración con otros sistemas, políticas de copias de seguridad y herramientas que permitan su programación tanto a nivel de diseño como a nivel de reglas o procedimientos que encapsulen la arquitectura de la base de datos.

PostgreSQL 8.3

PostgreSQL es un Sistema de Gestión de Bases de Datos Objeto-Relacionales. Está ampliamente considerado como el sistema de bases de datos de código abierto más avanzado del mundo y se destaca por su robustez, escalabilidad y cumplimiento de los estándares SQL.

Entre las características de PostgreSQL están: alta concurrencia, que evita tener que bloquear una tabla cuando se está escribiendo en ella; las copias de seguridad en línea, la replicación asíncrona, las transacciones anidadas y el optimizador de consultas. (UptoDown, 2009)

Se ejecuta en la mayoría de los Sistemas Operativos más utilizados en el mundo incluyendo, Linux, varias versiones de UNIX y por supuesto Windows. Y constituye una solución real a los complejos problemas del mundo empresarial manteniendo la eficiencia al consultar los datos.

El hecho de ser un producto open source, sin costos de licencia, convierte a PostgreSQL en una alternativa atractiva para las empresas que buscan ahorro significativo de sus costos, porque se tiene la libertad de usarlo, modificarlo y distribuirlo en productos comerciales o no comerciales.

1.12.6 Navegadores web

Los navegadores son aplicaciones capaces de interpretar las órdenes recibidas en forma de código HTML fundamentalmente y convertirlas en las páginas que son el resultado de dicha orden. Permite al usuario recuperar y visualizar documentos de hipertexto, comúnmente descritos en HTML, desde servidores web.

Mozilla Firefox 3.0

Mozilla Firefox es un navegador web libre desarrollado por la Corporación Mozilla. Considerado uno de los navegadores más usado en la actualidad, es multiplataforma y disponible en varias versiones de Microsoft Windows, Mac OS X y GNU/Linux. Su código fuente es software libre, publicado bajo una triple licencia GPL/LGPL/MPL. Mozilla Firefox es compatible con varios estándares web, incluyendo HTML, XML, XHTML, CSS, JavaScript, DOM e imágenes. (López, 2004)

1.13 Conclusiones

En este capítulo se realizó el estudio de los procesos financieros y los sistemas informáticos vinculados a la gestión bancaria en las entidades. Se llevó a cabo el análisis de las herramientas y tecnologías definidas por la línea de arquitectura como estándares y políticas precisas para la producción de software.

Finalmente, se confirmó la importancia y necesidad de desarrollar un módulo que informatice los procesos bancarios en las entidades del país y solucione de forma eficiente los problemas referentes a la gestión empresarial.

CAPÍTULO II: Diseño e Implementación del módulo Banco.

2.1 Introducción

Para asegurar el control de los procesos financieros bancarios realizados en las diferentes entidades nacionales, es obligatorio el estricto cumplimiento de las legislaciones pertinentes brindando mayor seguridad, rapidez y garantía en dichos procesos.

El presente capítulo tiene como objetivo describir la solución informática del módulo Banco del subsistema Finanzas. Se obtiene el diseño de la solución arquitectónica a partir de requisitos identificados por los analistas, patrones de diseño y definiciones arquitectónicas establecidas en la línea base de la arquitectura del proyecto. Se definen los componentes que agrupan las funcionalidades necesarias; además se estudian los estándares de codificación y se realiza la implementación y descripción de algoritmos no triviales.

2.2 Diseño de la Solución Arquitectónica

El diseño es el primer paso en la fase de desarrollo de cualquier producto o sistema de ingeniería. Su objetivo es producir un modelo o representación de una entidad que se va a construir posteriormente. (Pressman, 2005)

2.2.1 Valoración del análisis

Partiendo de los artefactos obtenidos durante el análisis de los procesos bancarios para el módulo Banco se realiza la valoración del mismo. Se validan los requisitos para demostrar que sus definiciones son las que el usuario final necesita. Los requerimientos identificados están claros y se entienden correctamente, lo que permite alcanzar retroalimentación en cuanto a si el sistema diseñado basándose en los requerimientos permite al usuario realizar su trabajo de manera eficiente y efectiva.

Para asegurar que los requisitos han sido establecidos sin ambigüedades o inconsistencias y que los errores encontrados durante la definición de los mismos hayan sido corregidos, se realizó una revisión a las especificaciones realizadas para ser aprobadas y no se detectaron deficiencias, omisiones, ni errores en la especificación. Se realizaron revisiones de artefactos generados en el proceso de análisis de software, actividades de documentación, revisión, y control de cambios.

Se comprueba que las necesidades de los clientes se encuentren cubiertas por los requerimientos capturados y especificados, creando una puerta de entrada apropiada y un punto de partida para las actividades de diseño e implementación.

2.2.2 Principales funcionalidades

Los requisitos funcionales identificados y aprobados se agruparon en un conjunto de funcionalidades con una finalidad u objetivo específico:

1. Gestionar tipo de cuenta bancaria:
   - ✓ Adicionar tipo de cuenta.
   - ✓ Modificar tipo de cuenta.
   - ✓ Consultar tipo de cuenta.
   - ✓ Listar tipos de cuentas bancarias.
   - ✓ Eliminar tipo de cuenta.
2. Gestionar cuenta bancaria:
   - ✓ Adicionar cuenta bancaria.
   - ✓ Modificar cuenta bancaria.
   - ✓ Eliminar cuenta bancaria.
   - ✓ Consultar cuenta bancaria.
   - ✓ Mostrar operaciones realizadas con una cuenta bancaria.
   - ✓ Mostrar estados de cuenta de una cuenta bancaria.
   - ✓ Mostrar unidades usuarias de una cuenta bancaria.
   - ✓ Desasociar sobregiro bancario.
   - ✓ Asociar sobregiro bancario.
   - ✓ Listar cuentas bancarias.
   - ✓ Mostrar disponibilidad de efectivo de una cuenta bancaria.

3. Gestionar talonario de instrumentos bancarios:
   - ✓ Adicionar talonario de instrumento bancario.
   - ✓ Modificar talonario de instrumento bancario.
   - ✓ Controlar uso de talonario de instrumento bancario.
   - ✓ Consultar talonario de instrumento bancario.
   - ✓ Listar talonarios de instrumentos bancarios.
   - ✓ Eliminar talonario de instrumento bancario.
4. Gestionar instrumento bancario:
   - ✓ Adicionar instrumento bancario.
   - ✓ Modificar instrumento bancario.
   - ✓ Eliminar instrumento bancario.
   - ✓ Cancelar instrumento bancario.
   - ✓ Asociar operaciones bancarias a instrumento bancario.
   - ✓ Imprimir instrumento bancario.
   - ✓ Confirmar instrumento bancario.
   - ✓ Listar instrumento bancario.
   - ✓ Contabilizar instrumento bancario.
5. Liquidar obligaciones:
   - ✓ Liquidar obligaciones.
   - ✓ Modificar liquidación de obligaciones.
   - ✓ Eliminar liquidación de obligaciones.
   - ✓ Cambiar de cuentas por pagar a efectos por pagar.
6. Liquidar obligaciones fiscales:

- ✓ Liquidar obligaciones fiscales.
- ✓ Modificar obligaciones fiscales.
- ✓ Eliminar liquidación de obligaciones fiscales.

7. Realizar reembolso:
   - ✓ Realizar reembolso.
   - ✓ Modificar reembolso.
   - ✓ Eliminar reembolso.
8. Realizar pago anticipado:
   - ✓ Realizar pago anticipado.
   - ✓ Modificar pago anticipado.
   - ✓ Confirmar pago anticipado.
   - ✓ Eliminar pago anticipado.
9. Realizar pago automático:
   - ✓ Realizar pago automático.
   - ✓ Modificar pago automático.
   - ✓ Eliminar pago automático.
10. Realizar cobro automático:
    - ✓ Realizar cobro automático.
    - ✓ Modificar cobro automático.
    - ✓ Eliminar cobro automático.
11. Realizar otros pagos:
    - ✓ Realizar otros pagos.
    - ✓ Modificar otros pagos.

- ✓ Eliminar otros pagos.

12. Procesar estado de cuenta:
    - ✓ Procesar estado de cuenta.
    - ✓ Validar estado de cuenta.
    - ✓ Consultar estado de cuenta.
    - ✓ Consultar operaciones cargadas a estado de cuenta.
    - ✓ Consultar desglose de correcciones de un estado de cuenta.
    - ✓ Contabilizar estado de cuenta.
    - ✓ Listar estados de cuenta.
    - ✓ Eliminar estado de cuenta.
    - ✓ Imprimir estado de cuenta.
13. Gestionar diferencias-correcciones con banco:
    - ✓ Adicionar diferencia con banco.
    - ✓ Modificar-corregir diferencia con banco.
    - ✓ Eliminar diferencia con banco.
    - ✓ Consultar diferencias con banco.
    - ✓ Listar diferencias con banco.
14. Gestionar operaciones asociadas al estado de cuenta:
    - ✓ Asociar operaciones bancarias a estado de cuenta.
    - ✓ Desasociar operaciones bancarias a estado de cuenta.
    - ✓ Listar operaciones asociadas en el estado de cuenta.
15. Gestionar conciliación bancaria:
    - ✓ Calcular conciliación bancaria.

- ✓ Consultar conciliación bancaria.
- ✓ Buscar conciliación bancaria.
- ✓ Listar conciliaciones bancarias.
- ✓ Imprimir conciliación bancaria.

16. Realizar cierre:
    - ✓ Realizar cierre diario.
    - ✓ Imprimir cierre diario.
    - ✓ Realizar cierre de periodo.
    - ✓ Imprimir cierre de periodo.
    - ✓ Realizar cierre de ejercicio.
    - ✓ Imprimir cierre de ejercicio.

2.3 Patrones de diseño empleados

Experto: se puso en práctica en todos los componentes, con el uso de clases que poseen responsabilidades específicas a cumplir de acuerdo con la información que manejan, los componentes cuentan con clases controladoras, modelos y entidades que poseen funciones concretas de acuerdo con la información que gestionan. Además, se modeló una clase entidad por cada tabla de la base de datos, posibilitando el trabajo específico y directo con el experto en la información.

Creador: Es útil contar con el principio general para la asignación de responsabilidades de creación porque permite que el diseño pueda soportar bajo acoplamiento, mayor claridad, encapsulación y reutilización. En el módulo se evidencia este patrón en cada componente, las clases controladoras son responsables de crear el objeto de las modelos, y estas a su vez de las entidades.

Controlador: Se utilizan cuando la aplicación es muy extensa, de esta forma, en el módulo en vez de tener un solo controlador y saturarlo, se tienen clases Controllers, que son controladores más pequeños especializados en las funcionalidades de cada componente.

Bajo acoplamiento: Los componentes en el módulo fueron diseñados bajo este principio, porque solo establecen las relaciones necesarias entre ellos. La base de datos fue definida de modo que entre las tablas existiera dependencia mínima, haciéndolas más independientes y reutilizables para reducir el impacto de los cambios y acrecentar la oportunidad de una mayor productividad.

Alta cohesión: En el módulo existe afinidad entre cada clase y los métodos que implementan, estas poseen responsabilidades vinculadas acordes a la información que controlan; y colaboran con otros objetos para compartir el esfuerzo si la tarea es grande, facilitando su mantenimiento y reutilización.

Fachada: Su aplicación posibilita promover un débil acoplamiento entre los diferentes componentes que conforman el módulo y su integración con otros subsistemas. Se usa una única clase Service por componente que proporciona los servicios necesarios publicados en el IOC donde se implementan todas las funcionalidades que el componente es capaz de brindar.

2.3.1 Estructura del módulo de Banco en Componentes

A partir de los requisitos definidos y las principales funcionalidades identificadas, el módulo se estructuró en 6 componentes siguiendo los lineamientos del grupo de arquitectura central del proyecto ERP Cuba.

