UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS

Facultad 6

Subsistema Monitoreo de Operaciones en Cuentas de Clientes del Sistema Quarxo

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

AUTOR

Enrique Capote Pasanaut

TUTORES

Ing. Lissett Díaz Mesa

Ing. Manuel Alejandro Borroto Santana

La Habana, junio del 2013

"Año 55 de la Revolución"

PENSAMIENTO

"Seamos realistas, soñemos lo imposible"

DECLARACIÓN DE AUTORÍA

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

Para que así conste firmo la presente a los ___ días del mes de _____ del año _____.

Autor:

_____

Enrique Capote Pasanaut

Tutor:

_____

Ing. Lissett Díaz Mesa

Tutor:

_____

Ing. Manuel A. Borroto Santana

AGRADECIMIENTOS

Resulta difícil mencionar a todas las personas que me ayudaron a alcanzar este momento tan importante. No podría herirlas de ninguna manera porque mi gratitud por ellas nunca será parte del olvido y si en mi condición humana de no ser perfecto, no menciono algún nombre, sepan que no quedo mal con ellos, sino conmigo mismo. Gracias por las experiencias compartidas; ya fueran positivas o negativas, de ellas aprendí mucho.

Quiero agradecer antes que a nadie, a la persona que me dió la vida y la razón en primer lugar por la que emprendiera este viaje, mi madre, de quien heredé una fuerza de voluntad inquebrantable.

A mi abuelo quien le debo tanto y siempre ha sido el indicador de que en el mundo aún existe lo correcto.

A mi abuela por aguantar todas mis malacrianzas y a pesar de ello seguir queriéndome; eso sí es una tarea difícil.

A mi hermano, mi padre y mi familia toda, que de una forma u otra se preocupó por que no perdiera el camino.

A Sonia, por mostrarme que siempre hay luz aunque todo esté oscuro, por su cariño, apoyo y ser una amiga sincera.

Quisiera agradecer a todos mis profesores, que me aportaron el conocimiento y la preparación para realizar mi sueño. A mis amigos, los verdaderos que me apoyaron en cada contratiempo y gozaron con mi júbilo, a Tito, Rayner, Nilmar, Greisy, Arrebato y Wendy. A mis compañeros de aula del CPT6D con los que pasé buenos momentos, Portillo, Tony, Yanisleydis, Aquiles, Javier, Cariheilys, Osvaldo.

Mi más sincera gratitud a mis tutores Lissett y Manuel, así como a los especialistas del CEIGE, que siempre me brindaron sus conocimientos y ayuda incondicional. A Héctor, Rafael, Yoan, Adrián, Alejo, Efrain, Alain, Asdrubal, Javier, Eduardo, Ray, Jorge, Matías, Dariel y Evelio. Mi abrazo más cálido a mis socios del proyecto Banco de la facultad 3, A Radamés (2pa0), Jorgito (Ushiha), Silvio, Luis Carlos (Overture), Raúl, Reinier, Revens, Yanito, Yadelis y Raylith (Kiki).

A la profesora Yanelis por darme una segunda oportunidad.

A Omar por su apoyo y comprensión.

A Lázaro (Lacho) por su tiempo dedicado a orientarme.

A mis compañeros de trabajo de la dirección de laboratorios que siempre recuerdo con tanta alegría.

A todas esas personas que contribuyeron a que pudiera terminar este trabajo, gracias.

DEDICATORIA

A mi madre por enseñarme a levantarme.

A mis abuelos por ser lo más grande que me dio la vida.

RESUMEN

La informatización del sistema bancario cubano, como parte del proceso de digitalización que requiere el país, demanda una alta capacidad tecnológica y operativa, lo que trae consigo el empleo de herramientas computacionales para el procesamiento de la información.

La Universidad de las Ciencias Informáticas ha desempeñado un papel protagónico en ese sentido, de manera que está inmersa en el continuo desarrollo y soporte del sistema *Quarxo*. Dicho sistema responde de manera rápida y certera a la actividad contable y financiera que se realiza en el Banco Nacional de Cuba. Dentro de los procesos más importantes que aún no se encuentran automatizados en el banco citado, se ubica la gestión de monitoreo de operaciones en cuentas de clientes. Con el propósito de llevar a cabo dicha funcionalidad, el presente trabajo comprende el Análisis, Diseño y la Implementación del subsistema Monitoreo, que será incorporado al actual sistema en explotación.

Como guía en el proceso de desarrollo se optó por el Modelo de Desarrollo de Software para el CEIGE. Se utilizaron diversos patrones de diseño, herramientas, lenguajes y tecnologías de la plataforma *Java*, empleándose eficazmente las potencialidades del software libre. Se elaboraron los artefactos propios de los flujos de trabajo del desarrollo de software involucrado y se validó la solución propuesta. Para el logro de los objetivos fue necesario introducir mejoras en los subsistemas Clientes y Cuentas de Clientes. Finalmente se obtuvo el subsistema Monitoreo de Operaciones en Cuentas de Clientes, compuesto por los módulos Expediente y Alerta.

Palabras clave: Monitoreo, Expediente, Alerta, *Quarxo*, Diseño.

ÍNDICE DE CONTENIDOS

ÍNDICE DE ILUSTRACIONES

ÍNDICE DE TABLAS

INTRODUCCIÓN

Las Tecnologías de la Información y las Comunicaciones (TIC) se desarrollan en la actualidad de manera exponencial, influyendo en todas las esferas sociales. La inclusión de las TIC en la vida cotidiana ofrece grandes oportunidades como el mejoramiento del entorno social, político y económico, aumentándose consigo la tendencia hacia la digitalización

Con la creciente liberalización e integración de los mercados financieros mundiales, el libre movimiento de capitales, así como la rapidez y eficacia de la transferencia de dinero por vía digital, se ha consolidado el sistema financiero global. Por otra parte también se ha hecho más vulnerable a diversos actos delictivos y amenazas, tales como estafas electrónicas, estratagemas para infiltrar dispositivos electrónicos, blanqueo de dinero mediante comercio electrónico y accesos indebidos a sistemas informáticos de las instituciones de banco, con la finalidad de realizar transferencias ilegales de recursos.

Respecto a los tópicos mencionados el *Norton Cybercrime Report* dio a conocer, que los costos financieros directos de los ataques cibernéticos a nivel mundial ascienden a 114 mil millones USD, además de 274 mil millones USD que se calculan por pérdida de tiempo productivo. Estas cifras son superiores incluso a las que genera el mercado negro mundial de la venta de drogas, que alcanzó la cifra de 288 mil millones de USD. (1)

Con el objetivo de poner frenos a esta situación se forma la Oficina de Lucha contra el Fraude (OLAF), comisión presupuestada por la Unión Europea (UE), que entre sus tareas y responsabilidades se encuentran investigar el fraude financiero y elaborar una política de lucha contra el mismo. (2)

En Cuba las entidades bancarias combaten las ilegalidades financieras, apoyándose en la instrucción 26/2006 dictada por el Banco Central de Cuba (BCC). La misma representa una guía donde se establecen una serie de pautas para prevenir, detectar, enfrentar e impedir las operaciones de movimiento de capitales ilícitos o lavado de dinero, así como operaciones indebidas en los cobros y pagos. Se debe aplicar tomándose en cuenta la observación de las normas dictadas en la resolución 60/11, por la Contraloría General de la República sobre el control interno. Independientemente de estas medidas, cada entidad bancaria las implementa de acuerdo sus necesidades. (3)

Como parte de la colaboración que mantiene la Universidad de las Ciencias Informáticas (UCI) en la informatización de las entidades del país, se implantó en el Banco Nacional de Cuba (BNC) desde

junio del 2012, el sistema de gestión bancaria *Quarxo*. El mismo fué desarrollado por uno de los proyectos productivos del Centro de Informatización de la Gestión de Entidades (CEIGE).

A raíz de las crecientes necesidades del BNC, el cliente solicitó la realización de una segunda fase del sistema, con el objetivo de incorporar al mismo un grupo de funcionalidades no incluidas en su primera versión. El monitoreo de las cuentas de clientes constituye una de las solicitudes, a resolverse con la siguiente entrega del sistema en explotación. Para cumplir con esta tarea, es necesaria la revisión de los expedientes de las cuentas de clientes y el seguimiento de las transacciones.

Los expedientes de las cuentas de clientes comprenden varios modelos tales como el Conozca a su Cliente y la Debida Diligencia, documentos que hoy no se encuentran digitalizados, lo que propicia un control pobre e ineficiente.

El proceso de seguimiento de las transacciones se realiza de forma aleatoria y manual, implicando que en la mayoría de los casos no se detecten operaciones fraudulentas, o estas se identifiquen días después de la ocurrencia del hecho, afectándose con ello a la economía nacional, la imagen de la entidad y la confianza de sus clientes.

Por la situación anteriormente expuesta se presenta como **problema a resolver:** ¿Cómo monitorear las operaciones diarias en las cuentas de clientes del BNC?

Para darle respuesta al problema de la investigación se trazó como **objetivo general:** Desarrollar el subsistema Monitoreo de Operaciones en Cuentas de Clientes para el Sistema *Quarxo*.

Se definió como **objeto de estudio** el proceso de monitoreo en cuentas de clientes enmarcado por el **campo de acción,** informatización del monitoreo en cuentas de clientes del BNC.

Se plantea como **idea a defender**, que con el desarrollo del subsistema Monitoreo para el sistema Quarxo, se podrán monitorear las operaciones en cuentas de clientes del BNC.

Para la organización del trabajo se proponen los siguientes **objetivos específicos:**

1. Fundamentar la investigación mediante la elaboración del marco teórico.
2. Describir procesos e identificar requisitos para el subsistema de monitoreo de operaciones.
3. Diseñar e Implementar el subsistema de monitoreo de operaciones.
4. Validar el subsistema implementado.

Para dar cumplimiento a los objetivos trazados, las **tareas de la investigación** estarán encaminadas a:

1. Caracterización de las tecnologías, herramientas y métodos seleccionados por el proyecto de desarrollo para la implementación de la solución.
2. Caracterización de los patrones de diseño para la búsqueda de soluciones a problemas comunes en el desarrollo.
3. Caracterización de los procesos, regulaciones, instrucciones y sistemas informáticos que se emplean actualmente para el monitoreo de operaciones de clientes en las instituciones financieras.
4. Identificación, especificación y validación de los requisitos de software basado en el modelo de desarrollo del centro CEIGE.
5. Realización del modelo del análisis y diseño apoyado en el modelo de desarrollo del centro CEIGE.
6. Realización del modelo de Implementación de la solución a partir del modelo de desarrollo del centro CEIGE.
7. Realización de las pruebas unitarias y de integración.
8. Validación del subsistema de monitoreo de operaciones.

En la investigación se emplearon diferentes **métodos científicos**.

Dentro de los métodos teóricos:

- Análisis histórico - lógico: Se utiliza para el análisis de los procesos de monitoreo y control en diferentes entidades de banco, a partir de la investigación en diferentes momentos históricos.
- Modelación: Se emplea para representar por medio de diagramas el proceso de monitoreo de operaciones en cuentas de clientes, obteniéndose como resultado un mejor entendimiento de la posible solución a implementar.
- Análisis - síntesis: Para comprender y tener un mejor entendimiento del proceso de monitoreo y control de las operaciones en cuentas de clientes, este se dividió en varias partes para conocer los elementos que lo componen y luego se fusionaron para definir las características y las relaciones entre ellos.

Técnicas utilizadas:

- Observación**:** Se utilizó para distinguir las características de las operaciones en cuentas de cliente del Banco Nacional de Cuba, así como para observar el desarrollo y funcionamiento del subsistema.
- Entrevista: Se utilizó de manera informal para la recogida de información.

Posibles resultados de esta investigación:

1. El registro de los expedientes de clientes del Banco Nacional de Cuba normalizados y digitalizados, accesibles para la operadora de monitoreo y control para requeridas consultas y realización de chequeos.
2. Un subsistema para el monitoreo de las operaciones en cuentas de clientes.
3. Alertas para reforzar el control a partir de la integridad de la información socio - económica para la gestión bancaria del Banco Nacional de Cuba.

El presente trabajo está estructurado en tres capítulos.

Capítulo 1 Fundamento Teórico: En este capítulo se abarcan los elementos teóricos asociados a la investigación. Se realiza un estudio del estado del arte de las herramientas que se emplean para el monitoreo de operaciones en cuentas de clientes y se describen las diferentes tecnologías que se emplearán para el desarrollo de la investigación.

Capítulo 2 Propuesta de Solución: En este capítulo se realiza una caracterización del sistema, identificando trabajadores, involucrados y artefactos dentro de los procesos del negocio. Se definen y especifican los requerimientos funcionales del subsistema. Además se efectúa el diseño de la solución, donde se esbozan todas las clases para cada uno de los objetos determinados y se propone un modelo de datos que sustente las clases desarrolladas.

Capítulo 3 Implementación y validación de la solución propuesta: Este capítulo se centra principalmente en la implementación del subsistema Monitoreo de Operaciones en Cuentas de Clientes. Se explica la interacción entre componentes del sistema a través de diagramas y se describen un subconjunto de clases y funcionalidades. Para la validación se realizan pruebas de unidad (estructurales y funcionales). Se brinda una explicación detallada de las pruebas de caja blanca que se realizan al código del software y de las pruebas de caja negra que se efectúan a la interfaz del mismo.

CAPÍTULO 1 FUNDAMENTO TEÓRICO

1.1 Introducción

En este capítulo se abordan aspectos fundamentales que conforman el soporte teórico para el desarrollo de la investigación. Se analizan algunos sistemas existentes que tienen integradas herramientas para el monitoreo de operaciones en cuentas de clientes, a fin de seleccionar características relevantes que puedan resultar necesarias al BNC para la gestión de monitoreo. Además, se describen las características de la metodología de desarrollo, lenguaje de programación, patrones, *frameworks* y otros elementos propuestos para lograr un desarrollo exitoso del subsistema.

1.2 Términos asociados al dominio del problema

Conozca a su cliente: Es una política que implementan los bancos a nivel mundial para identificar a sus clientes ocasionales o habituales. Tiene como base un documento de identificación viable, mediante el cual pueden establecer medidas para reconocerlos debidamente y tener un conocimiento de sus transacciones y actividades, en el orden de determinar la coherencia entre ellas. (3)

Una forma que permite aplicar esta política, es por mediación del nombrado “Conozca a su Cliente”, documento legal que reúne datos sobre el cliente y se actualiza anualmente.

Debida Diligencia: Grupo de acciones que forman parte de una política del ámbito financiero, aplicada a nivel mundial para evitar riesgos operativos, legales, de cumplimiento e incluso de reputación. Enmarca un conjunto de procesos necesarios para poder captar y evaluar información de forma sistemática, permitiéndose identificar la naturaleza real y los riesgos de cualquier operación, para poder actuar de forma certera y asumir decisiones con suficiente información. (3)

El BNC aplica esta política en sus relaciones con el cliente, mediante la comprobación de los siguientes elementos:

- Identificación del cliente.
- Verificación de la información suministrada por el cliente.
- Monitoreo continuo de la(s) cuenta(s) y sus operaciones.
- Vigilancia sistemática en cuanto a si cumple con su objeto social.
- Análisis de los estados financieros para entender la actividad del cliente y verificar si mantienen relación con las operaciones que realiza y la información suministrada.
- Conservación de los documentos de identificación y de las operaciones realizadas.
- No apertura de cuentas anónimas o cuentas con nombres evidentemente ficticios.

1.3 Mecanismos para el monitoreo de operaciones

Con el transcurso del tiempo las organizaciones se han tornado muy dependientes de las tecnologías de la información y las comunicaciones, con el objetivo de aprovechar las comodidades que estas brindan para realizar y controlar sus operaciones.

Los sistemas de información utilizados pueden ser simples o complejos, dependiendo de varios factores como la cantidad de operaciones, el tamaño de la empresa y la dispersión geográfica de los centros de trabajo que la conforman.

En este sentido conviene señalar, que la mayoría de los sistemas y software existentes en la actualidad tienen una arquitectura basada en los conceptos *ERP* (en inglés: *Enterprise Resource Planning*, en español: Planeación de Recursos Empresariales), *SCM* (en inglés: *Supply Chain Management*, en español: Administración de la Cadena de Abastecimiento), *CRM* (en inglés: *Customer Relationship Management*, en español: Administración de Relaciones con los Clientes) y *BI* (en inglés: *Business Intelligence*, en español: Inteligencia de Negocios). Cada uno de estos sistemas tiene una serie de módulos, cuyas operaciones son registradas en una base de datos central, a la cual se accede mediante una red desde lugares remotos, independientemente del lugar donde operen.

Todo lo anterior expuesto queda bajo la denominación genérica de *e-Business*. Algunos de los módulos que se pueden citar son los de recursos humanos, adquisiciones, almacenes, tesorería, ventas, proyectos de inversión, presupuesto y contabilidad.

En la actualidad se cuenta con una gran variedad de software, tanto para la administración del área de auditoría, como para la extracción y análisis de datos, monitoreo continuo de las operaciones o administración de riesgos. Al contar con esta gama de herramientas, el departamento de auditoría cuenta con la flexibilidad necesaria para aplicar la mejor tecnología a cada tarea específica, dependiendo también de diversos factores y de la complejidad, objetivos, alcances y tareas a lograr por cada revisión.

En este sentido los auditores pueden utilizar software especializado, sistemas con el propósito de analizar y verificar información, principalmente en cuanto al cumplimiento normativo. Esta clase de enfoque es considerado una mejor práctica por la *IIA* (en inglés: *Institute of Internal Auditors*).

La auditoría de monitoreo continuo está compuesta de procedimientos de auditoría automatizados, diseñados para recabar información en línea directamente de las bases de datos de los sistemas de información. Dichos procedimientos tienen el propósito de evaluar las operaciones de la organización y

el grado de cumplimiento de la normatividad externa e interna. Su implementación consiste en la evaluación del control interno de las operaciones para identificar riesgos y con base en estos diseñar los procedimientos automatizados de auditoría, que permitan formular propuestas para administrar estos riesgos y realizar revisiones periódicas a estas operaciones, vigilándose de esta manera su cumplimiento, tanto por la misma área como por auditoría interna.

Para iniciar los trabajos se debe contar con auditores de perfil informático y experiencia en el proceso a revisar, así como una capacitación continua sobre herramientas y lenguajes informáticos, pues por lo general, es necesario conectarse a las bases de datos que contiene la información a revisar. Asimismo, se requieren auditores financieros u operacionales de preferencia especializados en los procesos a revisar, para dirigir y explotar al máximo la información.

Adicionalmente se necesita acceso a la red, licencias de software y claves de usuario con permisos de consulta a los sistemas. Lo anterior permite mantener independencia del área de auditoría, respecto de las áreas usuarias e informáticas, al tener acceso directo a los lugares donde se almacena la información. (4)

Una vez satisfechas estas condiciones, se deben realizar las actividades siguientes:

1. **Análisis del proceso seleccionado e identificación de riesgos:** Es importante evaluar y conocer el proceso para identificar los riesgos y los puntos de control que los mitiguen. Asimismo, la normatividad que regula el proceso para ser incluida en el análisis a realizar, ya que contiene las condiciones a las que se apega el proceso y las desviaciones que pueden convertirse en excepciones, debilidades o deficiencias de control.
2. **Definición y diseño de monitoreo:** Una vez identificados los riesgos, se puede realizar el modelo conceptual del monitoreo, a partir de las debilidades identificadas y contemplar los controles existentes en el proceso. Posteriormente se diseña el monitoreo en sí, analizándose el procesamiento y la arquitectura de la información, así como el registro informático de las operaciones. A partir de todo lo anterior, se desarrollan los procedimientos automatizados, determinándose las consultas a realizar y los parámetros a cumplir.
3. **Obtención de herramientas informáticas:** Una vez concluido el diseño, se puede identificar la herramienta a utilizar de manera individual o combinada. A continuación se mencionan algunas:
    - Paquetes de auditoría (ACL, *IDEA* y *Fastsearch*).
    - Módulo de auditoría de SAP (en inglés: AIS, *Audit Information System*).
    - Herramientas de extracción y análisis de información (*Business Objects y Access*).
    - Paquetería genérica (*Microsoft Office*).

- Utilerías de bases de datos (*Oracle y SQL Server*).
- Lenguajes de programación (*Java*, C++ y *Visual Basic*).

4. **Desarrollo de aplicaciones:** Una vez seleccionadas las herramientas y diseñado el monitoreo, se procede a construir la aplicación y crear los procedimientos automatizados de auditoría, realizándose las pruebas sistémicas correspondientes.
5. **Liberación del monitoreo:** Al concluir el desarrollo, se deben realizar las pruebas operativas necesarias a los procedimientos creados, con la finalidad de verificar la oportunidad y la operación, adecuarlas según sea el caso y capacitar al personal sobre la operación de la solución desarrollada.
6. **Puesta en producción:** Finalmente, los procedimientos se ponen a disposición de las áreas operativas y de los auditores para que los utilicen en el desarrollo continuo de sus revisiones.

1.4 Soluciones existentes

Software para Auditoría Interna y Auditoría de Sistemas ACL

ACL es una solución de software reconocida por la comunidad de auditores como el preferido para la extracción y análisis de datos, detección de fraudes y monitoreo continuo. Es potente, fácil de usar y permite convertir datos en información significativa para apoyar el alcance de los objetivos de negocios. (5)

A continuación se mencionan las características principales:

- Cuenta con un análisis interactivo para obtener resultados inmediatos.
- Fácil acceso de datos en distintos ambientes y sistemas, brindándose al auditor independencia de las funciones de procesamiento de datos y reducción del tiempo en que tiene disponible la información.
- Rapidez y facilidad de uso, lo que permite el análisis de grandes volúmenes de información cubriendo el 100% de los datos.
- Funciones propias de auditoría listas para su uso como estratificación, identificación de duplicados, faltantes, muestreo estadístico, comparaciones y cálculos.
- Automatización de tareas repetitivas, resultados gráficos y en reportes, protección de los datos originales y procesos de auto documentación.
- Es aplicable en auditorías financieras, de operaciones o de sistemas, análisis de ventas, control de calidad y revisiones de nóminas.

Ventajas de ACL:

- Menor costo de adquisición, mantenimiento y entrenamiento.
- Incremento de la calidad y productividad.
- Integridad y confiabilidad de la información para la toma de decisiones.
- Investigaciones más detalladas ya que se identifican fácilmente las discrepancias entre los archivos.

Software para análisis de datos IDEA

IDEA (en inglés: *Interactive Data Extraction and Analysis*) es un sistema informático canadiense de análisis, extracción y auditoría de datos para apoyar el trabajo de auditores, contadores, investigadores de fraude y profesionales en sistemas. Permite importar archivos desde diferentes formatos y reportes, leer, visualizar, analizar, procesar, obtener muestras y extraer datos. Provee cinco métodos de muestreo estadístico y funcionalidades para identificar vacíos en secuencia de registros, chequear registros duplicados, comparar, unir y agregar archivos, así como crear reportes personalizados. Es aplicable a investigaciones sobre fraudes y lavado de activos. (6)

Otras de sus características es que cuenta con dos plataformas, *IDEA* Básico e *IDEA Server*. Cuenta con un gran número de ayudas en línea en *HTML*. A la versión básica se le pueden adicionar componentes como:

- *Smart Analyzer* para pruebas básicas de auditoría financiera y reportes que pueden ser ejecutados por cualquier auditor con un mínimo esfuerzo.
- *Examiner* para explorar la seguridad y analizar logs generados en redes *Windows*.
- *UNIX Security Auditor* para revisar logos de seguridad en sistemas operacionales *UNIX*.
- *Symsure* para identificar los niveles de riesgos y de controles mediante el examen de transacciones y archivos de datos.

Requerimientos*:*

- *Windows 2000 SP4* o superior, *Windows XP SP1* o superior.
- 1 GB en RAM.
- Espacio en disco: para el programa 100 MB, para datos 1.5 veces el tamaño del archivo a analizar.