Básicamente, los principales procesos del módulo Banco se centran en los siguientes componentes:

Configuración: Gestiona los tipos de cuenta bancaria, las cuentas bancarias y los talonarios de instrumentos bancarios**.**

Instrumento: Gestiona los instrumentos bancarios, liquida obligaciones, realiza reembolsos, cobros automáticos, pagos anticipados y automáticos.

Estado de cuenta: Procesa los estados de cuenta, gestiona las conciliaciones bancarias, las diferencias-correcciones con banco y las operaciones asociadas a los estados de cuenta.

Cierre: Contiene la funcionalidad realizar cierre de período contable y de ejercicio fiscal de módulo.

Carga Inicial: Permite introducir al sistema los estados de cuenta, las diferencias de los estados de cuenta y los instrumentos y realizar conciliación bancaria.

Recuperaciones: Gestiona los reportes del módulo.

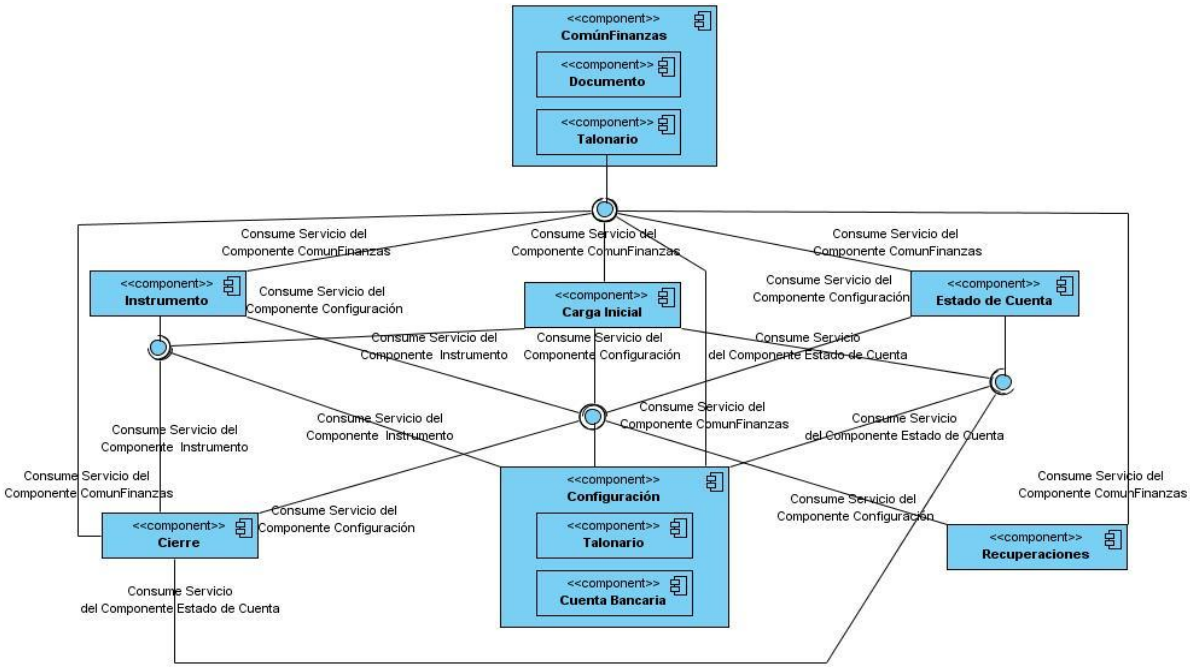

Fig. 4: Diagrama de componentes.

El componente *Configuración* incluye 2 subcomponentes que definen a través de interfaces un conjunto de funcionalidades o servicios de negocio para que puedan ser accedidos por los demás componentes que lo necesitan para su beneficio. El subcomponente *CuentaBancaria* gestiona las cuentas bancarias, configurando el tipo de cuenta que atiende la entidad. Brinda 16 servicios a través de las interfaces *CuentaService* y *TipoCuentaService*. Estos servicios son consumidos por el resto de los componentes del subsistema, exceptuando el subcomponente *Talonario*. Consume servicios de los componentes *Instrumento* y *EstadoCuenta* del propio módulo, y del componente de integración C*omúnFinanzas*. Se integra además, con los subsistemas Caja, Configuración, Estructura y Composición y Contabilidad. El subcomponente *Talonario* se encarga de la gestión de los talonarios asignados a cada cuenta bancaria. No brinda servicios, sólo consume del componente de integración C*omúnFinanzas*.

El componente *CargaInicial* ingresa al sistema la información acumulada antes del momento de iniciar el procesamiento en el sistema. No brinda servicios. Consume de los componentes *CuentaBancaria*, *Instrumento* y *EstadoCuenta* del propio módulo y del componente de integración C*omúnFinanzas*. Se integra con los subsistemas Caja, Configuración y Contabilidad.

El componente *Instrumento* gestiona los instrumentos correspondientes a una operación realizada, asociados a cada talonario perteneciente a una cuenta. Ofrece 16 servicios a través de su interfaz *InstrumentoBancoService*, que son consumidos por los componentes del propio módulo, *CuentaBancaria*, *Cierre* y *CargaInicial*. Consume servicios del componente de integración *ComúnFinanzas* y se integra con otros subsistemas como Caja, Cobros y Pagos, Configuración, Estructura y Composición, Contabilidad y Capital Humano.

El componente *EstadoCuenta* gestiona los estados de las cuentas que llegan a la entidad desde el banco y realiza el proceso de conciliación bancaria. Brinda 31 servicios a través de sus interfaces *EstadoCuentaService* y *ConciliacionService* que son consumidos por los componentes *CuentaBancaria*, *Instrumento*, *Cierre*, *Recuperaciones* y *CargaInicial*. Consume servicios del subcomponente *CuentaBancaria* del propio módulo y del componente de integración C*omúnFinanzas*. Se integra con los subsistemas Configuración, Estructura y Composición y Contabilidad.

El componente *Cierre* realiza el cierre diario, de período contable y de ejercicio fiscal. No ofrece servicios, pero consume de los componentes *CuentaBancaria*, *Instrumento*, *EstadoCuenta* y del componente de integración C*omúnFinanzas*. Se integra con el subsistema Caja.

El componente *Recuperaciones* gestiona los reportes del módulo. No ofrece servicios, pero consume de los componentes *CuentaBancaria*, *EstadoCuenta* y del componente de integración C*omúnFinanzas. S*e integra con el subsistema Caja.

2.3.2 Estructura del componente ComúnFinanzas

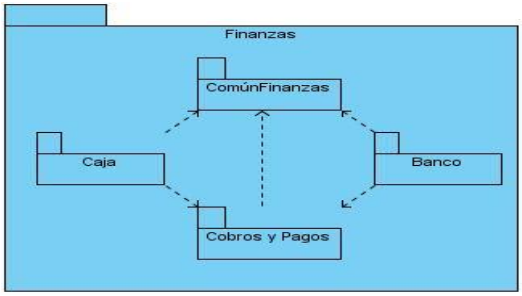

Fig. 5: Subsistema Finanzas.

En el subsistema Finanzas cada módulo requiere un amplio empleo de los talonarios y documentos para realizar las operaciones contables o respaldar de la ejecución de las mismas. Por ello, se incluye un

componente genérico denominado C*omúnFinanzas* que integra las necesidades comunes de los módulos Banco, Caja y Cobros y Pagos. Este componente se diseñó para recibir las peticiones de dichos módulos indistintamente y emitir la respuesta correcta en todo momento, permitiendo ahorrar en tiempo y esfuerzo en el desarrollo de la solución para cada uno de los módulos de forma similar y evitando la redundancia de información.

El componente C*omúnFinanzas* está compuesto por 2 subcomponentes:

- Documento: Es el encargado de gestionar todos los documentos del subsistema de Finanzas.
- Talonario: Gestiona los talonarios del subsistema de Finanzas, ya sean de documentos o instrumentos.

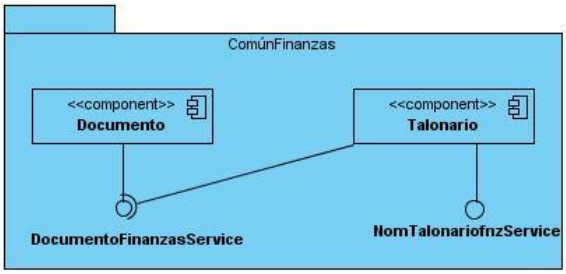

Fig. 6: Estructura interna del componente ComúnFinanzas.

El *subcomponente* Documento a través de la interfaz *DocumentoFinanzasService* brinda un conjunto de funcionalidades accedida por el subcomponente *Talonario*. De forma similar gran parte de los componentes de los módulos Caja, Banco y Cobros y Pagos también accederán a ella para realizar sus funciones de negocio. El subcomponente *Talonario* cuenta con la interfaz *NomTalonarioService* que ofrece los servicios necesarios para el funcionamiento de otros componentes del subsistema Finanzas.

2.3.3 Modelo de Datos en términos de componentes

Para el mejor diseño del sistema se decide agrupar las tablas del modelo entidad por componentes. Las clases entidades se distribuyen de forma que se puedan aprovechar las ventajas del diseño por componentes y la reutilización de los mismos, evitando que las clases estén desagregadas; y así mejorar el rendimiento y la compresión por el equipo de desarrollo del módulo. (Ver Anexo 1)

2.3.4 Diseño de clases por componente

Los diagramas de clases del diseño describen gráficamente las especificaciones de las clases de software y expresan la definición de clases como componentes del software.

Diagrama de clases del diseño del componente Configuración:

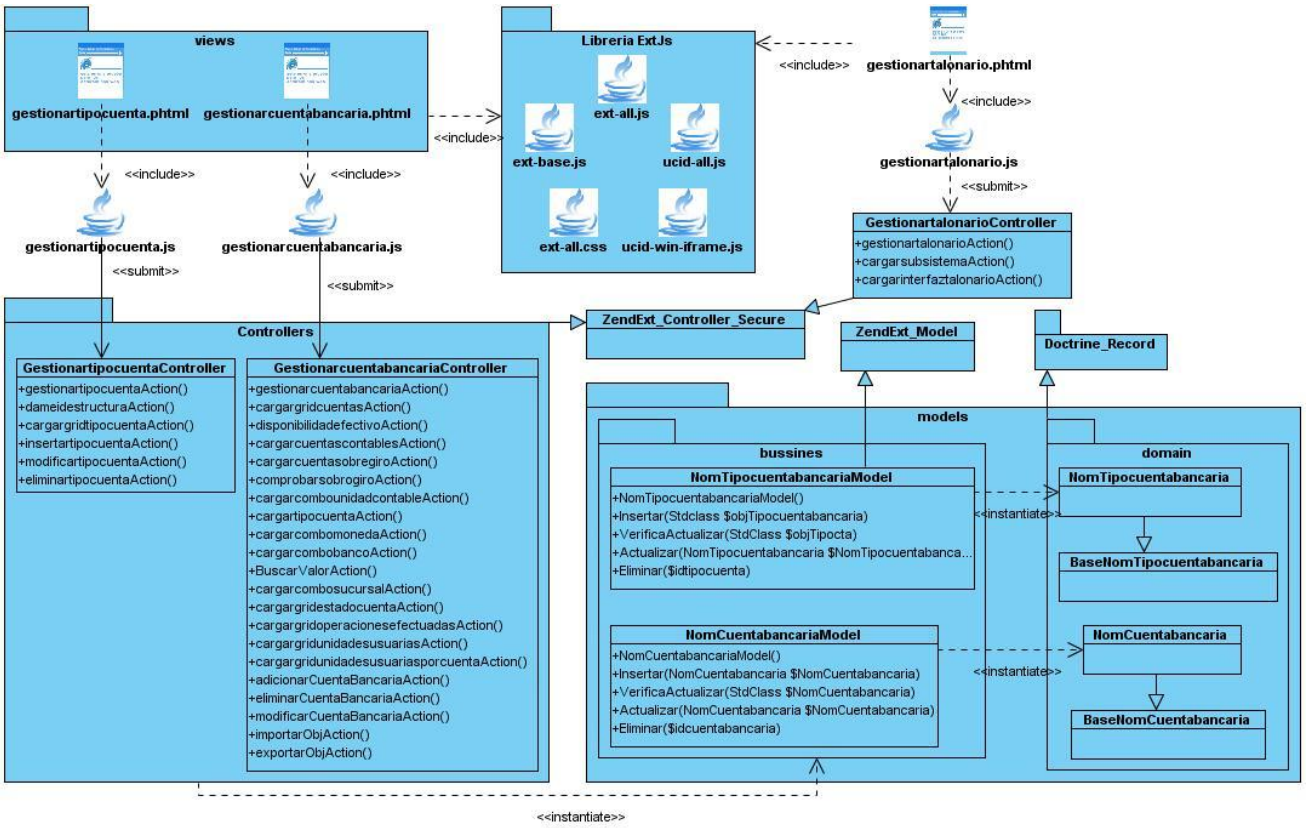

Fig. 7: Diagrama de clases del diseño del componente Configuración.