IBM Systems Solutions for Branch Banking (en español: Soluciones de Sistemas IBM para Sucursales Bancarias)

Plataforma desarrollada por *IBM* que posibilita a los clientes del sector bancario y servicios financieros, centralizar la operatoria y reducir los costos de operación. Es segura, estable, configurada y probada, escalable y fácil de mantener, propiciando un ambiente ideal para el crecimiento de los bancos. Las nuevas soluciones bancarias construidas sobre las potentes plataformas *xSeries* y *BladeCenter* de *IBM*, basadas en el procesador *Xeon* de *Intel*, incluyen características de software, redes y seguridad para que los administradores puedan monitorear las operaciones e implementar software en las sucursales desde una ubicación central. (9)

VERSAT-Sarasola

VERSAT-Sarasola es un sistema de gestión contable-financiero integrado cubano, constituido por diez módulos o subsistemas que permiten obtener información sin mayores complejidades de uso. Expresa sus resultados en una gama de funciones tipo y predeterminadas. Interactúa con *Microsoft Excel*, utilizando sus facilidades para el diseño de información. (7)

Principales características del VERSAT:

- Herramienta para la planificación económica, el control y el análisis de gestión.
- Diseñado para su empleo en cualquier tipo de entidad empresarial o presupuestada.
- Permite llevar el control y registro contable individual de todos los hechos económicos que se originan en las estructuras internas de las entidades, así como exponer el estado financiero y toda la información económica y contable en este universo.
- Se estructura en un grupo de subsistemas en los cuales se procesan y contabilizan los documentos primarios, donde se anotan los movimientos, los recursos materiales, laborales y financieros que se utilizan en una entidad.
- Logra establecer un proceso de interacción usuario-sistema.
- Rapidez y fiabilidad, a partir de la configuración del proceso de contabilización de los documentos primarios y de las propias posibilidades de trabajo contenidas en cada subsistema.

Solución integral Rodas XXI

Rodas XXI es un sistema integral económico administrativo cubano, desarrollado por la empresa CITMATEL, que posibilita automatizar el funcionamiento de cualquier empresa o unidad presupuestada nacional. En la actualidad es empleado por más de 450 entidades del país. Es un sistema en constante desarrollo que tiene muy en cuenta la opinión del usuario y ofrece cada vez, mejores soluciones para ofrecer un trabajo más viable y rápido. (8)

Principales posibilidades que ofrece Rodas XXI:

- Maneja un número ilimitado de empresas.
- Sus módulos se pueden configurar según las características del usuario.
- Permite el intercambio de los comprobantes generados por cada módulo con el de Contabilidad.
- Trabaja con doble moneda.
- Crea reportes fácilmente.
- La información está protegida por claves.
- Lleva un registro de las operaciones relacionadas con el sistema para permitir auditar el mismo.

Todos los módulos cuentan con una ayuda en línea, un manual de usuario detallado, además de un CD tutorial, que ofrece la oportunidad de aprender a trabajar con un potente paquete económico. Apoyándose en las bondades que ofrece la multimedia, estos tutoriales le permitirán adentrarse dentro del sistema Rodas XXI, guiado por textos e ilustraciones explicativas.

Los sistemas descritos anteriormente son considerados soluciones informáticas de alta calidad, debido a la gama de funcionalidades que los componen y a la seguridad con que manejan la información. A pesar de ello ninguno de los sistemas estudiados se consideró factible a implantar en el BNC, ya que no se ajustan al negocio y la mayoría son privativos, lo cual dificulta su adquisición por el estado cubano. Por estas razones, para resolver el problema de la investigación se decidió incorporar un subsistema de monitoreo al sistema Quarxo. El estudio realizado a los sistemas permitió identificar aspectos relevantes a considerar en la construcción de la solución, tales como la generación de alertas, la visualización de interfaces y la selección de filtros para búsquedas de datos.

1.5 Metodología de desarrollo de software

Una metodología es una colección de procedimientos, técnicas, herramientas y documentos auxiliares que ayudan a los desarrolladores de software en sus esfuerzos por implementar nuevos sistemas de información. Están formadas por fases, cada una de las cuales se puede dividir en sub-fases, que guiarán a los desarrolladores de sistemas a elegir las técnicas más apropiadas en cada momento del proyecto y también a planificarlo, gestionarlo, controlarlo y evaluarlo. (10)

En la actualidad y a partir de la concepción de diferentes ideologías sobre cómo desarrollar un software, han surgido un gran cúmulo de metodologías de desarrollo que atienden principalmente a

aspectos importantes tales como el tamaño del proyecto, la criticidad que presenta, la calificación del personal disponible, así como la cultura que presenta el mismo.

Comúnmente se dividen en dos grupos principales: las metodologías ágiles y las robustas. Las ágiles buscan minimizar los riesgos para desarrollar software en cortos lapsos de tiempo y resaltan las comunicaciones cara a cara en vez de la documentación, además acentúan que la primera medida de progreso es el software funcional, entre ellas están: *eXtreme Programming* (*XP*), *Scrum*, *Crystal*, *Open Unified Process* (*OpenUP*), *Agile Unified Proccess* (*AUP*). Por su parte las metodologías robustas destacan por ser apropiadas para productos y proyectos grandes en las que existe un rigor de requisitos y diseño adecuado para los procesos de prueba, como ejemplo de estas están: *RUP*, *Iconix y Microsoft Solution Framework* (*MSF*).

1.5.1 Modelo de Desarrollo de Software para el CEIGE

Por exigencias del centro CEIGE, para el desarrollo de la investigación fue preciso utilizar su propio modelo de desarrollo, denominado Modelo de Desarrollo de Software para el CEIGE. Se define como un proceso orientado a componentes, o sea, un conjunto de actividades para transformar los requisitos de un sistema. No es una guía con pasos firmemente establecidos, sino un conjunto de metodologías adaptables al contexto y necesidades de cada organización. Dicho modelo se basa en el nivel dos de *CMMI* (en inglés: *Capability Maturity Model Integration,* en español: Integración de Modelos de Madurez de Capacidades) establecido en la UCI.

Destaca por su flexibilidad y alineación al proyecto. Genera una serie de artefactos que son los productos tangibles del proceso, como por ejemplo Descripción de proceso de negocio, Modelo conceptual, Descripción de requisitos, Diagramas de clases del diseño y Diseño de caso de prueba. También cuenta con roles como analista, desarrollador y administrador de la calidad, siendo los papeles que desempeña una persona en un determinado momento. El Modelo de Desarrollo de Software para el CEIGE consta de dos fases principales, la Fase Inicio y la Fase Desarrollo, divididas en siete disciplinas: Estudio preliminar, Modelado del negocio, Requisitos, Análisis y diseño, Implementación, Pruebas internas y por último Pruebas de liberación. (10)

Ha sido probado anteriormente en el proyecto *Quarxo*.

1.6 Lenguaje y herramienta de modelado

Uno de los lenguajes definidos en el centro para la modelación de los artefactos es *UML* (en inglés: *Unified Modeling Language*), sistema de notación estándar en el mundo del desarrollo de sistemas. Está conformado por un lenguaje de modelado visual que se emplea para especificar, visualizar, construir y documentar artefactos de un sistema de software. Pretende unificar la experiencia pasada sobre técnicas de modelado e incorporar las mejores prácticas actuales en un acercamiento estándar. Incluye conceptos semánticos, notación y principios generales. Tiene partes estáticas, dinámicas, de entorno y organizativas. (11)

Para representar los procesos de negocio se utilizó la notación gráfica *Business Process Management Notation (BPMN)*, que describe la lógica de los pasos de un proceso de Negocio. Esta notación ha sido especialmente diseñada para coordinar la secuencia de los procesos y los mensajes que fluyen entre los participantes de las diferentes actividades. Proporciona un lenguaje común para que las partes involucradas puedan comunicar los procesos de forma clara, completa y eficiente. De esta forma *BPMN* define la notación y semántica de un Diagrama de Procesos de Negocio (BPD, del inglés *Business Process Diagram*). El modelado en *BPMN* se realiza mediante diagramas muy simples con un conjunto muy pequeño de elementos gráficos. Con esto se busca que para los usuarios del negocio y los desarrolladores técnicos sea fácil entender el flujo y el proceso.

Existen diversos medios informáticos para el modelado, a los que se les denomina herramientas *CASE*. Dichas herramientas son la aplicación de tecnologías informáticas a las actividades, técnicas y metodologías propias de desarrollo. Tienen como objetivo acelerar el proceso para el que han sido diseñadas, automatizar o apoyar una o más fases del ciclo de vida del desarrollo de software. Como principales ventajas en el uso de estas herramientas tenemos la mejora de la calidad de los desarrollos realizados y el aumento de la productividad.

La herramienta propuesta para el modelado de los procesos en esta investigación es el *Visual Paradigm* (*VP*) en su versión 8.0, debido a su eficacia en la visualización y diseño de elementos de software.

VP está orientado a la creación de diseños, el cual esgrime al paradigma de programación orientado a objetos. Además ayuda a una rápida construcción de mejores aplicaciones, mayor calidad y a un menor coste. Permite modelar diagramas de clases, código inverso y generar documentación. Provee soporte para la generación de código, tiene integración con diversos *IDE* como *NetBeans* (de *Sun*

Microsystems), *JDeveloper* (de *Oracle*), *Eclipse* (de *IBM*), *JBuilder* (de *Borland*), así como la posibilidad de realizarse la ingeniería inversa para aplicaciones realizadas en *Java*, .*NET*, *XML* e *Hibernate*. (12)

1.7 Arquitectura de software

Uno de los elementos más importantes en el proceso de desarrollo de software es sin duda la definición de la arquitectura del software. Esta constituye el diseño de mayor nivel en la organización de una aplicación. La arquitectura del software aporta una visión abstracta de alto nivel, pues posterga el detalle de cada uno de los módulos definidos a pasos posteriores del diseño. Entre sus tareas se destacan:

- Definir los módulos principales.
- Definir las responsabilidades que tendrá cada módulo.
- Definir la interacción que existirá entre los módulos.
- Control y flujo de datos.
- Secuenciación de la información.
- Protocolos de interacción y comunicación.
- Ubicación en el hardware.

La *IEEE* (en inglés: *Institute of Electrical and Electronics Engineers*) define la arquitectura de software como: “La Arquitectura del Software es la organización fundamental de un sistema formada por sus componentes, las relaciones entre ellos y el contexto en el que se implantarán y los principios que orientan su diseño y evolución”. (13)

1.8 Patrones

Los patrones son soluciones concretas que se utilizan en situaciones frecuentes, para favorecer la reutilización de código y resolver determinados problemas durante el desarrollo de software. Son esqueletos que los desarrolladores adaptan a las necesidades particulares de sus aplicaciones. (14)

Los patrones arquitectónicos reflejan la estructura global de un sistema. Especifican un conjunto predefinido de subsistemas y una serie de recomendaciones para organizar los distintos componentes que posee.

Los patrones de diseño se centran en aspectos más específicos y tienen un nivel de abstracción menor; los mismos están más próximos a la implementación final. Constituyen la base para la búsqueda de soluciones a problemas comunes en el desarrollo de software y otros ámbitos referentes al diseño de interacción o interfaces.

Patrón arquitectónico MVC

MVC (en español: Modelo-Vista-Controlador) es un patrón que divide la aplicación en tres partes esenciales:

- **Modelo:** Encapsula el núcleo funcional y los datos involucrados. Representa la funcionalidad y los datos esenciales, y es independiente de la representación en las interfaces. Contiene también los accesos a la base de datos, desvinculando a la vista y al controlador del acceso a la base de datos.
- **Vista:** Presentación de la información por pantalla. Obtiene datos del modelo y los despliega para el usuario. Cada vista tiene asociado un controlador. El controlador recibe eventos como entradas (movimientos del mouse, activación de botones) y los traduce a solicitudes de servicio.
- **Controlador:** Manejan el flujo del sistema. Define la forma en la que debe reaccionar la interfaz del usuario, frente a la entrada de datos. Es el encargado de escuchar los cambios en la vista y enviarlos al modelo, remitiendo los datos a la vista como un ciclo. (14)

Patrones de diseño GRASP

Los Patrones Generales para Asignar Responsabilidades: *GRASP* (en inglés: *General Responsibility Asignment Software Patterns*), describen los principios fundamentales de la asignación de responsabilidades a objetos, expresados en forma de patrones. Su nombre se debe a la importancia de captar estos principios, para diseñar eficazmente el software orientado a objetos. (11)

GRASP está compuesto por los siguientes:

- **Patrón Experto:** Este patrón consiste en asignar una responsabilidad al experto en información, es decir, a la clase que cuenta con la información necesaria para cumplir con la responsabilidad. Se refuerza el encapsulamiento y se favorece el bajo acoplamiento.
- **Patrón Creador:** Este patrón guía la asignación de responsabilidades relacionadas con la creación de objetos, tarea muy frecuente en los sistemas orientados a objetos. Su propósito principal es encontrar un creador que se debe conectar con el objeto producido en cualquier evento. Brinda soporte al bajo acoplamiento.
- **Patrón Bajo Acoplamiento:** Pretende asignar una responsabilidad para mantener el bajo acoplamiento, es decir, el diseño de clases más independientes, que no se relacionen con muchas otras, que reduzcan el impacto de los cambios, que sean más reutilizables y acrecienten la oportunidad de una mayor productividad.