El diseño realizado se representa con estereotipos web debido a que el software es una aplicación web. Se representan las clases *gestionartipocuenta.phtml*, *gestionarcuentabancaria.phtml* y *gestionarcuentabancaria.phtml* que contienen componentes generados en la librería JavaScript ExtJs y son responsables de visualizar a través de los js *gestionartipocuenta.js*, *gestionarcuentabancaria.js* y *gestionartalonario.js* respectivamente, la información necesaria para gestionar los tipos de cuentas bancarias, las cuentas y los talonarios. Estas clases a la vez incluyen las controladoras con igual nombre

que contienen la implementación de las funcionalidades, y heredan de *ZendExt_Controller_Secure* para gestionar la seguridad.

Se representan también las clases que forman parte de la capa lógica del negocio y de la cual se nutren las clases controladoras. El paquete *Model* maneja los datos persistentes dentro del componente y contiene el *Bussines* y el *Domain*. Hereda de *ZendExt_Model* para gestionar la seguridad de los datos persistentes ubicados en el Model.

Los diagramas de clases del diseño para el resto de los componentes se muestran en el Anexo 2.

2.4 Implementación

2.4.1 Integración entre componente

La arquitectura en 3 capas: presentación (view), negocio (controller) y acceso a datos (models), consiste en el flujo de los datos desde la vista hacia la capa de datos y viceversa, a través de los diferentes elementos que la componen. Consta de 4 nodos de integración: vista-controlador, controlador-modelo, modelo-framework Doctrine y Doctrine-base de datos. La comunicación entre las capas dentro de un mismo componente se realiza mediante llamadas a métodos o eventos de forma directa.

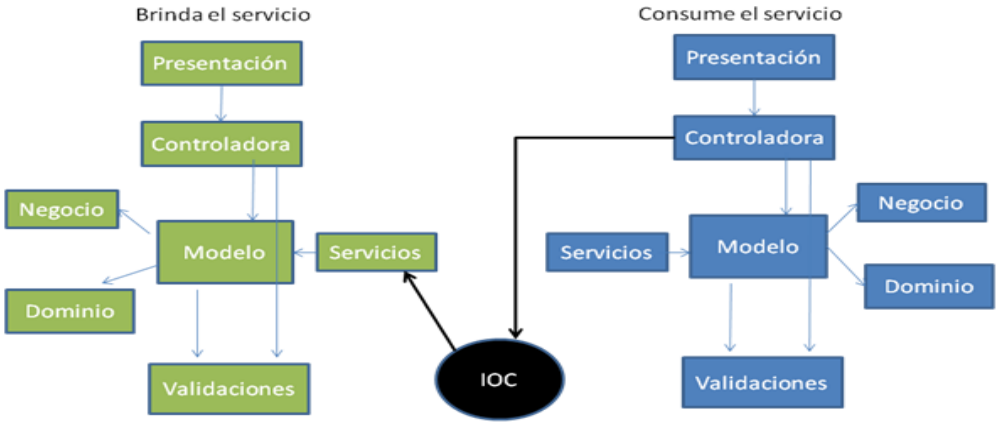

Fig. 8: Integración entre componentes

Entre diferentes módulos y componentes, la integración se basa en el patrón Inversión de Control (IoC) y se realiza a través de un componente incluido en el framework Zend_Ext, que permite operar sobre distintos esquemas en la base de datos realizando las transacciones adecuadas. En dicho componente se

define el fichero ioc.xml que contiene la ubicación de cada uno de los componentes y los servicios que ofrecen las clases services correspondientes. De esta manera, la puerta de entrada de cada componente es el paquete de las clases de servicios que buscan en los modelos o entidades las funcionalidades requeridas. Con la integración se persigue obtener una forma eficiente y flexible de combinar recursos internos o externos de los subsistemas.

Algunos servicios que ofrece el módulo Banco:

- **ObtenerCuentasBancarias**: Servicio del componente Configuración para el módulo Facturación del subsistema Logística que permite obtener las cuentas bancarias de la entidad.
- **ObtenerCuentaPorId**: Servicio del componente Configuración para el módulo Facturación del subsistema Logística que permite obtener la cuenta bancaria por su identificador.

2.4.2 Estándares de codificación

Los estándares de codificación son pautas de programación que no están enfocadas a la lógica del programa, sino a su estructura para facilitar la lectura, comprensión y mantenimiento del código.

Debido a la complejidad del sistema CEDRUX, el numeroso personal involucrado en él y el alto nivel de integración existente entre sus componentes, el grupo arquitectónico del proyecto definió normas de codificación con el fin de obtener un estándar en la implementación por el equipo de desarrollo que permitiera asegurar la calidad del software, obteniendo un código más legible y reutilizable.

A continuación se describen algunos de estos estándares empleados durante la implementación del módulo Banco.

2.4.2.1 PascalCasing

El estándar PascalCasing establece que los identificadores, nombres de clases, variables, métodos o funciones están compuestos por múltiples palabras juntas, iniciando cada palabra con letra mayúscula.

La nomenclatura de las clases del módulo Banco se realizó sobre la base de este estándar, usando palabras compuestas sugerentes acordes al propósito de la misma.

Nomenclatura de clases según su tipo

- ***Controllers***: clases controladoras del negocio.

El nombre de la clase controladora debe estar estructurado por el nombre propio de la misma en mayúsculas seguido por la palabra Controller y heredar siempre de la super clase del framework ZendExt_Controller_Secure. Ejemplo: GestionarCuentaBancariaController

- ***Clases de los modelos***

  - ✓ ***Business***: Clases modelo del negocio.

Las clases modelo tendrán por identificador el nombre de la tabla en la que trabajan seguido por la palabra Model y heredarán de la super clase del framework Zend_Ext llamada ZendExt_Model. Ejemplo: NomCuentaBancariaModel.

  - ✓ ***Domain***: clases entidades del dominio.

Los archivos situados en el domain tienen el mismo nombre de las tablas que representan, definiéndolas como clases php, pueden incluir los prefijos Dat o Nom para diferenciar entre sus usos. Ejemplo: NomCuentaBancaria.

  - ✓ ***Generated***: Clases bases del dominio

Las clases que se encuentran dentro de Generated comienza su nombre con la palabra: “Base”, seguido del nombre de la tabla en la Base de Datos. Ejemplo: BaseNomCuentaBancaria.

- ***Validators***: Clases validadoras.

El nombre de la clase validadoras se especifica con nombres compuestos con el distintivo que terminan en Validator. Ejemplo: CuentaBancariaValidator.

- ***Services***: Clases que ofrecen los servicios de los componentes.

Estas clases, de acuerdo con las operaciones que realizan y prestaciones que brindan, se definen con calificativos sugerentes, agregándoles la terminación Service. Ejemplo: CuentaBancariaService.

2.4.2.2 CamelCasing

El estándar CamelCasing es parecido al PascalCasing con la particularidad de que la letra inicial del identificador no comienza con mayúscula. Esta notación se utilizó para el nombre de funciones y atributos.

Nomenclatura de las funciones

El identificativo a emplear para las funciones o métodos se escribe con la primera palabra en minúscula utilizando la notación CamelCasing y nombres que deduzcan su propósito. Ejemplo: cargarCuentaBancaria.

Los denominadores de las acciones de las clases controladoras tienen la peculiaridad de ir seguidos por la palabra "Action". Ejemplo: cargarCuentaBancariaAction.

Nomenclatura de los atributos

Las variables se nombran convenientemente de acuerdo con el estándar CamelCasing. Ejemplo: idTasaMonetaria y numeroTalonario.

2.4.2.3 Notación húngara

Esta convención, también conocida como notación: REDDICK por el nombre de su creador, se basa en definir prefijos para cada tipo de datos según el ámbito de las variables con el fin de brindar mayor información al nombre de la variable, método o función.

La notación húngara fue utilizada para la definición de variables de acuerdo con los siguientes prefijos:

Tipos de Datos	Prefijos
Arreglos	arr
Objetos	obj
Enteros	int
Cadena	str
Float	flt
Boolean	bool

Tabla 1: Prefijos para la creación de variables.

Nomenclatura de las variables

El nombre a emplear para las variables se escribe siguiendo la notación CamelCasing comenzando con el prefijo según su tipo de datos. Ejemplo: arrIntrumentos define un arreglo de instrumentos.

Nomenclatura de las constantes

La definición de las constantes se realiza utilizando todas las letras en mayúscula. Ejemplo: SUBSISTEMA.

2.4.3 Descripción de clases por componente y tipo

A continuación se describen las clases y sus operaciones más importantes por componentes agrupándolas de acuerdo a la clasificación siguiente:

Clases Controladoras

Las clases controladoras son responsables de la lógica de negocio, gestionando todo el flujo de datos y operaciones entre la vista y demás clases del negocio.

Clases Modelo

Actúan como intermediarias entre las clases controladoras y las clases entidades. Complementan las funcionalidades de las clases controladoras y realizan las funciones de insertar, modificar y eliminar datos.

Clases Entidad

Las clases entidad modelan los objetos del sistema y comportamiento asociado; frecuentemente representan conceptos y datos persistentes de larga duración. Contienen métodos para la gestión consultas y solicitud de información de la base de datos**.**

2.4.3.1 Componente Cierre

Clases Controladoras

Nombre: CierreController	
Tipo de clase: Controladora	
Para cada responsabilidad:	
Nombre:	**Descripción:**
cargarcuentabancariaAction	Carga todas las cuentas bancarias
chequearCierreDiarioAction	Verifica si el subsistema realiza cierre diario
setCierreDiarioAction	Efectúa cierre diario
cerrardiarioAction	Efectúa cierre diario

cerrarcuentabancariaAction	Realiza el cierre para una cuenta bancaria seleccionada.
cerrarperiodoAction	Realiza el cierre de periodo
cerrarejercicioAction	Realiza el cierre de ejercicio

Tabla 2: Descripción de la clase CierreController.

Clases Modelo

Nombre: CierreModel	
Tipo de clase: Modelo	
Para cada responsabilidad:	
Nombre:	**Descripción:**
CerrarPeriodo	Realiza el cierre del periodo
CerrarEjercicio	Realiza el cierre del ejercicio

Tabla 3: Descripción de la clase CierreModel.

Ver descripciones de clases para otros componentes en el Anexo 5 de la versión del documento de Tesis en formato digital.

2.5 Conclusiones

Con el desarrollo de este capítulo se obtuvo la solución del módulo Banco del subsistema de Finanzas como producto configurable del Sistema Integral de Gestión de Entidades CEDRUX, con el objetivo de contribuir al mejoramiento de los procesos financieros de las entidades cubanas, brindando de este modo un impulso decisivo a la informatización de la sociedad.