- **Patrón Alta Cohesión:** Su objetivo es asignar una responsabilidad, de modo que la cohesión siga siendo alta. Las clases con alta cohesión se caracterizan por tener responsabilidades estrechamente relacionadas y no realizar un trabajo enorme. Una clase con alta cohesión es útil porque es bastante fácil darle mantenimiento, entenderla y reutilizarla. Es un patrón evaluativo que el desarrollador aplica al valorar sus decisiones de diseño.
- **Patrón Controlador:** Consiste en asignar la responsabilidad a una clase para manejar los mensajes correspondientes a eventos de un sistema. El controlador es un intermediario entre la interfaz de usuario y el núcleo de las clases donde reside la lógica de la aplicación. Este patrón sugiere que la lógica de negocios debe estar separada de la capa de presentación para aumentar la reutilización del código y a la vez tener un mayor control.

Patrones de diseño GOF

Por sus siglas en inglés *Gang of Four*. Llamados así debido a que fueron cuatro autores los que escribieron el libro "*Design Patterns*" que ilustra sus funciones. (11)

A continuación se describen:

- **Patrón Fachada:** Patrón estructural que trata de simplificar la interfaz entre dos sistemas o componentes de software. Oculta un sistema complejo detrás de una clase que funciona como pantalla o fachada. La idea principal es la de ocultar todo lo posible la complejidad de un sistema, el conjunto de clases o componentes que lo definen, de forma que solo se ofrezca un punto de entrada al sistema cubierto por la fachada. Su uso aísla los posibles cambios que se puedan producir en alguna de las partes.
- **Patrón Mediador:** Patrón de comportamiento que coordina las relaciones entre sus asociados. Permite la interacción de varios objetos, sin generar acoples fuertes en esas relaciones.
- **Patrón Cadena de Responsabilidad:** La cadena de responsabilidad se encarga de evitar el acoplamiento del remitente de una petición a su receptor, dando a más de un objeto la posibilidad de manejar la petición.
- **Patrón *Singleton*:** Patrón creacional diseñado para restringir la creación de objetos pertenecientes a una clase. Su objetivo es garantizar que una determinada clase solo tenga una instancia y proporcionar un punto de acceso global a esta. Es un patrón muy sencillo de diseñar y a menudo es implementado por otros patrones.

Patrón DAO

Por sus siglas en inglés *Data Access Object*. Este patrón maneja la conexión con la fuente de datos para obtener y almacenar información. Su uso ofrece varios beneficios para la persistencia de datos como son la separación del acceso a datos de la lógica de negocio, lo cual suele ser altamente recomendable en sistemas medianos o grandes, o que manejen lógica de negocio compleja.

Permite el encapsulamiento de la fuente de datos, lo cual es especialmente beneficioso en sistemas con acceso a múltiples entradas. Posibilita el ocultamiento de la API con la que se accede a los datos, lo que viabiliza que se pueda cambiar el negocio del manejo de los datos persistentes sin que afecte el resto. Por ejemplo cambiar de *framework* de acceso a datos, centralizando todo el acceso a datos en una capa independiente. (15)

1.9 Ambiente de desarrollo

El ambiente de desarrollo es el conjunto de herramientas, aplicaciones, lenguajes y tecnologías vinculadas a la implementación y despliegue de un sistema informático.

Lenguajes de programación

Entre los lenguajes de programación del lado del servidor más usados en el desarrollo web se encuentran *PHP*, *C#* y *Java*. El primero es un lenguaje gratuito e independiente de plataforma, rápido, con una amplia biblioteca de funciones y mucha documentación. En cuanto a C#, es un lenguaje orientado a objetos, desarrollado y estandarizado por *Microsoft* como parte de la plataforma *.NET*. Por último contamos con el lenguaje *Java*, fácil de usar, multiplataforma, sumamente flexible, permite la creación de programas modulares y de códigos reutilizables. Dicho lenguaje además es compilado, utiliza el paradigma orientado a objeto y permite el desarrollo de programas seguros y de alto rendimiento. (16)

Por las características antes expuestas, se optará por *Java* como lenguaje de programación del lado del servidor, bajo la plataforma *JEE* (en inglés: *Java Enterprise Edition*). Es una tecnología que apunta a facilitar el diseño y desarrollo de aplicaciones empresariales robustas, escalables y seguras. Además simplifica las aplicaciones empresariales basándolas en componentes modulares y estandarizados. Ofrece un conjunto de servicios a dichos componentes y maneja muchas de las funciones de la aplicación de forma automática, sin necesidad de una programación compleja.

En cuanto a los lenguajes de programación del lado del cliente se encuentra *JavaScript*, el cual es compatible con la mayoría de los navegadores modernos. Permite crear efectos especiales en las páginas web y definir interactividades con el usuario. El navegador del cliente es el encargado de

interpretar las instrucciones *JavaScript* y ejecutarlas. Es un lenguaje bastante sencillo, rápido y fácil de aprender por personas de poca experiencia.

Alternativamente existe *XHTML*, lenguaje de marcado pensado para sustituir el *HTML*. Permite generar documentos y contenidos de hipertexto generalmente publicados en la Web. Es además una reformulación del lenguaje *HTML* que se puede jactar de ser compatible con *XML*. (17)

1.9.1 Frameworks

Framework se traduce aproximadamente en “marco de trabajo” y consiste en una estructura de software que posee componentes personalizables e intercambiables para el desarrollo de una aplicación. Brinda comodidades al desarrollador, mecanizándose el trabajo común y cargándolo como una especie de plantilla o solución preestablecida. Se puede considerar como una aplicación genérica incompleta y configurable a la que se le puede agregar algunas piezas para construir una aplicación concreta. (18)

En otras palabras, un *framework* es una herramienta informática compuesta de varias funcionalidades generales, que brinda la facilidad para el desarrollo y organización de una aplicación con características específicas.

En el contexto actual de la industria de software se cuenta con un grupo considerable de *frameworks* que facilitan y agilizan el desarrollo de sistemas. Específicamente para los lenguajes seleccionados con anterioridad, existen un conjunto de ellos que se emplean con mucha frecuencia. Para lo referente al trabajo de acceso a datos, se pueden mencionar a algunos *frameworks* de elevado prestigio como *Athena Framework*, *Ebean ORM* e *Hibernate Framework*. Por otro lado están un conjunto de *frameworks* que hacen posible la aplicación del patrón *MVC y* poseen una elevada aceptación a nivel mundial, entre ellos: *Struts*, *JBoss Seam* y *Spring Framework.*

De igual manera para la lógica de presentación se utilizan *frameworks*, debido a la importancia que ella posee en los sistemas informáticos, teniéndose en cuenta su interacción con los usuarios. Diversas son las librerías desarrolladas para *JavaScript*, que enriquecen las interfaz de usuario y brindan un ambiente de trabajo más limpio, entre ellas destacan *Qooxdoo*, *Ext JS* y *Dojo Toolkit*.

Ateniéndose a las características del proyecto y a las prestaciones que brindan cada uno de estos *frameworks*, se decidió hacer uso de los que se caracterizan a continuación.

Spring framework 3.1

Spring está compuesto por un conjunto de módulos, de los cuales se pueden tomar aquellos que faciliten la implementación del trabajo en cuestión. Ideado principalmente por Rob Johnson, es libre y de código abierto desde febrero del 2003. El centro de *Spring* está basado en el principio de Inyección de Dependencias. Esta técnica hace externa la creación y el manejo de las dependencias de las clases, con lo que se logra una mayor limpieza y claridad en el código.

Es un *framework* que ha venido a revolucionar la manera de programar aplicaciones *Java* debido a la facilidad de crear componentes reutilizables. Se adapta fácilmente con otros *frameworks* y ofrece autonomías a los desarrolladores, además de soluciones bien documentadas. Permite desarrollar un software seguro y robusto, mediante el uso de las prácticas comunes en la industria de software. (19)

Spring contiene una gran variedad de características que le proporcionan una funcionalidad muy basta; dichas características están organizadas en siete módulos:

- ***Spring Core*****:** "Núcleo", es la parte fundamental del *framework* ya que provee toda la funcionalidad de Inyección de Dependencias permitiéndose administrar la funcionalidad del contenedor de *beans*.
- ***Spring Context*****:** "Contexto", provee de herramientas para acceder a los *beans* de una manera elegante. El paquete de contexto hereda sus características del paquete de *beans* y añade soporte para mensajería de texto.
- ***Spring DAO*****:** Provee una capa de abstracción de *JDBC* que elimina la necesidad de teclear código tedioso y redundante. Permite que la programación del acceso a datos sea un poco más limpia y entendible. Permite administrar transacciones tanto declarativas como programáticas.
- ***Spring ORM*****:** Provee capas de integración para API de mapeo objeto - relacional, incluyéndose *Hibernate* e *iBatis*. El mapeo de *ORM* se puede usar en conjunto con otras características de *Spring*, como la administración de transacciones mencionada con anterioridad.
- ***Spring AOP*****:** Provee una implementación de programación orientada a aspectos. Sus funcionalidades permiten desacoplar el código de una manera limpia para implementar funcionalidades que por lógica y claridad debería estar separada.
- ***Spring Web*****:** Provee características básicas de integración orientado a la web y se encarga de crear el contexto para las aplicaciones web. Incluye soporte para una variedad de tareas como la subida de archivos y la vinculación de parámetros de las peticiones a objetos del negocio.
- ***Spring Web MVC*****:** Provee una implementación Modelo-Vista-Controlador para las aplicaciones web. La implementación de *Spring MVC* permite una separación entre código de modelo de dominio y las formas web.

Hibernate framework 3.5

Hibernate pertenece a los denominados *framework ORM* (en español: Mapeo de Objetos Relacional). Es una herramienta que facilita el mapeo de atributos entre una base de datos relacional tradicional y el modelo de objetos de una aplicación mediante archivos declarativos *XML*, o anotaciones en los *beans* de las entidades que permiten establecer estas relaciones. Propone un lenguaje de consulta orientado a objeto *HQL*, el cual puede ejecutar cualquier tipo de consultas sin importar el gestor de base de datos. (20)

Características de *Hibernate framework*:

- Permite expresar consultas utilizando *SQL* nativo o consultas basadas en criterios.
- Soporta todos los sistemas gestores de bases de datos *SQL*, integrándose de manera elegante y sin restricciones con los más populares servidores de aplicaciones *JEE* y contenedores web, además de poder utilizarse en aplicaciones de escritorio.
- Proporciona persistencia de una manera transparente para el desarrollador.
- Soporta el paradigma de orientación a objetos de una manera natural: herencia, polimorfismo, composición y el *framework* de colecciones de *Java*.
- Soporte para modelos de objetos con una granularidad muy fina. Permite una gran variedad de mapeos para colecciones y objetos dependientes.
- Provee un sistema de caché de dos niveles y puede ser usado en un clúster. Permite inicialización *lazy* (en español: perezosa) de objetos y colecciones.
- Proporciona el lenguaje *HQL*, el cual provee una independencia del *SQL* de cada base de datos, tanto para el almacenamiento de objetos como para su recuperación.
- Presenta un potente mecanismo de transacciones de aplicación llegándole incluso a permitir las interacciones largas (aquellas que requieren la interacción con el usuario durante su ejecución).
- Soporta los diversos tipos de generación de identificadores que proporcionan los sistemas gestores de bases de datos, así como generación independiente de la base de datos, incluyéndose identificadores asignados por la aplicación o claves compuestas.
- Brinda una jerarquía detallada para tratar todas las excepciones que pueden ocurrir al fallar la interacción con la base de datos.

Dojo Toolkit 1.3

Dojo es un *framework* de *JavaScript* que brinda varios instrumentos como controles de la interfaz de usuario, efectos, utilidades comunes y una API para gestionar el acceso asíncrono al servidor. Consta

de una colección de scripts estáticos para posibilitar el desarrollo de aplicaciones web, enriquecidas en el cliente e incorpora soporte para el trabajo con la tecnología *AJAX*.

Se destaca por permitir desarrollar aplicaciones que pueden funcionar con independencia del navegador web donde se ejecuten. Hace transparente el desarrollo para diferentes implementaciones del *DOM* y ofrece soporte para la internacionalización, e incluye un gran cúmulo de componentes visuales basados en *XHTML*, *CSS* y *JavaScript* para enriquecer la interfaz de usuario. Presenta una arquitectura modular donde destacan los módulos: *dojo*, *dijit* y *dojox*. (21)

1.9.2 Herramientas de desarrollo

Eclipse IDE Galileo 3.5

Eclipse es un *IDE* de código abierto y multiplataforma desarrollado por *IBM*. Emplea plugins para proporcionar toda su funcionalidad, a diferencia de otros entornos monolíticos donde las funcionalidades están todas incluidas, las necesite el usuario o no. La arquitectura de plugins permite integrar diversos lenguajes, introducir otras aplicaciones accesorias que pueden resultar útiles durante el proceso de desarrollo, tales como: herramientas *UML*, editores visuales de interfaces y ayudas en línea para librerías. (22)

Servidor de aplicaciones web Tomcat 6.0

Apache Tomcat es un software de código abierto implementado para las tecnologías *Java Servlet* y *Java Server Pages*. Es desarrollado en un entorno abierto, participativo y publicado bajo la licencia del software de *Apache*. Se ejecuta en varios sistemas operativos. Es un servidor configurable de diseño modular, con diversidad que permite garantizar una elevada seguridad y buenas prestaciones. Además brinda soporte para la plataforma de desarrollo *JEE*. (23)

Existe abundante documentación asociada al mismo, cuenta con una gran comunidad de desarrollo y es uno de los más usados internacionalmente.

Control de versiones (SubVersion) 1.6.6

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. (24)

Servidor de base de datos SQL Server 2005

SQL Server 2005 es un servidor de base de datos basados en el modelo relacional. Se caracteriza por brindar soporte para transacciones y procedimientos almacenados. Incluye un potente entorno gráfico para administración, permite además trabajar en modo cliente-servidor y promueve la escalabilidad y seguridad de la información.

Sus lenguajes de consulta son el *SQL* y el *T-SQL* (en inglés *Transact-SQL*), siendo este último el principal medio de programación y administración del servidor. Requiere para su funcionamiento del sistema operativo *Windows*, lo que representa un inconveniente para su utilización. (25)

Su uso se basa en una decisión del BNC.

1.10 Conclusiones parciales

Las tecnologías adoptadas para el desarrollo de la solución están acordes a los requerimientos del cliente, a las políticas del centro y a los estándares internacionales. Se establecieron de la siguiente forma: Modelo de Desarrollo de Software para el CEIGE como soporte en el desarrollo de software, *UML* como lenguaje de modelado, *Visual Paradigm 8.0* como herramienta *CASE* para el modelado, Modelo Vista Controlador como patrón de arquitectura, *GRASP*, *GoF* y *DAO* como patrones de diseños, *Java* bajo la plataforma *JEE* como lenguaje de programación, *Spring 3.1* como *framework* núcleo de la aplicación apoyado por los *framework*s *Hibernate 3.5* y *Dojo Toolkit 1.3*, *Eclipse Galileo 3.5* como herramienta de desarrollo, *Apache Tomcat 6.0* como contenedor web, *SubVersion 1.6* para el control de versiones y *SQL Server 2005* como sistema gestor de base de datos. Los sistemas estudiados aportaron ideas para el desarrollo del subsistema Monitoreo de Operaciones en Cuentas de Clientes.

CAPÍTULO 2 PROPUESTA DE SOLUCIÓN

2.1 Introducción

A partir del Modelo de Desarrollo de Software para el CEIGE, en el presente capítulo se abordan las disciplinas Modelado del negocio, Requisitos, así como Análisis y diseño, dejándose para el siguiente capítulo el resto de las disciplinas en las que se obtendrá finalmente el desarrollo del subsistema.

Se hace hincapié en la disciplina Análisis y diseño, donde se obtienen un grupo de artefactos del modelo de desarrollo, que ayudarán a mitigar problemas en las posteriores disciplinas de Implementación y Pruebas.

2.2 Modelado del negocio

El BNC cuenta con una herramienta informática llamada *Quarxo*, que permite gestionar eficientemente todas las operaciones bancarias. Entre los subsistemas que lo componen no existe alguno para el monitoreo.

La operadora de monitoreo y control es la encargada de supervisar las operaciones efectuadas sobre todas las cuentas de clientes. Dicha tarea se realiza manualmente, complicándose la detección de errores o hechos delictivos.

Para enfocar de manera eficiente las necesidades del cliente y garantizar que el software desarrollado brinde solución a las mismas, se realizó un estudio para adquirir un mejor entendimiento del funcionamiento de los procesos de negocio en el área de Monitoreo. Se identificaron 4 procesos principales que se explicarán a continuación.

Monitoreo de expedientes

El proceso de monitoreo de expedientes es engorroso y lento, a causa de que los documentos no se encuentran digitalizados. La operadora de monitoreo y control debe verificar de forma manual la vigencia e integridad de los datos en los expedientes de las cuentas de clientes. Para ello debe consultar grandes volúmenes de información. Debe comprobar que estén actualizados los documentos “Conozca a su Cliente” y “Debida Diligencia”, los cuales tienen una validez de un año.

Monitoreo de cuentas inactivas

El proceso de monitoreo de cuentas inactivas es pobre y lánguido. Consiste en comprobar la legalidad de operaciones realizadas en cuentas de clientes, que no hayan sido utilizadas en un período de más de seis meses. La operadora de monitoreo y control no puede seguir el estado de actividad de las operaciones en todas las cuentas de clientes, haciéndose tardía la verificación de las mismas.

Monitoreo de proveedores

El proceso de monitoreo de proveedores es ineficiente. La operadora de monitoreo y control debe bajar de internet, un listado en formato *PDF* con las principales entidades y organizaciones vinculadas al terrorismo, lavado de dinero o drogas. Luego comparar manualmente que no existan coincidencias con las listas de proveedores del BNC. La lista es demasiado extensa, dificultándose una comprobación correcta.

Monitoreo de Transacciones

El proceso de monitoreo de transacciones resulta difícil. La operadora de monitoreo y control requiere los montos de transacción por operaciones de una cuenta de cliente y chequea que estén enmarcados dentro del acotado permisible del cliente correspondiente. Esta tarea se realiza de manera aleatoria, consumiéndose gran cantidad de tiempo y muy pocos resultados.

Al analizar los flujos de los procesos de monitoreo se pueden determinar elementos que podrían ser objetos de automatización. Es posible normalizar y digitalizar el registro de los expedientes. Es factible programar alertas que señalen operaciones efectuadas sobre cuentas de clientes inactivas. También se pueden generar alertas que indiquen si se ha sobrepasado el monto permisible en una transacción.

2.2.1 Descripción del proceso de negocio Monitoreo de expedientes.

Objetivo	Monitorear expediente de cuentas de clientes en BNC
Evento(s) que lo genera(n)	N/A
Pre condiciones	El expediente está creado
Marco legal	Resolución 76 del 2000
Clientes internos	N/A
Clientes externos	N/A
Entradas	Expediente

Flujo de Eventos	
Flujo Básico Monitorear Expediente	
1. Buscar Expediente:	La Operadora MC anualmente busca el expediente de una cuenta de cliente determinada en el archivo.
2. Revisar Firmas Autorizadas:	La Operadora MC chequea que todos los representantes con firmas autorizadas tengan su nombramiento en el expediente.
3. Revisar Vigencia de los Modelos Debida Diligencia:	La Operadora MC revisa la vigencia de los modelos de la Debida Diligencia, comparando las fechas iniciales de los modelos con la actual, que no debe exceder de un año.
4. Revisar Vigencia del Modelo Conozca a su Cliente:	La Operadora MC revisa la vigencia del modelo Conozca a su cliente, comparando las fechas iniciales del modelo con la actual, que no debe exceder de un año.
Pos-condiciones	
1.	El expediente queda revisado.
Salidas	
Expediente (FD).	
Flujos Alternos	
2. a Firmas Autorizadas con Problemas.	
1.3.1 Informar Problemas en las Firmas Autorizadas:	El Operador MC le informa a la Presidencia el problema en el expediente.
Pos-condiciones	
Salidas	
Flujos Alternos	
3. a Debida Diligencia Desactualizada.	
1. Informar Problemas en los Modelos Debida Diligencia:	El Operador MC le informa a la Presidencia el problema en el expediente.
Pos-condiciones	
Salidas	
Flujos Alternos	
4. a Conozca a su Cliente Desactualizado.	
1. Informar Problemas en el Modelo de Conozca a su Cliente:	El Operador MC le informa a la Presidencia el problema en el expediente.
Asuntos Pendientes	
Posibles mejoras al proceso.	

- Se digitalizará parte de la información del expediente, lo cual hará más ágil el trabajo de monitoreo del mismo.
- La alta gerencia será notificada a través del sistema de los expedientes con problemas.

Requisitos funcionales candidatos:

- Registrar expediente de cuenta de cliente.
- Actualizar expediente de cuenta de cliente.
- Cancelar expediente de cuenta de cliente.
- Consultar expediente de cuenta de cliente.
- Revisar expediente de cuenta de cliente.
- Obtener un reporte de los expedientes con problemas.

Requisitos no funcionales candidatos:

Sobre el proceso:

- En la solución que presenta Quarxo en su primera fase se creó el subsistema Cuentas de Clientes, en el mismo se aprecia similitud con el expediente de una cuenta de cliente y por ello se pretende que en la segunda fase se realicen los cambios pertinentes para que pueda usarse para gestionar el expediente de forma digital. Uno de los campos del expediente que debe agregarse a la vista es el objeto de la cuenta.

Tabla 1: Descripción de proceso del negocio monitoreo de expediente

2.3 Requisitos

Los requisitos describen necesidades y aspiraciones que nacen de un producto determinado. Un requisito especifica el comportamiento que deberá tener un sistema, para reducir la posibilidad de errores y puntualizar de forma clara lo que el cliente desea. Muestran las cualidades, características, funciones y atributos que debe cumplir el sistema que se está desarrollando. Existen dos tipos de requisitos, los funcionales y los no funcionales. (26)

Técnicas para la captura de requisitos

Una adecuada compresión de los requisitos contribuye al desarrollo de mejores sistemas que cumplan con las necesidades y expectativas del cliente. Para llevar a cabo este procedimiento existen diversas técnicas que guían al analista en el proceso de establecer una comunicación con el cliente y el equipo de desarrollo, de las cuales fueron utilizadas las siguientes:

- **Entrevistas:** Se realizaron mediante reuniones y encuentros efectuados con los clientes, para obtener la información necesaria de los procesos relacionados con el monitoreo. A través de estas entrevistas se alcanzó como resultado la identificación de los requisitos funcionales necesarios para la satisfacción del cliente.

- **Tormenta de ideas:** Para su puesta en práctica fue necesario realizar una reunión en la cual los clientes y el equipo de desarrollo brindaran sus ideas en cuanto a la solución propuesta.

2.3.1 Requisitos funcionales (RF)

Con el objetivo de describir toda la interacción del usuario con el subsistema, teniéndose en cuenta las necesidades actuales del proceso de monitoreo, se definen dos módulos principales con varios requisitos funcionales, descritos a continuación.

Módulo Expediente: Tiene como objetivo principal gestionar la información de los expedientes en cuentas de cliente. El mismo contendrá las siguientes funcionalidades:

RF 1: **Registrar expediente**	El sistema permite registrar los datos insertados por el usuario cuando crea un Expediente de cuentas de cliente.
RF 2: **Buscar Expediente**	El sistema permite buscar un Expediente de cuentas de cliente creado anteriormente, de acuerdo a los criterios de búsqueda seleccionados por el usuario.
RF 3: **Consultar Expediente**	El sistema permite consultar un Expediente de cuentas de cliente creado anteriormente.
RF 4: **Imprimir Expediente**	El sistema permite imprimir los datos de un Expediente de cuentas de cliente consultado.

Tabla 2: Descripción de requisitos funcionales del módulo Expediente.

Módulo Alerta: Tiene como objetivo principal mostrar y gestionar alertas del sistema registradas por defecto y según parámetros definidos por el usuario. El mismo contendrá las siguientes funcionalidades:

RF 5: **Alerta Vencimiento de Expediente**	El sistema debe emitir una alerta cuando expire la validez de un Expediente de cuentas de cliente (Al transcurrir un año de ser registrado).
RF 6: **Alerta Uso de Cuenta Inactiva**	El sistema debe emitir una alerta cuando se realice una operación sobre una cuenta de cliente en estado inactiva (cuenta sin uso por más de seis meses).
RF 7: **Generar Alerta**	El sistema debe permitir generar una alerta mediante la

	configuración de parámetros como el valor máximo de créditos por transacción.
RF 8: **Buscar Alerta**	El sistema permite buscar una alerta generada anteriormente, de acuerdo a los criterios de búsqueda seleccionados por el usuario.
RF 9: **Cambiar Estado de Alerta**	El sistema debe permitir cambiar el estado de una alerta (Estados: Pendiente y Revisada).
RF 10: **Obtener Reporte de Alertas**	El sistema debe permitir obtener un reporte de todas las alertas con estado Pendiente.