La descripción del diseño de la solución arquitectónica facilitó la exitosa implementación e integración con el resto de los módulos del propio subsistema. También estableció las bases para un futuro aseguramiento de la gestión y control contable de los procesos financieros bancarios de forma confiable en las entidades. Se garantizó además, a partir de la arquitectura propuesta, el intercambio de información con otros módulos y subsistemas de CEDRUX. Con la solución propuesta se estimula el desarrollo de nuevas aplicaciones que pueden extender las funcionalidades incluyéndoles nuevos procesos de gestión afines a la actividad financiera; garantizando la flexibilidad de la arquitectura así como la independencia tecnológica del país al utilizar el software libre para su desarrollo.

CAPÍTULO III: Validación de la solución propuesta

2.6 Introducción

La dificultad para construir sistemas de software multiplica la probabilidad de que persistan errores aún después de haberse finalizado. Es imposible asegurar que un software se encuentre completamente libre de errores; sin embargo, existen formas y métodos para acercarse lo más posible a un resultado óptimo.

En este capítulo se realiza la validación de la solución propuesta. Se evalúa el diseño empleando métricas de software que proporcionan una medida de la complejidad y calidad del software. Se aplican pruebas con el objetivo de verificar la funcionalidad y estructura de cada componente desarrollado. Y se demuestra la eficiencia temporal de los algoritmos implementados.

2.7 Métricas usadas para la evaluación del modelo de diseño propuesto

Entre las métricas aplicadas durante la evaluación de la solución del diseño del módulo Banco se destacan las siguientes:

2.7.1 Métrica Tamaño Operacional de Clase (TOC)

Tamaño operacional de clase (TOC)	
Descripción:	Está dado por el número de métodos asignados una clase.
Atributos que afecta:	**Modo en que lo afecta:**
Responsabilidad	El aumento del TOC provoca un aumento de la responsabilidad asignada a la clase.
Complejidad de implementación	El aumento del TOC provoca un aumento de la complejidad de implementación de la clase.
Reutilización	Un aumento del TOC provoca una disminución en el grado de reutilización de la clase.

Tabla 4: Métrica Tamaño Operacional de Clase (TOC)

Resultados del instrumento de evaluación de la métrica Tamaño Operacional de clase (TOC)

Ver instrumentos y tabla de resultados para la métrica TOC en el Anexo 6 de la versión del documento de Tesis en formato digital.

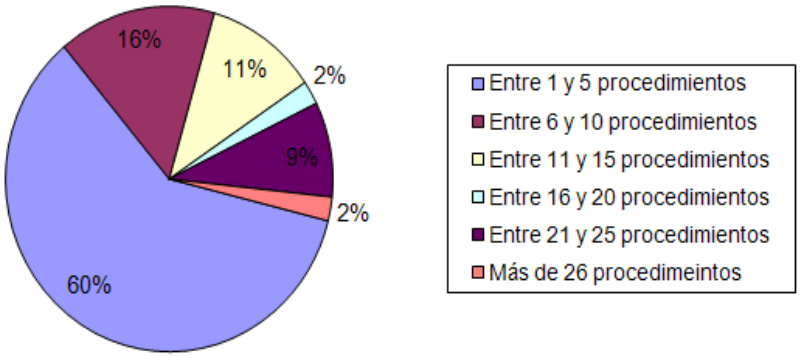

Fig. 9: Representación en % de los resultados obtenidos en el instrumento agrupados en los intervalos definidos.

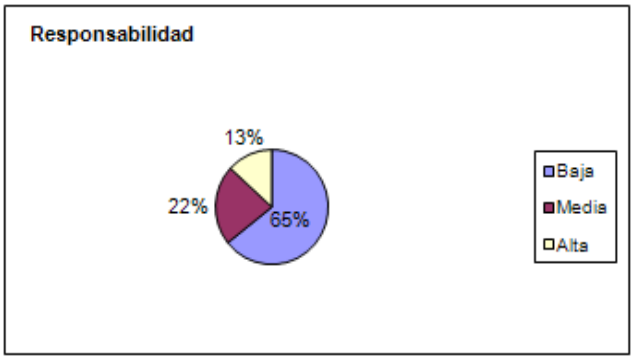

Fig. 10: Representación de la incidencia de los resultados de la evaluación de la métrica TOC en el atributo Responsabilidad.

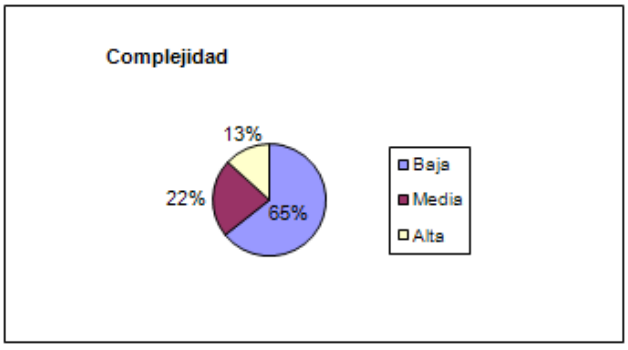

Fig. 11: Representación de la incidencia de los resultados de la evaluación de la métrica TOC en el atributo Complejidad de Implementación.

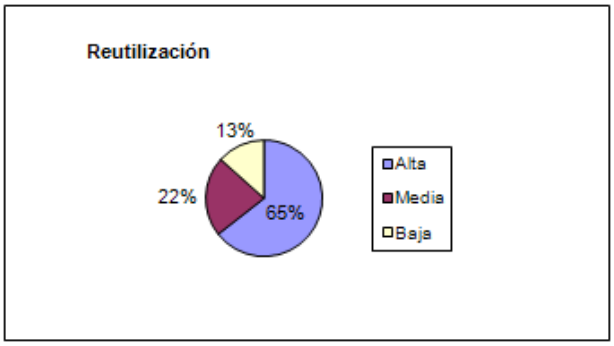

Fig. 12: Representación de la incidencia de los resultados de la evaluación de la métrica TOC en el atributo Reutilización.

Los resultados obtenidos durante la evaluación del instrumento de medición de la métrica TOC demuestran que el diseño propuesto para el módulo Banco se encuentra dentro de los niveles aceptables de calidad, mostrando que más de la mitad de las clases (64,4%) poseen menos cantidad de operaciones que la media registrada en las mediciones. Para el 87% de las clases los atributos de calidad fueron evaluados satisfactoriamente, confirmando la elevada reutilización, baja complejidad y responsabilidad en el diseño propuesto.

2.7.2 Métrica Relaciones entre Clases (RC)

Relaciones entre clases (RC)	
Descripción:	Está dada por el número de relaciones de uso de una clase con otras.
Atributos que afecta:	**Modo en que lo afecta:**
Acoplamiento	El aumento del RC provoca un aumento del Acoplamiento de la clase.
Complejidad del mantenimiento	El aumento del RC provoca un aumento de la complejidad del mantenimiento de la clase.
Reutilización	El aumento del RC provoca una disminución en el grado de reutilización de la clase.
Cantidad de pruebas	El aumento del RC provoca un aumento de la Cantidad de pruebas de unidad necesarias para probar una clase.

Tabla 5 Métrica Relaciones entre Clases (RC)

Resultados del instrumento de evaluación de la métrica Relaciones entre clases (RC)

Ver instrumentos y tabla de resultados para la métrica RC en el Anexo 7 de la versión del documento de Tesis en formato digital.

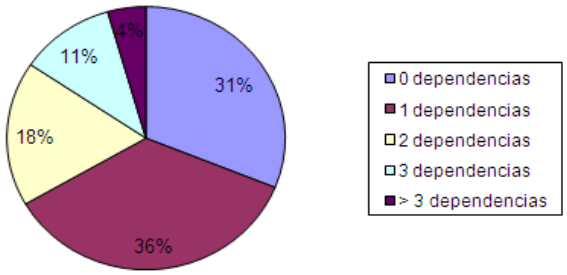

Fig. 13: Representación en % de los resultados obtenidos en el instrumento agrupados en los intervalos definidos.

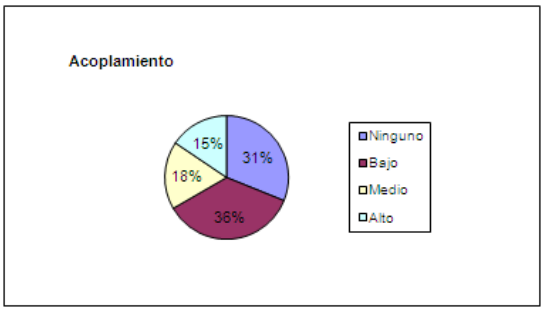

Fig. 14: Representación de la incidencia de los resultados de la evaluación de la métrica RC en el atributo Acoplamiento.

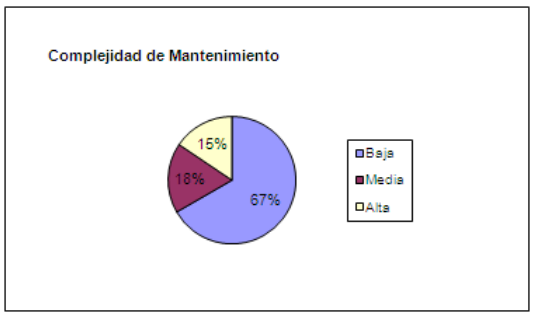

Fig. 15: Representación de la incidencia de los resultados de la evaluación de la métrica RC en el atributo Complejidad de Mantenimiento.

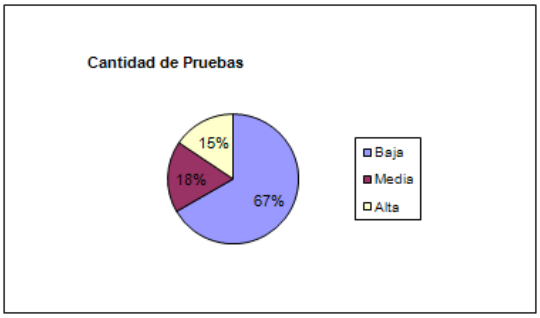

Fig. 16: Representación de la incidencia de los resultados de la evaluación de la métrica RC en el atributo Cantidad de Pruebas.

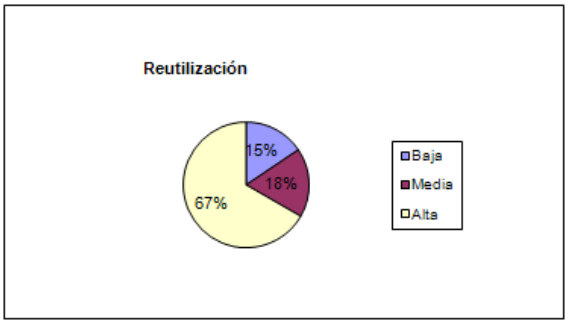

Fig. 17: Representación de la incidencia de los resultados de la evaluación de la métrica RC en el atributo Reutilización.

Los resultados obtenidos durante la evaluación del instrumento de medición de la métrica RC demuestran que el diseño propuesto para el módulo Banco se encuentra dentro de los niveles aceptables de calidad, mostrando que el 84,4% de las clases poseen menos de 3 dependencias entre clases. Los atributos de calidad fueron evaluados satisfactoriamente para el 85% de las clases, confirmando la elevada reutilización y bajo acoplamiento, complejidad y cantidad de pruebas en el diseño propuesto.

Las métricas de software aplicadas posibilitaron estimar la calidad de los atributos internos del producto, demostrando una aceptable calidad de diseño. La solución propuesta contribuirá a la disminución de disturbios durante la implementación del módulo, garantizando la reutilización y agilidad en el proceso de desarrollo de software.

2.8 Niveles de pruebas aplicados

Durante el desarrollo del módulo se definieron pruebas para diferentes objetivos, escenarios o niveles de trabajo. Para aplicarlas, se evalúa el sistema comenzando por los componentes más simples y pequeños, y se avanza progresivamente hasta probar todo el software en su conjunto:

- Pruebas de Desarrollador: Diseñadas e implementadas por el equipo de desarrollo.
- Pruebas Unitarias: Comienzan con la prueba de cada módulo. Verifican que los flujos de control y de datos estén cubiertos, y que funcionen como se espera.
- Pruebas de Integración: Verifican que los módulos probados operen correctamente cuando se combinen y descubren errores en las especificaciones de las interfaces de los paquetes.
- Pruebas de Aceptación: El cliente comprueba que el software funciona según sus expectativas, y se encuentra ejecutando las funciones y tareas para las cuales fue construido.

2.9 Estrategia de pruebas

Para llevar a cabo el proceso de pruebas al módulo se define una estrategia de pruebas con el propósito de garantizar la calidad del software.

Se determina aplicar, a partir de los métodos de caja negra y caja blanca, las técnicas de Partición de Equivalencia y Camino Básico respectivamente, en los niveles de Desarrollador, Unidad e Integración. Estos métodos no se tratan de manera separada a los niveles; al estar estrechamente relacionados no se hace distinción específica de ellos entre uno u otro nivel.

En los niveles de desarrollador y de unidad, a medida que se realice la implementación, se aplicarán pruebas comprobando que cada funcionalidad implementada se ajuste a los requerimientos. En el nivel integración se realizará la verificación y validación de las funcionalidades del módulo, como un conjunto de componentes integrados y combinados según la dependencia jerárquica y la comunicación entre ellos.

De igual forma, las técnicas de prueba no se hallan de forma aislada, sino como un conjunto integrado de acciones que combinadas permitirán verificar y evaluar la calidad de software. Se utiliza la técnica de Partición de Equivalencia para definir casos de prueba que descubran clases de errores, reduciendo el número de casos de prueba a desarrollar para demostrar que las funciones del software son operativas;

las entradas se aceptan de forma adecuada y se producen salidas correctas. Se selecciona la prueba del Camino Básico para obtener una medida de la complejidad ciclomática de los procedimientos, el número de caminos independientes y la cantidad mínima de casos de pruebas a realizar.

Los casos de pruebas se diseñarán basados en los requisitos y el código, comparando cada funcionalidad implementada con la descrita, para verificar hasta qué punto se cumple con las necesidades del cliente.

Independiente a la estrategia de pruebas definida, existe un equipo de calidad encargado de probar el módulo para su liberación posterior a las entidades piloto donde el cliente comprobará que el software funciona según sus expectativas, y se encuentra ejecutando las funciones y tareas para las cuales fue construido.

2.10 Diseño de casos de prueba para caja negra

Los casos de prueba para caja negra se basan en las diferentes entradas que puede recibir el software, y sus correspondientes valores de salida; están centrados en realizar pruebas del software a través de la funcionalidad.

Los casos de prueba demuestran que:

- ✓ Las funciones del software son operativas.
- ✓ Las entradas se aceptan de la forma adecuada produciendo el resultado correcto.
- ✓ La integridad de la información externa (por ejemplo archivos de datos) se mantiene.

Para verificar que la aplicación se comporta según los requerimientos establecidos por el cliente, se diseñan casos de pruebas usando el método de caja negra. A continuación se especifica el caso de prueba para el requisito “Calcular conciliación bancaria” el cual define que dada una cuenta bancaria, se agreguen al saldo inicial, las diferencias detectadas con los estados de cuenta, los cheques que quedan en tránsito, y cualquier otra operación que descuadre con banco; para llegar a un saldo final, donde pueda demostrarse que están cuadrados o conciliados los saldo según banco (saldo final estado de cuenta) y según libros.

Condiciones de ejecución:

- ✓ Se debe identificar y autenticar ante el sistema y además debe tener los permisos para ejecutar esta acción.
- ✓ Se debe seleccionar el subsistema Finanzas.
- ✓ Se debe seleccionar la opción Banco/Configuración/Estado de cuenta/Conciliación bancaria.
- ✓ Debe existir al menos una cuenta bancaria registrada en el sistema.

Nombre del requisito	Descripción general	Escenarios de pruebas	Flujo del escenario
1: Calcular conciliación bancaria.	El sistema calcula la conciliación bancaria de una cuenta.	EP 1.1: Calcular conciliación bancaria especificando los datos válidos.	– Se especifican los datos de la conciliación. – El sistema calcula la conciliación.
		EP 1.2: Calcular conciliación bancaria dejando espacios en blanco.	– Se especifican los datos de la conciliación dejando los espacios en blanco.

Tabla 6: Descripción del caso de prueba para el requisito Calcular conciliación bancaria.

Descripción de variable:

No.	Nombre de campo	Tipo	Válido	Inválido
1	Cuenta bancaria	Lista desplegable	NA	NA
2	Fecha	Campo de fecha	Datos de fecha	Cualquier dato que no tenga formato de fecha.

Tabla 7: Descripción de variables del caso de prueba para el requisito Calcular conciliación bancaria.

Juegos de datos a probar:

Id. del escenario	Escenario	Cuenta	Fecha	Respuesta del sistema	Resultado de la prueba
EP 1.1	Buscar las conciliaciones bancarias que han sido registrados en el sistema.	V(121-Cuenta CUC)	V(15/02/2009)	El sistema calcula la conciliación bancaria.	
EP 1.2	Calcular conciliación bancaria dejando espacios en blanco.	I(Vacío)	V(15/02/2009)	El sistema no muestra nada.	
		V(121-Cuenta CUC)	I(Vacío)		

Tabla 8: Datos de prueba del caso de prueba para el requisito Calcular conciliación bancaria.

2.11 Diseño de casos de prueba para caja blanca

Los casos de prueba de caja blanca no constituyen una alternativa a las técnicas de caja negra, sino un enfoque complementario que intenta descubrir diferentes tipos de errores a los encontrados en los métodos de la caja negra. Permiten elegir y ejercitar una serie de caminos lógicos importantes, que invoquen las estructuras de datos para comprobar su validez. Garantizan que durante la prueba se ejecute cada sentencia del programa.

Los métodos de prueba de la caja blanca permiten obtener casos de prueba que:

- ✓ Se ejerciten por lo menos una vez todos los caminos independientes para cada módulo.
- ✓ Se ejerciten todas las decisiones lógicas en sus vertientes verdaderas y falsas.
- ✓ Se ejecuten todos los bucles en sus límites y con sus límites operacionales.
- ✓ Se ejerciten las estructuras internas de datos para asegurar su validez.

El diseño de los casos de pruebas asegura que las operaciones internas del programa se ajusten a las especificaciones y que todos los componentes sean probados adecuadamente.

Para complementar la prueba de caja negra realizada anteriormente, se aplicó la técnica del camino básico al procedimiento “adicionarconciliacionAction”. (Ver Anexo 8 de la versión del documento de Tesis en formato digital)

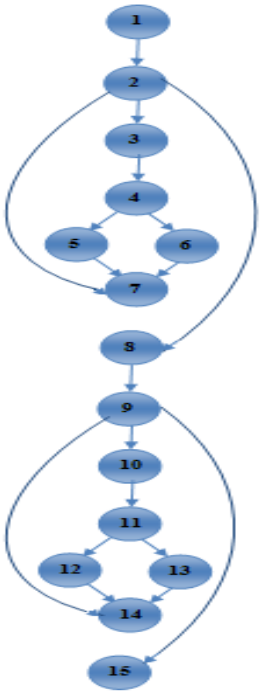

Fig. 18: Grafo de flujo asociado al algoritmo adicionarconciliacionAction.

Acorde al resultado del cálculo de la complejidad ciclomática para el procedimiento, se determina que es necesario diseñar 5 casos de pruebas por el número de caminos básicos independientes a ejecutar; lo que significa que es necesario realizar como mínimo 5 pruebas al algoritmo:

Caso de prueba para el camino básico # 1. (1-2-8-9-15)	
Descripción	Registra una conciliación para una cuenta bancaria de acuerdo con los estados de cuentas de contabilidad y banco contenidos en los arreglos libro y banco respectivamente.
Condición de ejecución	La cuenta bancaria no debe poseer estados de cuentas de contabilidad y banco asociados.

Entrada	libro = array(); banco = array();
Resultados esperados	Se registra una conciliación bancaria satisfactoriamente.
Resultados	Se registra una conciliación bancaria sin estados de cuentas de contabilidad y banco asociados.

Tabla 9: Caso de prueba para el algoritmo adicionarconciliacionAction: camino básico #1.

Caso de prueba para el camino básico # 2. (1-2-3-4-5-7-2-8-9-15)	
Descripción	Registra una conciliación para una cuenta bancaria de acuerdo con los estados de cuentas de contabilidad y banco contenidos en los arreglos libro y banco respectivamente.
Condición de ejecución	La cuenta bancaria debe contener estados de cuentas de contabilidad y no poseer estados de cuentas de banco asociados.
Entrada	libro = {“diferenciamo”=> 100, “idestadodecuenta”=>18712}; banco = array();
Resultados esperados	Se registra una conciliación bancaria satisfactoriamente.
Resultados	Se registra una conciliación bancaria con un estado de cuenta de contabilidad y sin estados de cuenta de banco asociados.

Tabla 10 Caso de prueba para el algoritmo adicionarconciliacionAction: camino básico #2.

El Anexo 9 de la versión del documento de Tesis en formato digital muestra el resto de los casos de pruebas del camino básico para el algoritmo adicionarconciliacionAction.

Las pruebas realizadas no se ejecutaron de forma aislada sino paralelo al desarrollo del software abarcando requerimientos, funciones y lógica interna del programa. Se aplicaron los métodos de caja negra y caja blanca para validar tanto la interfaz como el correcto funcionamiento interno del software. Combinar ambos enfoques permitió lograr mayor fiabilidad en el proceso de pruebas al diseñar los casos de prueba usando los dos tipos de técnicas a la vez.

La validación del software proporcionó un alto grado de confianza y seguridad en el programa y en los resultados que se obtuvieron durante las pruebas.

2.12 Complejidad o eficiencia temporal

Una vez que se dispone de algoritmos que funcionen correctamente, es preciso definir criterios para medir sus comportamientos (rendimiento), centrados fundamentalmente en la simplicidad y la utilización eficiente de los recursos.

Para que un algoritmo sea eficiente debe mantener tan bajo como sea posible el consumo de recursos necesarios en su ejecución atendiendo a dos parámetros:

- ✓ espacio: la memoria que el algoritmo utiliza (complejidad espacial).
- ✓ tiempo: el tiempo que tarda en ejecutarse (complejidad temporal).

La complejidad temporal de un algoritmo representa la medida del tiempo que tarda en operar sobre sus entradas. Para el cálculo del tiempo de ejecución se asocian funciones que tienen un comportamiento similar al que define el algoritmo, entre las que se destacan:

- Logarítmico: $\Theta(\log n)$
- Lineal: $\Theta(n)$
- Quasilineal: $\Theta(n \cdot \log n)$
- Cuadrático: $\Theta(n^2)$
- Cúbico: $\Theta(n^3)$
- Polinómico: $\Theta(n^k)$ con k fijo.
- Exponencial: $\Theta(k^n)$ con k fijo.

Se realiza el análisis del algoritmo “adicionarconciliacionAction”, seleccionado anteriormente en otros epígrafes, con el objetivo de calcular la complejidad temporal del mismo. (Ver anexo 3)

El resultado obtenido O(n) demuestra que es un algoritmo eficiente, con un orden de complejidad relativamente bajo que puede tratar grandes volúmenes de datos, partiendo de que los algoritmos que presentan una complejidad mayor a 2n se consideran intratables o desprovistos de solución.

2.13 Conclusiones

Durante el desarrollo de este capítulo se ejecutaron métricas de diseño para caracterizar numéricamente los distintos aspectos del desarrollo y se realizaron pruebas que permitieron evaluar todos los elementos del software. Fueron validadas las funcionalidades implementadas, a través de diferentes pruebas de caja negra, mostrando cómo respondían adecuadamente a los requisitos funcionales y garantizando la satisfacción plena de las necesidades reales de los usuarios y demandas del cliente. De igual forma, se aplicaron pruebas de caja blanca para efectuar las revisiones al código.