Tabla 3: Descripción de requisitos funcionales del módulo Alerta.

2.3.2 Requisitos no funcionales (NRF)

Para detallar las cualidades o propiedades con las que el sistema debe contar, se delimitaron una serie de requisitos no funcionales con el objetivo de brindar al producto, usabilidad, adaptabilidad, madurez y atractivo.

Tipo de requisito	PCs clientes	PCs servidores	
		Servidor 1	**Servidor 2**
Software	Máquina virtual de *Java* 6.0u20 o superior. *Morzilla Firefox* 3.6.	Windows Server 2003. Apache Tomcat 6.0 o superior. Máquina virtual de Java 6.0u20 o superior.	Windows Server 2003. Microsoft SQL Server 2005 o superior.
Hardware	Procesador Pentium IV o superior 2.0 GHZ o superior. RAM: 256 MB (recomendado 512). Una tarjeta de red.	Procesador: Core 2 Duo 2.0 GHZ o superior. RAM: 4 GB. Disco duro: 160 GB. UPS: 1. Lector de CD: 1.	

Tabla 4: Descripción de requisitos no funcionales del sistema Quarxo

Usabilidad.

RNF 1: El sistema permitirá visualizar una descripción textual en los mensajes de error.

RNF 2: El sistema permitirá ordenar las etiquetas de las funcionalidades en el menú según las dependencias de ejecución del negocio.

RNF 3: Las etiquetas de cada funcionalidad y los campos de cada interfaz tendrán títulos asociados a su función de negocio.
RNF 4: El sistema puede ser empleado por personas que no cuenten con vastos conocimientos de informática.

Adaptabilidad.
RNF 5: El sistema podrá ser operado desde el navegador web *Mozilla Firefox* 3.6 en adelante.

Confidencialidad.
RNF 6: La información manejada por el sistema debe estar protegida ante el acceso no autorizado y la divulgación.

Disponibilidad.
RNF 7: El sistema debe estar disponible todo el tiempo para los usuarios autorizados.

Restricciones de diseño.
RNF 8: El producto de software final debe diseñarse sobre una arquitectura modelo - vista - controlador.

Madurez.
RNF 13: El sistema no permitirá la entrada de datos incorrectos.
RNF 14: El sistema impondrá campos obligatorios para garantizar la integridad de la información que se introduce por el usuario.

Atracción.
RNF 15: El sistema permitirá, mediante el uso de íconos, diferenciar los mensajes de información, error y de advertencia.
RNF 16: El sistema establecerá una tipografía y colores estándares en toda la aplicación.

Seguridad.
RNF 17: El sistema requiere para su uso de un usuario y contraseña, para permitir el acceso de cada usuario a las funcionalidades que se les definieron a partir de su área de trabajo.

2.3.3 Validación de requisitos

La validación de requisitos tiene como objetivo, definir que los requerimientos especificados realmente detallan el sistema que el usuario necesita o desea. Verifica que las especificaciones de requisitos no

presentan omisiones, conflictos o ambigüedades, además de que sean correctas las interpretaciones por parte del equipo de desarrollo de software.

Entre las técnicas utilizadas para validar los requisitos identificados se utilizaron las siguientes:

- **Revisiones**: Esta técnica se utiliza para corregir cualquier error existente en la documentación o modelado de los requisitos, con el objetivo de encontrar conflictos en el producto y poder trazar alternativas de solución.
- **Prototipos**: Se utilizan como modelo a escala de la solución final, para brindarle al cliente una visión más clara de cómo quedaría el producto que va a recibir. Mediante los prototipos se puede verificar si las especificaciones han sido construidas de acuerdo a los requisitos del sistema. Ofrecen como resultado un modelo para el desarrollo del producto, que atiende a las necesidades específicas del cliente.

2.4 Arquitectura propuesta para el sistema

La arquitectura es el resultado de una agrupación de elementos arquitectónicos, con una estructura apropiada que admita satisfacer las funcionalidades y requerimientos necesarios para el buen desempeño de un sistema. En esta materia el sistema *Quarxo* cuenta con una arquitectura en capas, compuesta por una capa de presentación, una capa de negocio, una capa de acceso a datos, así como una capa de dominio transversal a las demás, portadora de los objetos del dominio. Esta arquitectura presenta un orden jerárquico que minimiza la dependencia entre componentes, donde cada capa proporciona servicios a la superior y es servida por la inferior.

A continuación se describen cada una de las capas según su aplicación en el sistema.

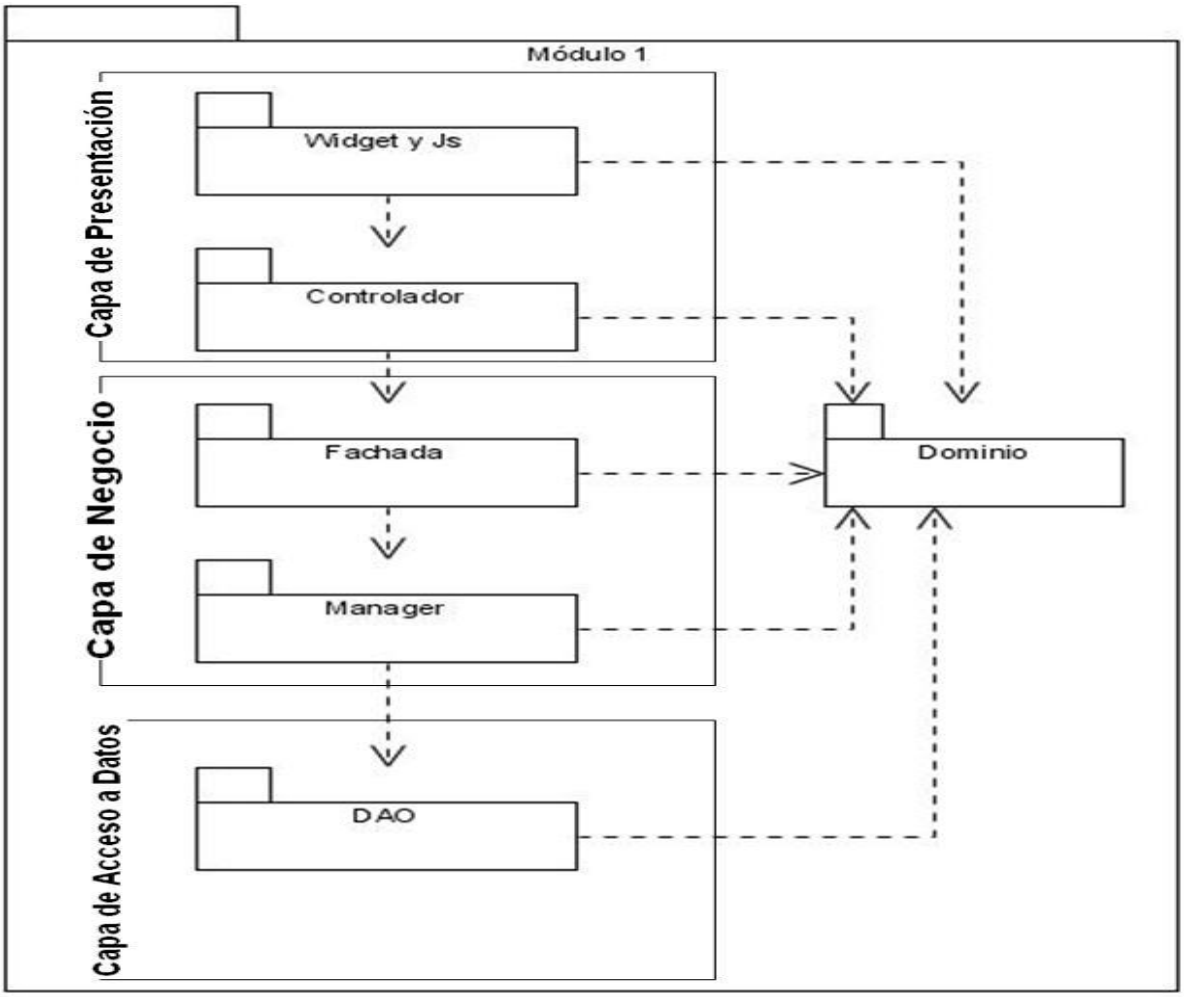


Ilustración 1: Estructura de las capas lógicas del sistema Quarxo

Capa de presentación

Para tratar la capa de presentación se utiliza *Spring MVC*, encargado de recibir, controlar y enviar respuestas a las peticiones realizadas desde el cliente, bajo las adaptaciones realizadas al *framework* por el proyecto *SIGEP* (en español: Sistema de Gestión Penitenciaria). Se emplea asimismo, *Spring Web Flow* para representar y controlar los flujos complejos reutilizables de la aplicación. Además se asume la librería *Java Script Dojo* para generar las interfaces que interactúan con el usuario, contiguo a *Spring Security*. La Capa de presentación está relacionada con la capa de negocios y la capa de dominio.

Capa de negocio

En la capa de negocio se manipulan algunos módulos del *framework Spring* para declarar y representar las relaciones de dependencia de cada una de las clases, tales como *Spring Transaction* y *Spring AOP* (en inglés: *Aspect Oriented Programming*), encargados de ejecutar las transacciones y la auditoría de cada uno de los métodos del negocio. Se emplea también *Spring Security* para asegurar las invocaciones de los métodos. Ésta capa está divida en dos subcapas: *Facade* y *Manager*. *Facade* se encarga del intercambio entre la capa de presentación y la capa de negocio, sin poseer lógica de negocio. *Manager* por su parte contiene la jerarquía de clases suficientes para implementar el negocio de la aplicación y manejar la capa de acceso a datos para adquirir los datos persistidos, así como la capa de dominio para generar las entidades del negocio.

Capa de acceso a datos

La capa de acceso a datos es la encargada de gestionar la persistencia de los datos. Con este propósito en la aplicación se recurre al *framework Hibernate*, para ejecutar las operaciones básicas sobre la base de datos y *Spring JDBC* para interactuar con los procedimientos almacenados. En el tratamiento de esta capa se esgrime el patrón *DAO* (en inglés: *Data Acces Object*, en español: Objetos de Acceso a Datos). Las interfaces de los *DAOs* contienen fundamentalmente las operaciones de inserción, modificación, eliminación y localización de un objeto a partir de la llave primaria.

2.4.1 Estructura de capas en el Sistema Quarxo

Dicha arquitectura está basada en la complejidad de los procesos y la relación entre ellos, por lo que se organizó de forma jerárquica por subsistemas, módulos y componentes; quedándose estructurado de la siguiente forma:

- **Subsistema:** Agrupa un conjunto de Módulos que estén relacionados con los procesos que ejecutan.
- **Módulo:** Agrupa un conjunto de requisitos relacionados con uno o más procesos bancarios que estén estrechamente relacionados.
- **Componentes:** Conjunto de funcionalidades comunes que son reutilizados por el resto de los módulos del sistema.