A partir de los resultados de las métricas y las pruebas aplicadas al sistema, se obtuvo un código de mayor calidad, funcionalmente probado, y se evaluaron satisfactoriamente los atributos relacionados con el desarrollo del software. Por último, se demostró la eficiencia de los algoritmos implementados a partir de la complejidad temporal de los mismos.

CONCLUSIONES GENERALES

A partir del análisis de los procesos financieros y sistemas informáticos vinculados a la actividad bancaria, se demostró la necesidad e importancia de desarrollar un sistema informático para realizar la gestión bancaria de forma eficiente en las entidades cubanas.

En tal sentido, se realizó un estudio de las herramientas y tecnologías que posibilitó entender y avalar la selección de técnicas utilizadas. Y se efectuó el diseño e implementación del módulo Banco para el Sistema Integral de Gestión de Entidades CEDRUX, probado y validado mediante métricas y pruebas de software.

De acuerdo con el objetivo general propuesto, se obtuvo el Módulo Banco como producto configurable y totalmente funcional que cumple con los requerimientos descritos, permite el control de los procesos bancarios en las empresas y se adapta a las legislaciones financieras actuales y a las necesidades del proceso de informatización en el país. El módulo además, posibilitará realizar de forma ágil las operaciones contables, integrándose con otros subsistemas necesarios para su correcto funcionamiento.

La investigación realizada contribuye al mejoramiento de los procesos financieros de las entidades cubanas y constituye un aporte decisivo a la informatización y economía del país.

RECOMENDACIONES

Se recomienda:

- Incorporar una nueva funcionalidad que permita al usuario realizar la conciliación bancaria de una cuenta determinada brindando la posibilidad de escoger el método a aplicar.
- Continuar el proceso de pruebas de software al módulo para aumentar la eficiencia y confiabilidad en el mismo.
- Corregir No Conformidades que se identifiquen durante la implantación del módulo en las empresas piloto.
- Extender el proceso de implantación del módulo hacia otras entidades del país.
- Identificar nuevas funcionalidades del negocio a desarrollar en versiones posteriores del sistema.
- De manera general, refinar la solución propuesta a partir de sugerencias y otras recomendaciones.

REFERENCIAS BIBLIOGRÁFICAS

- **ASSETS. 2004.** ¿Qué es Assets? [Online] 2004. http://assets.co.cu/assets.asp.
- **Briano, Fernando. 2008.** Control de versiones con Subversion. [Online] 2008. http://picandocodigo.net/downloads/docs/subversion-presentacion-01.pdf.
- **CBM. 2010.** SAP Enterprise Resource Planning (ERP). [Online] 2010. http://www.gbm.net/soluciones/enterprise_resource_planning.php.
- **CiberAula. 2006.** Una introducción a Apache. [Online] 2006. http://linux.ciberaula.com/articulo/linux_apache_intro/.
- **CITMATEL. 2010.** RODAS XXI: Un producto cubano para la empresa cubana. [Online] 2010. http://www.rodasxxi.cu.
- **Corzo, Giancarlo. 2008.** ExtJs lo bueno, lo malo y lo feo. [Online] 2008. http://blogs.antartec.com/desarrolloweb/2008/10/extjs-lo-bueno-lo-malo-y-lo-feo/.
- **Mª Luisa Melo**.**2006**. "AJAX acelera el desarrollo de interfaces web en las aplicaciones interactivas Representa una amplia colección de las últimas tendencias en tecnologías de Internet" (22/09/2006).http://www.idg.es/computerworld/articulo.asp?id=179064
- **Framework, Zend. 2010.** Introducción a Zend Framework. [Online] 2010. http://manual.zfdes.com/es/introduction.overview.html.
- **Función, Tu. 2006.** Tutorial Básico de Ajax. [Online] 2006. http://www.tufuncion.com/tutorial_basico_ajax.
- **Garlan, Perry. 1995.** Special Issue on Software Architecture : IEEE Transctions on Software Engineering. 1995.
- **GESTIÓN, CENTRO DE SOLUCIONES DE. 2009.** *Definición del ciclo de vida de los proyectos de desarrollo de software v1.0.* 2009.
- **Global, SAP. 2010.** SAP Solutions. [Online] 2010. http://www.sap.com.
- **Gómez, Pablo. 2009.** Instalación de Doctrine ORM en Debian/Ubuntu. [Online] 2009. http://www.arzion.com/empresa-de-internet/posts/Instalacion-de-Doctrine-ORM-en-DebianUbuntu.

- **Henst, Christian Van Der. 2001.** ¿Qué es el PHP? [Online] 2001. http://www.maestrosdelweb.com/editorial/phpintro/.
- **Lic. Miguel P Cabrera González, Msc. Guillermo Obregón Rodríguez, Msc. Margarita Cárdenas Negrin, Lic. Luis Mario Carralero Silva. 2004.** *XV FORUM DE CIENCIA Y TECNICA SISTEMA ECONÓMICO INTEGRADO VERSAT Sarasola.* 2004. p. 26.
- **López, Alejandro Cadavid. 2004.** Mozilla Firefox, el navegador web del momento. [Online] 2004. http://www.maestrosdelweb.com/editorial/firefox/.
- **Manager, Free Download. 2007.** Visual Paradigm para UML. [Online] 2007. http://www.freedownloadmanager.org/es/downloads/Paradigma_Visual_para_UML_%28M%C3%8D%29_14720_p/.
- **Mariaelena. 2010.** Entrevista con funcionales del Ministerio de Finanzas y Precios. *Procesos bancarios de una entidad.* 2010.
- **MP3es. 2009.** Spket IDE. [Online] 2009. http://wiki.mp3.es/Es/Spket_IDE.
- **Openbravo. 2010.** Openbravo ERP: Características relevantes. [Online] 2010. http://www.openbravo.com.
- **Productos, SICS. 2008.** SICS: Productos. [Online] 2008. http://www.sics.cu/productos.aspx.
- **Roger, Pressman. 2005.** *Ingenieria de Software, un enfoque práctico.* 2005.
- **SISCONT. 2009.** SISCONT, Software Contable Financiero. [Online] 2009. http://www.siscont.com/DESCRIPTIVO%20SISCONT.pdf.
- **Systems, Popkin Software and. 2008.** Modelado de Sistemas com UML. [Online] 2008. http://es.tldp.org/Tutoriales/doc-modelado-sistemas-UML/doc-modelado-sistemas-uml.pdf.
- **Tamargo, Lourdes Cerezal. 2009.** La contabilidad en una nueva tecnología. [Online] 2009. http://www.betsime.disaic.cu/secciones/tec_feb_02.htm#1.
- **Torre, Aníbal de la. 2006.** Lenguajes del lado servidor o cliente. [Online] 2006. http://www.adelat.org/media/docum/nuke_publico/lenguajes_del_lado_servidor_o_cliente.html.
- **TuFunción. 2007.** Los mejores IDEs para Php. [Online] 2007. http://www.tufuncion.com/ide-php.

❖ **UptoDown. 2009.** PostgreSQL. [Online] 2009. http://postgresql.uptodown.com/.

BIBLIOGRAFÍA

- **Banco Central de Cuba. 2008.** *Resolución No. 245/08.* Septiembre 26, 2008.
- **Banco Central de Cuba. 2007.** *INSTRUCCIÓN No. 1/07.* Octubre 31, 2007.
- **Banco Nacional de Cuba.. 1994.** *RESOLUCIÓN No. 324/94.* Noviembre 21, 1994.
- **Barco, Antonio. 2006.** Arquitectura Orientada a Servicios (SOA). [Online] 2006. http://arquitecturaorientadaaservicios.blogspot.com/2006/03/pero-qu-es-realmente-soa.html.
- **Centro de soluciones de gestión. 2009.** Ciclo de vida del Proyecto. 2009.
- **Equipo de producción. 2009.** Modelo de Desarrollo orientado a componentes del proyecto ERP - CUBA. 2009.
- **Leyet Fernández, Osmar. 2010.** *Documento De Descripción De La Arquitectura De Software.* La Habana, Cuba. 2010
- **Ministerio de Finanzas y Precios. 2007.** *Resolución No.12/2007.* 2007.
- **Ministerio de Finanzas y Precios.. 2007.** *Resolución No. 14/2007.* 2007.
- **Rolando Alfredo Hernández León, Sayda Coello González. 2002.** *El paradigma cuantitativo de la investigación científica.* 2002.
- **Sola, Elianys Hurtado. 2007.** *ERP-ARQ Manual del marco de trabajo. 2007.*

Anexos

Anexo 1: Modelo de Datos en términos de componentes.

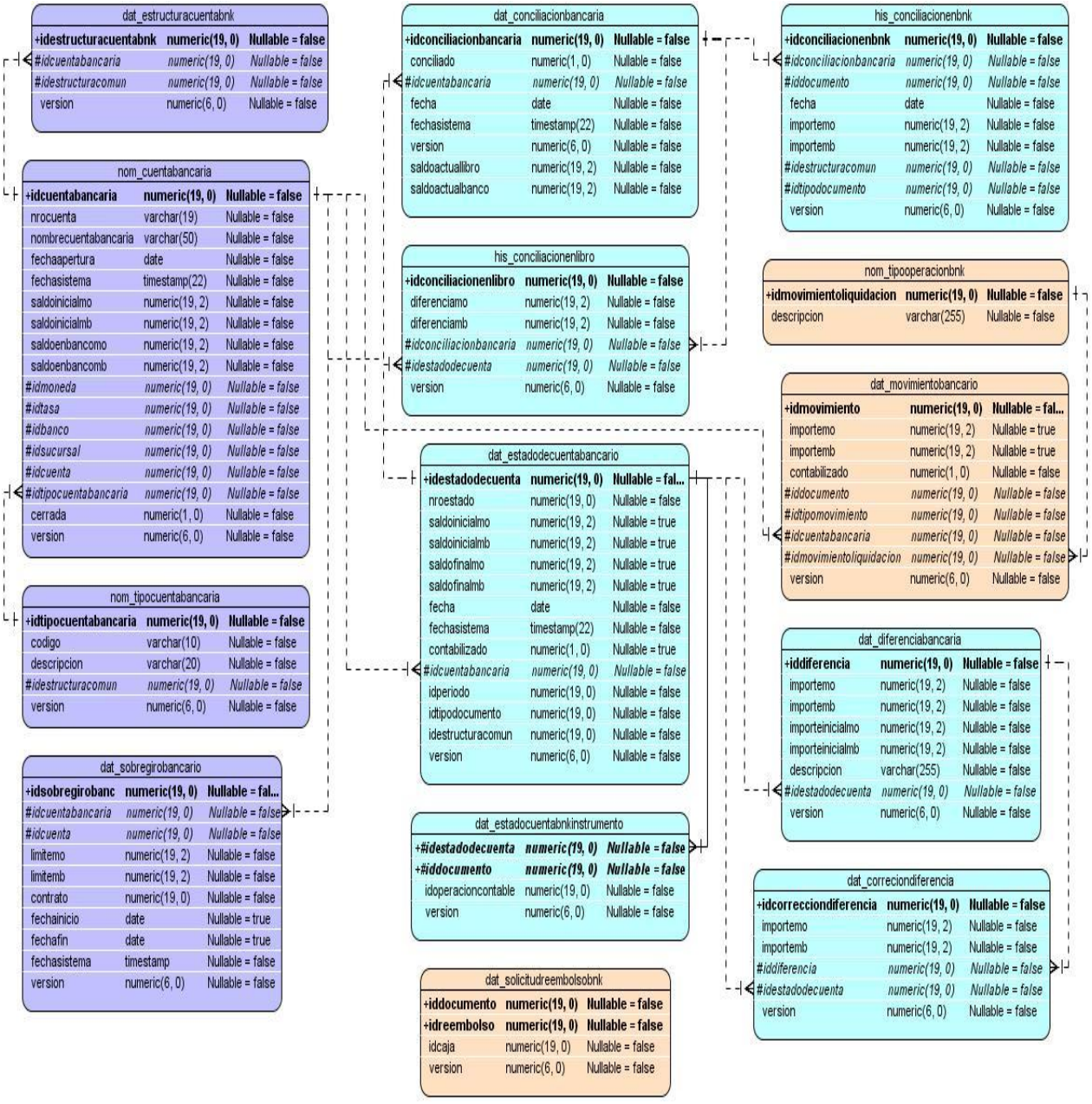

Fig. 19: Modelo de Datos en términos de componentes.

Anexo 2: Diagrama de clases del diseño de los componentes del módulo Banco

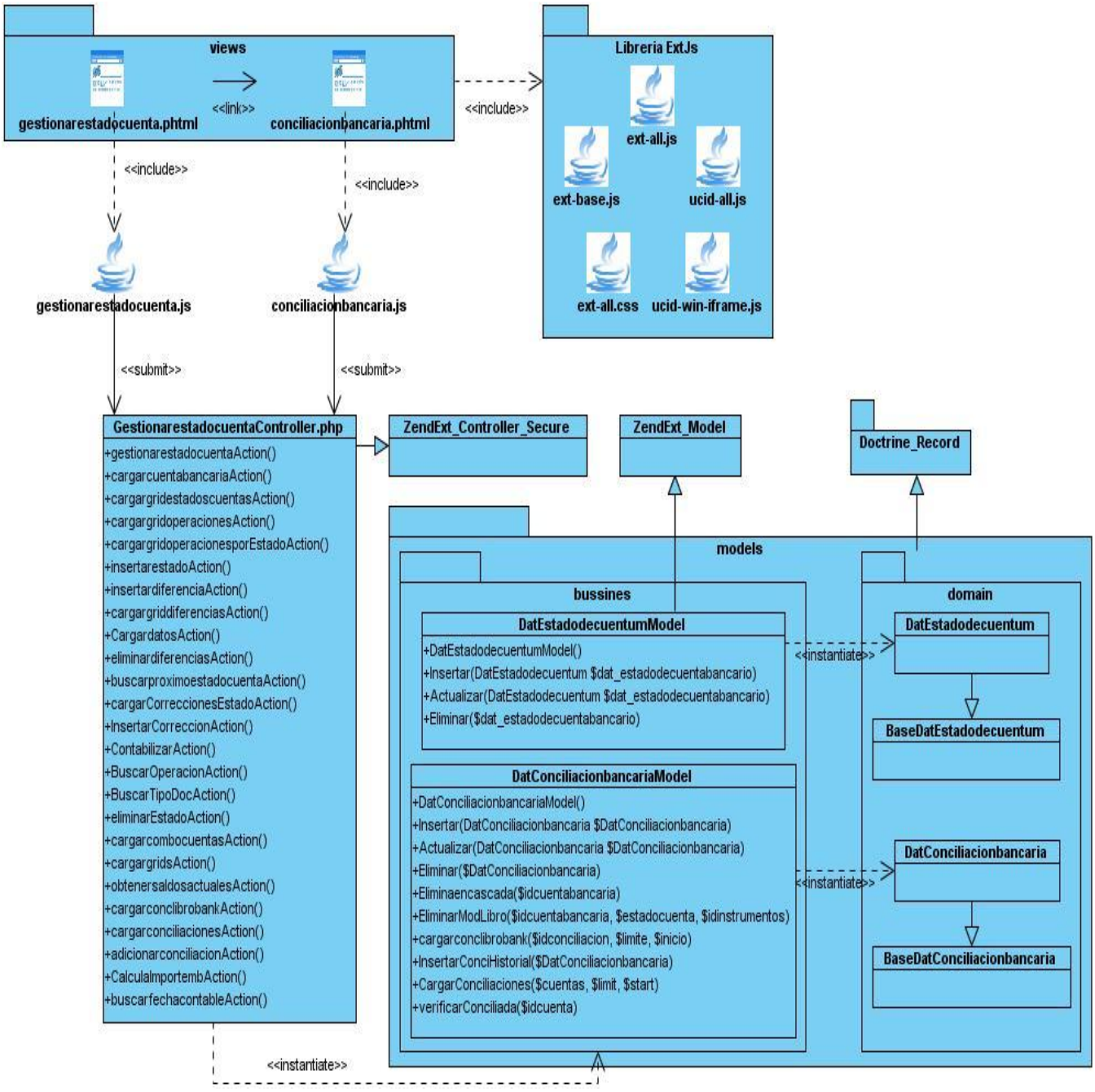

Fig. 20: Diagrama de clases del diseño del componente Estado de cuenta.

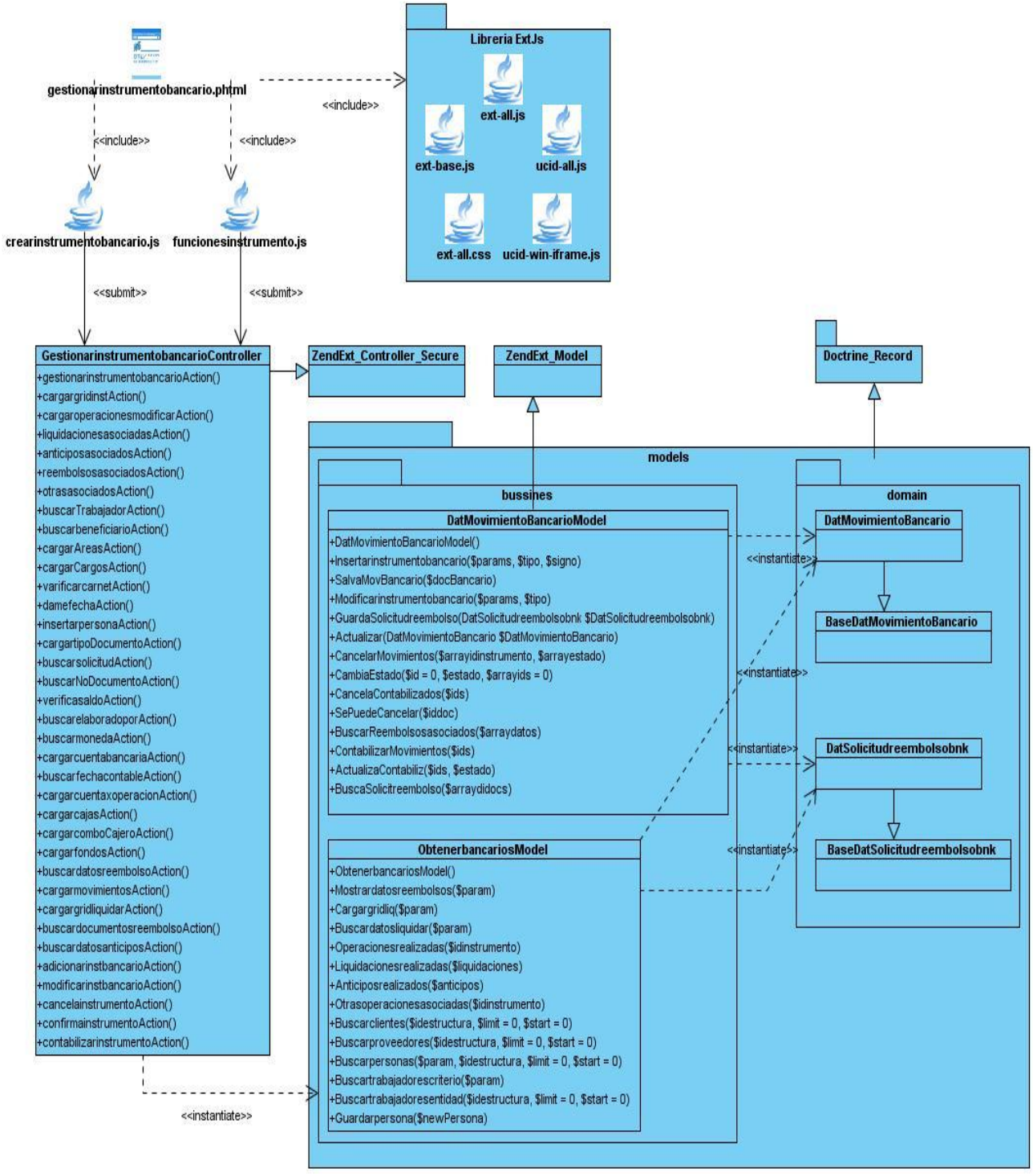

Fig. 21: Diagrama de clases del diseño del componente Instrumento.

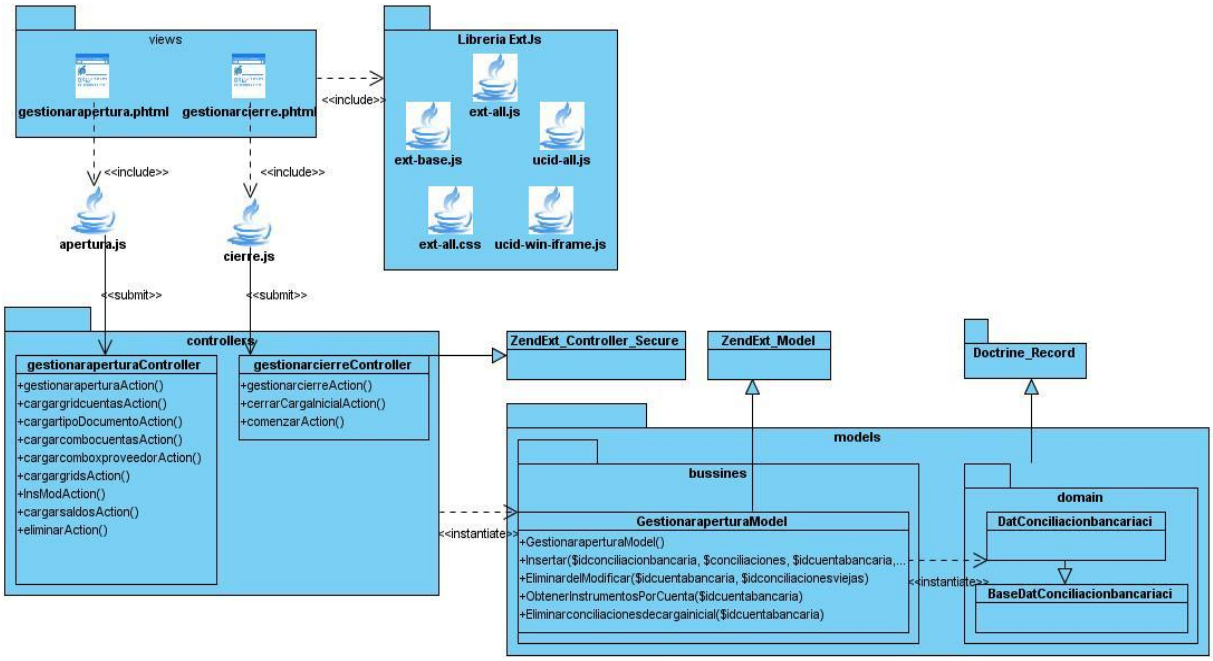

Fig. 22: Diagrama de clases del diseño del componente Carga Inicial.