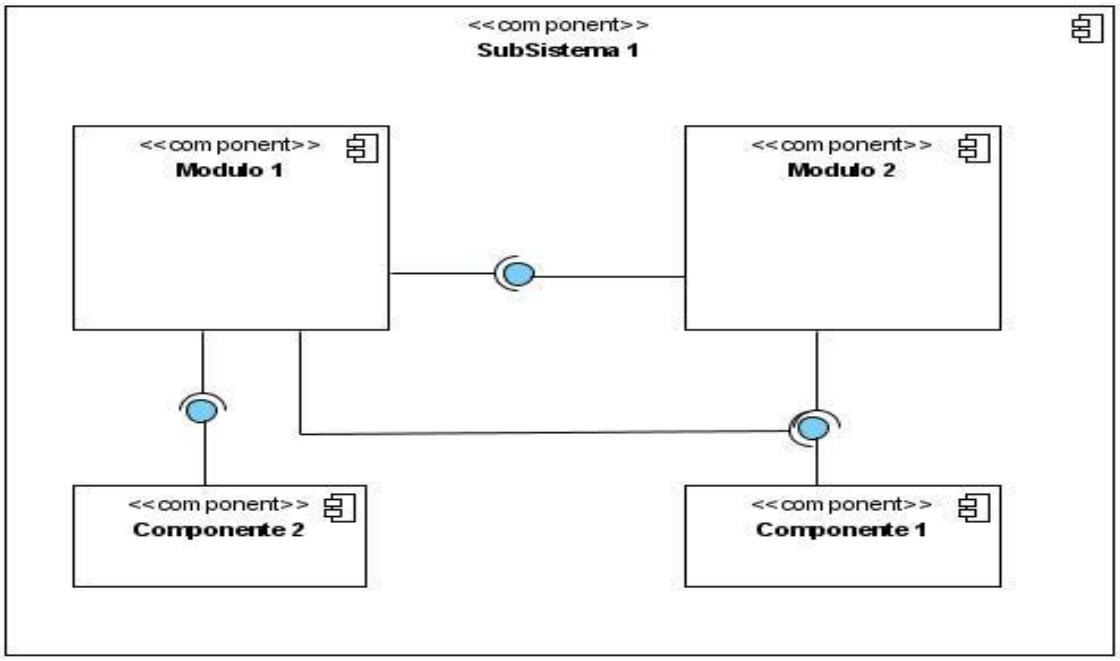


Ilustración 2: Arquitectura de un subsistema Quarxo

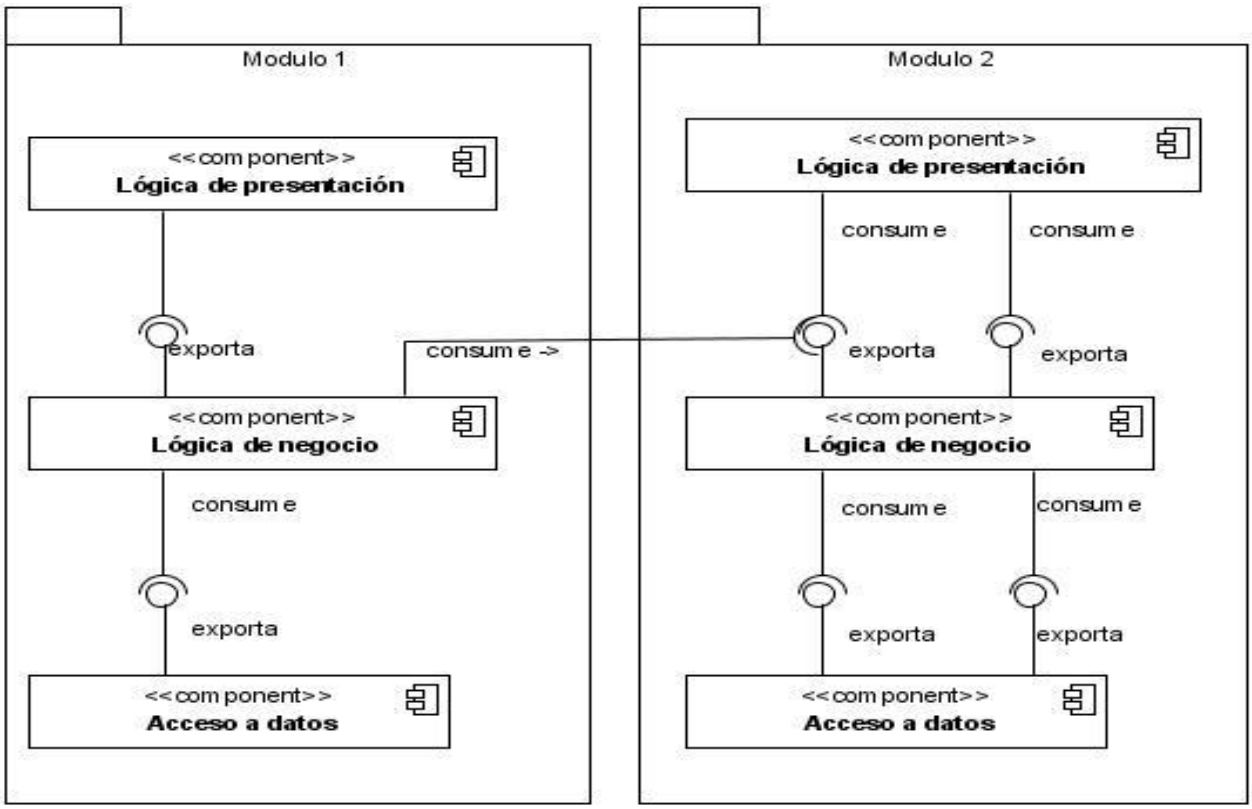


Ilustración 3: Arquitectura de los módulos en el sistema Quarxo

2.5 Modelo de diseño

Es el modelo de un sistema que representa un grupo de objetos que interactúan entre sí, enfocándose en el impacto de los requisitos sobre el sistema, que atiende a las descripciones de los mismos. Asiduamente se emplea como entrada en las actividades de la implementación, pues permite abstraerse en el contexto funcional del sistema, para tener una idea abarcadora de qué se va a hacer y cómo se va a hacer. El modelo de diseño genera una serie de artefactos que funcionan como una guía realizada por el diseñador, e interpretada por el desarrollador, con el propósito de llevar a cabo la implementación del software, basándose en todas las peticiones definidas por el cliente. (27)

2.5.1 Modelo de paquetes

Los paquetes facilitan obtener una organización estructural en los sistemas que se desarrollan, ya que admiten descomponer sistemas de gran tamaño en componentes más pequeños y manejables. Asimismo se encargan de representar las dependencias existentes entre estas descomposiciones lógicas, para obtener una clara idea de cómo interactúan entre sí. Mediante este modelo podemos adquirir y comprender la jerarquía lógica y física que complementa el funcionamiento adecuado de los sistemas. (28)

Para proporcionar una comprensión más sencilla de la aplicación se dispuso organizar por criterios y en paquetes, las clases y ficheros de un módulo, agrupándose los mismos según la función que cumplen. A continuación se nombran las clases que componen cada paquete.

Paquete configuration: Contiene los ficheros *XML* de configuración de los diferentes contextos de *Spring*:

- dataaccess.xml: Contexto de acceso a datos.
- business.xml: Contexto de negocio.
- webflow.xml: Contexto de *Spring WebFlow*.
- servlet.xml: Contexto de *Spring*.

Paquete flow: Contiene los flujos *.XML* del módulo correspondiente.

Paquete manager: Contiene las interfaces e implementaciones del negocio del módulo correspondiente.

Paquete facade: Contiene las interfaces y la implementación de las clases encargadas de proporcionar un punto de entrada a las funcionalidades que brinda un módulo.

Paquete web: Este paquete es el contenedor de los paquetes mvc y webFlow.

Paquete mvc: Contiene los paquetes donde se encuentra toda la lógica de presentación del servidor para *Spring MVC*:

- Paquete commad: Contiene clases que representan objetos a manipular en los formularios.
- Paquete validator: Contiene las clases encargadas de validar los datos.
- Paquete propertyEditor: Contiene las clases para convertir objetos.
- Paquete controller: Contiene las diferentes clases que heredan de los controladores de *Spring*.

Paquete webFlow: Contiene los paquetes donde se encuentra toda la lógica de presentación del servidor para *Spring WebFlow*:

- Paquete serviceFlow: Contiene las diferentes clases que median entre el flujo y el facade.
- Paquete commad: Contiene clases que representan objetos a manipular en los formularios.
- Paquete validator: Contiene las clases encargadas de validar los datos.
- Paquete propertyEditor: Contiene las clases para convertir objetos.

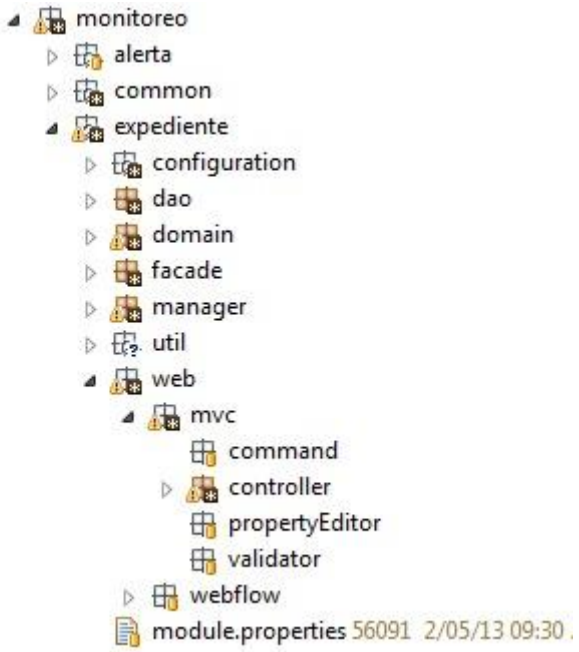


Ilustración 4: Configuración de paquetes del subsistema Monitoreo

Diagrama de paquetes Módulo Expediente

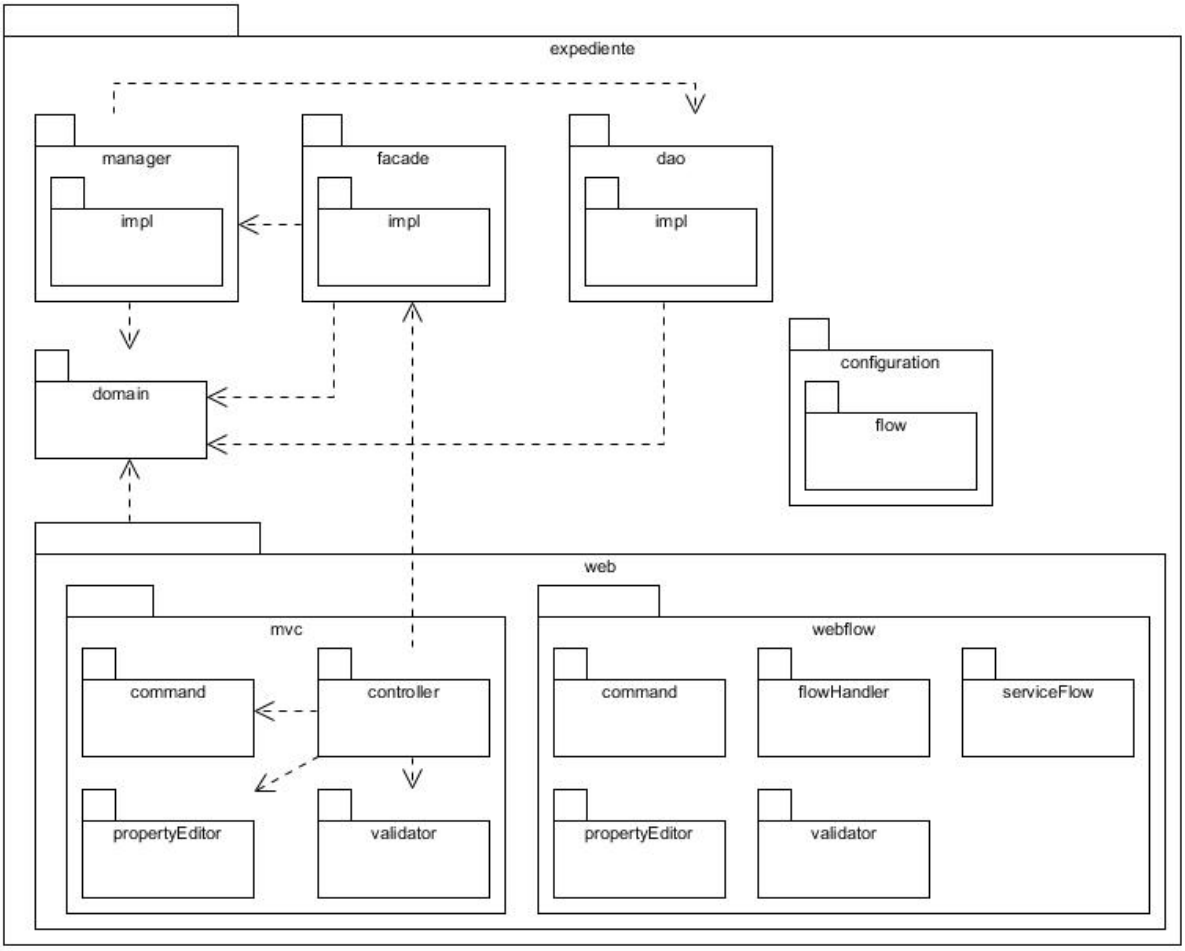


Ilustración 5: Modelo de paquetes correspondiente al módulo Expediente

El módulo Expediente se encarga de la gestión de los expedientes de las cuentas de clientes y contiene los siguientes paquetes:

Paquete configuration: En este paquete se encuentran los ficheros de configuración en formato *XML* de los diferentes contextos de *Spring* para el módulo.