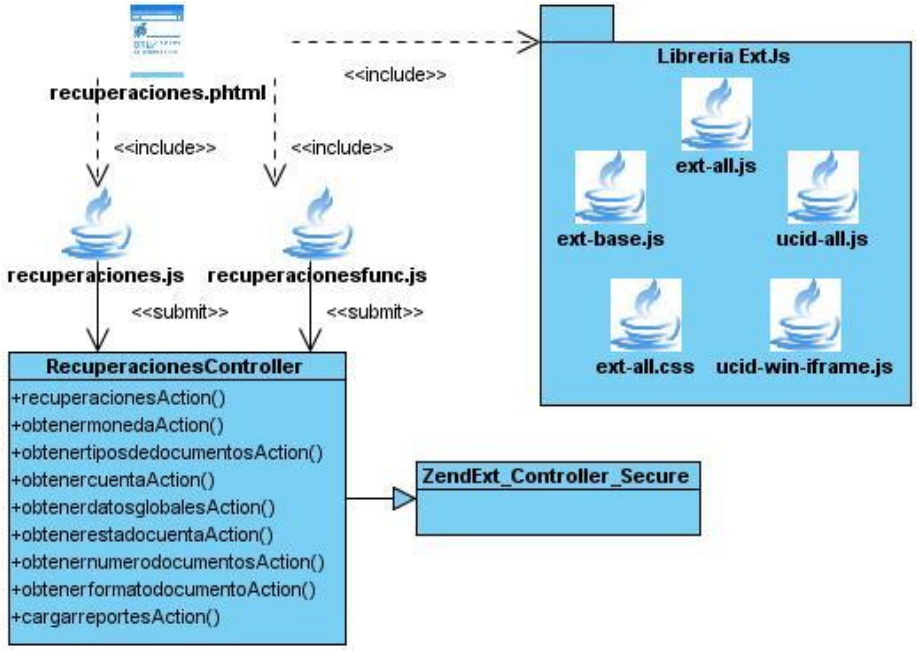

Fig. 23: Diagrama de clases del diseño del componente Recuperaciones.

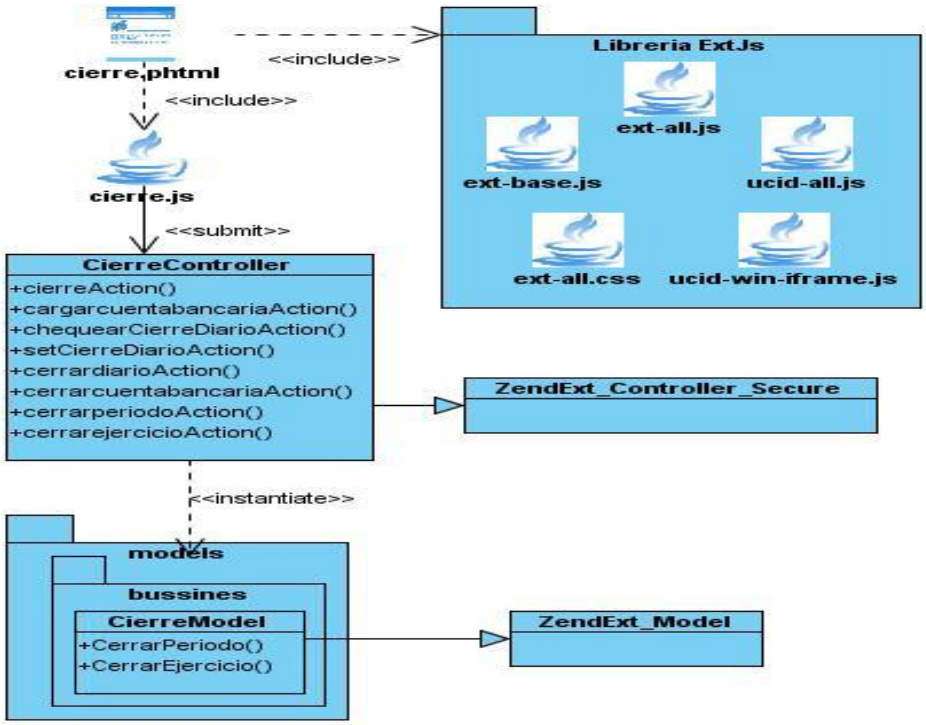

Fig. 24: Diagrama de clases del diseño del componente Cierre.

Anexo 3: Algoritmo adicionarconciliacionAction

```
public function adicionarconciliacionAction()
{   $ident = $this->global->Estructura->idestructura;
    $librobanco = json_decode(stripslashes($this->_request->getPost('librobanco')));
    $saldosactuales = json_decode(stripslashes($this->_request->getPost('saldosactuales')));
    $fecha = $this->_request->getPost('fecha');
    $idcuentabancaria = $this->_request->getPost('idcuentabancaria');
    $conciliado = $this->_request->getPost('conciliado');
    $idmoneda = $this->_request->getPost('idmoneda');
    $libro = $librobanco->libro;
    $banco = $librobanco->banco;
    $DatConciliacionbancaria= array();
    $DatConciliacionbancaria['conciliado'] = $conciliado;
    $DatConciliacionbancaria['idcuentabancaria'] = $idcuentabancaria;
    $DatConciliacionbancaria['fecha'] = $fecha;
    $DatConciliacionbancaria['saldoactuallibro'] = ($saldosactuales->saldolibro) ? $saldosactuales->saldolibro : 0;
    $DatConciliacionbancaria['saldoactualbanco'] = ($saldosactuales->saldobanco) ? $saldosactuales->saldobanco : 0;
    $DatConciliacionbancaria['hisconlibro']=array();

    foreach($libro as $i=>$lib){
        $DatConciliacionbancaria['hisconlibro'][$i]['diferenciamo'] = $lib->diferenciamo;
        if ($lib->diferenciamo > 0)
            $DatConciliacionbancaria['hisconlibro'][$i]['diferenciamb'] = $this->integrator->parametros->CalcImpMonBase($idmoneda, $lib->diferenciamo);
        else
            $DatConciliacionbancaria['hisconlibro'][$i]['diferenciamb'] = 0 - $this->integrator->parametros->CalcImpMonBase($idmoneda, 0 - $lib->diferenciamo);
        $DatConciliacionbancaria['hisconlibro'][$i]['idestadodecuenta'] = $lib->idestadodecuenta;
    }

    $pIntegrator = ZendExt_IoC_Inter::getInstance();
    $DatConciliacionbancaria['hisconbank']=array();

    foreach($banco as $i=>$bank){
        $DatConciliacionbancaria['hisconbank'][$i]['iddocumento'] = $bank->iddocumento;
        $DatConciliacionbancaria['hisconbank'][$i]['importemo'] = $bank->importemo;
        if($bank->importemo > 0)
            $DatConciliacionbancaria['hisconlibro'][$i]['diferenciamb'] = $this->integrator->parametros->CalcImpMonBase($idmoneda, $bank->importemo);
        else
            $DatConciliacionbancaria['hisconlibro'][$i]['diferenciamb'] = 0 -$this->integrator->parametros->CalcImpMonBase($idmoneda,  0 - $bank->importemo);
        $DatConciliacionbancaria['hisconbank'][$i]['fecha'] = $bank->fecha;
        $DatConciliacionbancaria['hisconbank'][$i]['idestructuracomun'] = $bank->idestructuracomun;
        $DatConciliacionbancaria['hisconbank'][$i]['idtipodocumento'] = $bank->idtipodocumento;
    }

    $model = new DatConciliacionbancariaModel;
    $result = $model->InsertarConciHistorial($DatConciliacionbancaria);
    $response->mensaje = ($result) ? 'Conciliaci&oacute;n guardada con &eacute;xito.' : 'Ocurri&oacute; un error al guardar.';
    echo (json_encode($response));
}
```

O (1) O (1) O (n) O (1) O (1) O (1) O (n) O (1) O (n) O (1)

Fig. 25: Cálculo de la complejidad ciclomática del algoritmo adicionarconciliacionAction.

GLOSARIO DE TÉRMINOS

Actividad: define un conjunto de tareas, funciones y definiciones, agrupadas bajo un mismo contexto, teniendo en cuenta la relación entre estas funciones, el lugar y el personal que las realiza, basados además en los principios del control interno.

Algoritmo: es un conjunto finito de instrucciones o pasos que sirven para ejecutar una tarea o resolver un problema.

Base de datos: es un conjunto de datos pertenecientes a un mismo contexto y almacenados sistemáticamente para su posterior uso.

Cheque: mandato de pago en el que se consigna el beneficiario y no se permiten endosos.

Conciliación Bancaria: documento donde ser realiza un análisis por la entidad para determinar las causas de las diferencias existentes entre el Estado de Cuenta del Banco y el saldo en libros para llegar a determinar el saldo correcto.

CSS (Cascading Style Sheets): es un lenguaje usado para definir la presentación de un documento estructurado escrito en HTML o XML (y por extensión en XHTML). Las hojas de estilo en cascada permiten separar la estructura de un documento de su presentación.

Cuenta Bancaria: contrato establecido por el Banco a personas jurídicas o naturales comprometiéndose a prestar los servicios que se hayan acordado del dinero que se deposite en su cuenta.

Datawarehouse: es un almacén de datos orientado a temas, integrado, no volátil, de tiempo variante, que se usa para dar soporte al proceso de toma de decisiones gerenciales.

Documento: modelo que contiene información primaria, y refleja hechos económicos y financieros. Los tipos definidos son: documento de pago, documento de obligaciones, documento de liquidación y documento de letra de cambio.

Documento de Pago: documentos que afectan a una cuenta bancaria, aumentando o disminuyendo su saldo. En esta clasificación se incluyen los instrumentos y efectos de pago, recibo de ingresos en efectivo y otras órdenes de cobros y pagos.

DOM (Document Object Model): es un conjunto de utilidades específicamente diseñadas para manipular documentos XML o XHTML y HTML. Técnicamente, es una API de funciones que se pueden utilizar para manipular las páginas XHTML de forma rápida y eficiente.

Dualidad monetaria: existencia de dos monedas que coexisten y comparten legalmente las funciones del dinero en la economía nacional.

Entidad: organización administrativa, comercial, económica, productiva y de servicios de carácter estatal, cooperativa, privada o mixta, residentes en el territorio nacional; así como las organizaciones sociales y de masas del país.

Estado de Cuenta: documento o relación detallada del movimiento de una cuenta bancaria en una fecha determinada y el saldo al final del mismo.

Fichero: es una manera particular de codificar información para almacenarla en un archivo informático.

Letra de cambio: título-valor que obliga a pagar una deuda a su vencimiento en un lugar determinado a favor de quien resulte su legítimo tenedor, se ajusta a las formalidades que establece la ley.

Multimoneda: utilización de varias monedas en las transacciones económicas y su registro; funcionalidad que supone la existencia de una moneda base y de varias monedas que tienen convertibilidad con relación a la moneda base.

Open Source: código fuente disponible públicamente.

Operación: operación a toda acción sobre el documento que genera asientos contables y que transfiere al documento de un estado a otro.

Operaciones Anticipadas: emisión o recepción de un pago anticipadamente a la obligación que está siendo pagada.

Operaciones Bancarias Automáticas: operaciones o transacciones realizadas por el banco ya sean por solicitud propia de la entidad o por operaciones que incluya el banco.

ORM (Object Relational Mapping): es una técnica de programación para convertir datos entre el lenguaje de programación orientado a objetos utilizado y el sistema de base de datos relacional. El mapeo objeto-relacional posibilita el uso de las características propias de la orientación a objetos (básicamente herencia y polimorfismo).

Pagaré: título-valor que constituye un reconocimiento de deuda por escrito o promesa de pago de una suma de dinero, hecha a la persona del acreedor.

Período contable: intervalo de tiempo en que serán registrados los hechos económicos acaecidos en una entidad. El período contable puede ser un día, una semana, un mes o un año.

Protocolo PDO: permite la transmisión de datos en tiempo real y con identificadores de alta prioridad hacia/desde el bus evitando sobrecargarlo con información redundante.

SOA (Service Oriented Architecture): Arquitectura de Software que propone la integración y el desarrollo de aplicaciones construidas sobre los servicios y la composición de servicios

Talonario: cuadernillo con el fin de mantener el control de la consecutividad numérica en el uso de los documentos primarios, al emitir un pago.

Transferencia bancaria: el banco la realiza siguiendo instrucciones de su cliente, debitando la cuenta del cliente por la cantidad objeto de la transferencia y acredita la cuenta del beneficiario.

Unidad usuaria: unidad contable que hace uso en las cuentas bancarias o fondos de efectivo en caja de otra unidad contable.