Paquete flow: En este paquete se guardan los archivos XML que definen los flujos para *Spring WebFlow*.

Paquete facade: En este paquete se encuentra la interfaz y su respectiva implementación, clases encargadas de brindar las funcionalidades que serán usadas por la presentación, así como las que se consumen por otros módulos.

Paquete manager: En este paquete se hallan las interfaces y las implementaciones que comprenden la lógica de negocio existente en el módulo, que será brindada a las capas superiores.

Paquete dao: En el paquete dao se encuentran igualmente las interfaces y las implementaciones encargadas de brindar los servicios de comunicación con la base de datos, además de hallarse también los ficheros de mapeo utilizados por *Hibernate* correspondientes al módulo.

Paquete domain: Aquí se localizan las clases relacionadas con el dominio del módulo en cuestión.

Paquete web: El paquete web agrupa un conjunto de clases y paquetes que son los encargados de realizar todo lo referente a la presentación en el lado del servidor.

Paquete mvc: En este subpaquete se agrupa todo lo referente a la lógica de presentación cuando se hace uso de *SpringMVC*.

Paquete webflow: En este subpaquete se agrupa todo lo referente a la lógica de presentación cuando se hace uso de *Spring WebFlow*.

Paquete controller: En este paquete se hallan las clases que extienden de los controladores propuestos por *Spring MVC*, encargadas de responder a las peticiones realizadas por el cliente.

Paquete serviceFlow: Contiene las clases encargadas de establecer la comunicación entre el flujo y la fachada del módulo.

Paquete flowHandler: Clases utilizadas para personalizar el trabajo con el *WebFlow*.

Paquete command: Se encuentran las clases que representan los datos que son introducidos y mostrados en los formularios de la interfaz de usuario.

Paquete propertyEditor: Agrupa las clases que actúan como intermediarias para convertir datos de la interfaz de usuario en objetos y viceversa.

Paquete validator: Cuenta con las clases encargadas de realizar la validación de datos en el lado del servidor.

2.5.2 Diagramas de clases del diseño

Los diagramas de clases se utilizan para modelar la visión estática de un sistema. Esta visión soporta los requisitos funcionales del sistema en concreto y los servicios que el sistema debería proporcionar a sus usuarios finales. Habitualmente contiene clases, interfaces y relaciones entre ellas de asociación, dependencia y/o generalización. Pueden contener paquetes o subsistemas, que se usan para agrupar elementos del modelo en partes más grandes. (29)

Diagrama de clases del diseño del módulo Expediente

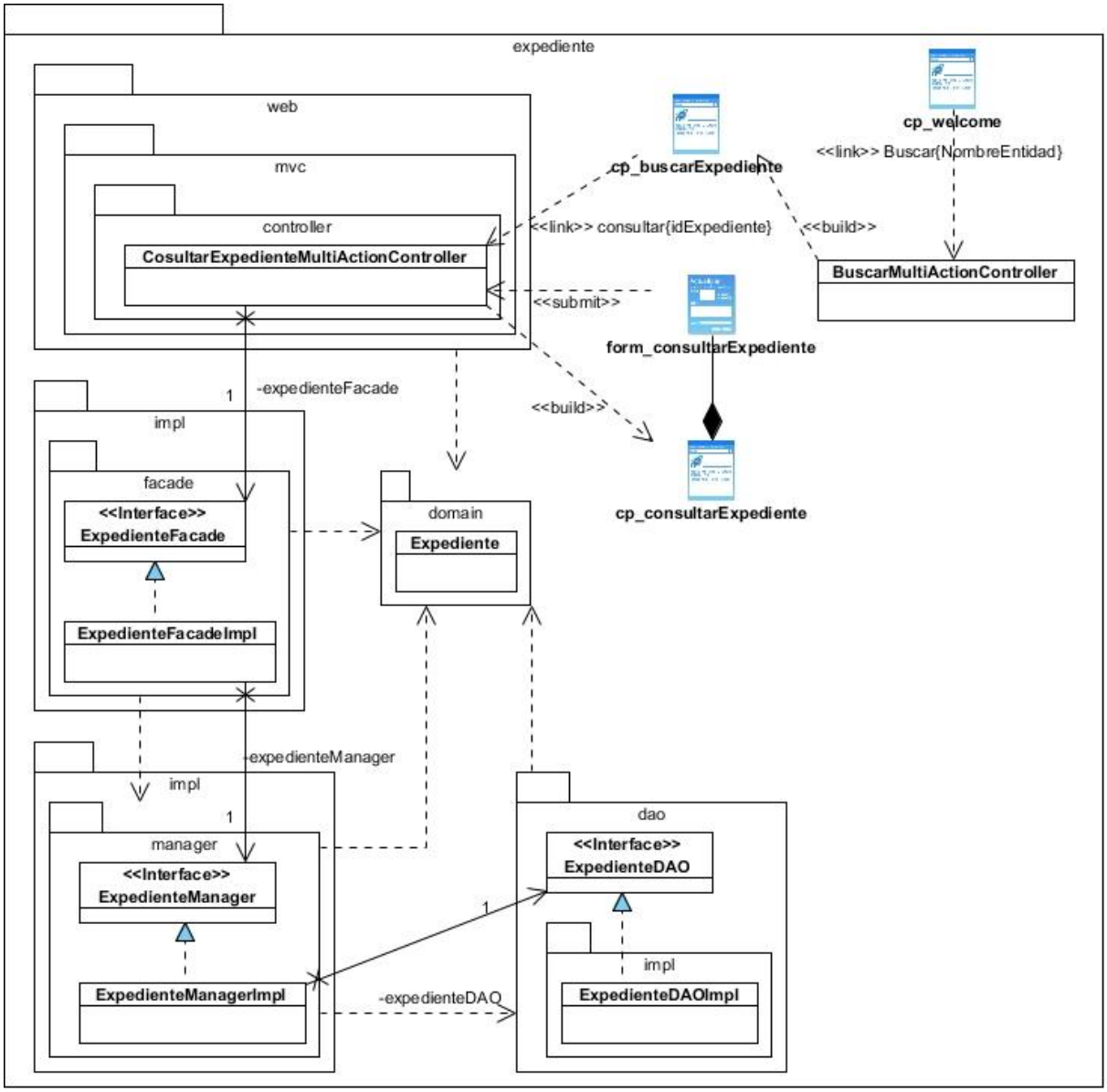


Ilustración 6: Diagrama de clases del diseño correspondiente al módulo Expediente

Debido a la utilización de *frameworks* y los estándares definidos por el grupo de arquitectura del proyecto, en la realización de los diagramas mostrados con anterioridad, fue necesario diseñar determinadas clases que pudieran crear confusión al tratar de comprender dichos diagramas. A continuación se brinda una breve descripción de dichas clases, para una mejor compresión de los diagramas.

Clases *ClientPage*: Páginas web encargadas de mostrar los formularios e información al usuario.

ConsultarExpedienteMultiAction: Clase que gestiona las acciones asociadas al flujo consultar Expediente.

ExpedienteFacade: Interfaz encargada de brindar las funcionalidades del módulo a la capa de presentación y a otros módulos que la requieran.

ExpedienteFacadeImpl: Clase encargada de implementar la lógica de programación de la Interfaz ExpedienteFacade.

ExpedienteManager: Clase que contiene las funcionalidades que se deben implementar en la clase ExpedienteManagerImpl, para dar respuesta a las acciones que se solicitan desde ExpedienteFacadeImpl.

ExpedienteDAO: Interfaz de comunicación que se encarga de brindar las funcionalidades de la capa de Acceso a Datos.

ExpedienteDAOImpl: Clase que implementa las funcionalidades de la capa de Acceso a Datos.

2.5.3 Diagramas de interacción (secuencia)

Los diagramas de secuencia de un sistema, posibilitan representar la interacción entre los objetos de la aplicación y comprender los datos enviados y recibidos entre dichos objetos. Son considerados herramientas valiosas para presentar y examinar las operaciones, permitiéndose realizar un seguimiento con grandes detalles de los procesos que se desarrollan. Para las interacciones entre los objetos, se manejan una serie de mensajes que representan la secuencia a seguir en cada caso.

Escenarios

Describen la secuencia de pasos exitosos a ocurrir en las posibles interacciones, facilitándose diferentes tipos de información según los niveles de detalle del sistema. Son técnicas para obtener

requerimientos desde varias perspectivas debido a que se centran en las interacciones, lo que permite tener una vista adecuada del procedimiento a seguir en cada requerimiento.

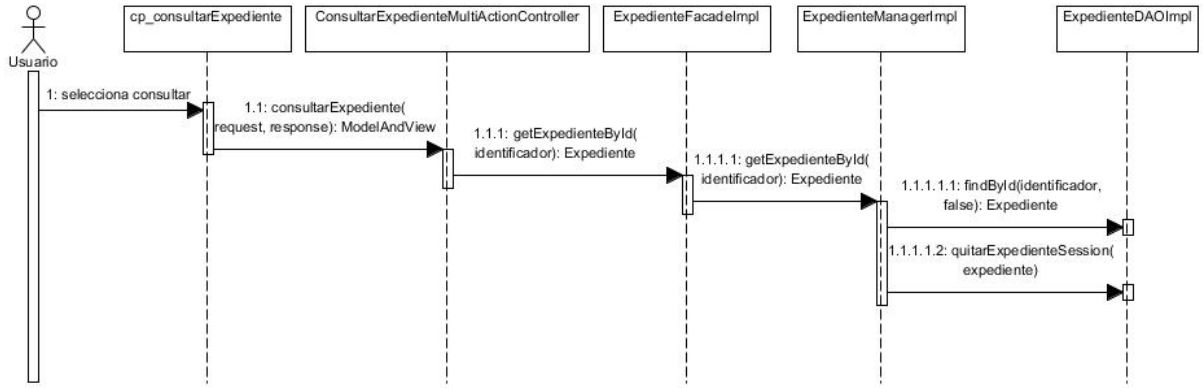


Ilustración 7: Diagrama de secuencia de la operación consultar expediente 1er escenario

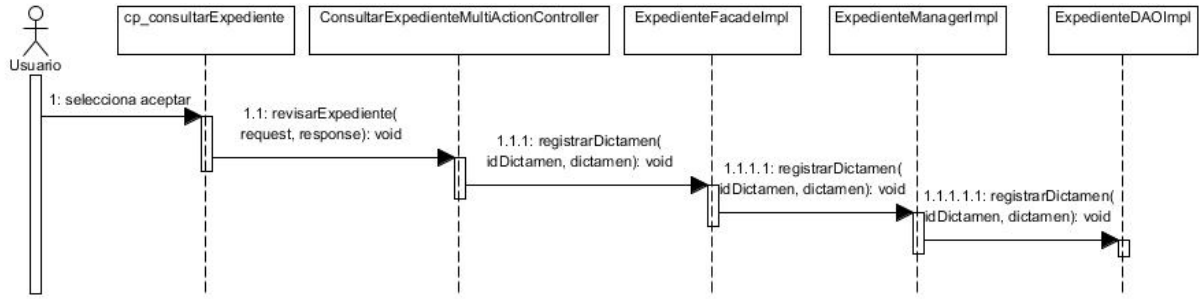


Ilustración 8: Diagrama de secuencia de la operación consultar expediente 2do escenario

2.5.4 Modelo de datos

La persistencia de datos se considera la pieza principal en todo sistema de información. Mediante el acceso a los datos almacenados se puede administrar, planear, controlar y tomar decisiones en cualquier organización. Por este motivo se torna necesaria la realización de un diseño de datos equilibrado y con calidad para gestionar grandes volúmenes de información.

El modelo de datos debe ser capaz de modelar la información que transforma el software, las funciones para permitir que ocurran las transformaciones y el comportamiento del sistema cuando ocurren estas transformaciones. Este se compone de tres piezas de información interrelacionadas: el objeto de datos, los atributos y la relación que conecta los objetos entre sí. (26)

A continuación se presenta parte del modelo de datos construido, para conseguir la interacción de los requisitos identificados con los datos almacenados en el sistema.

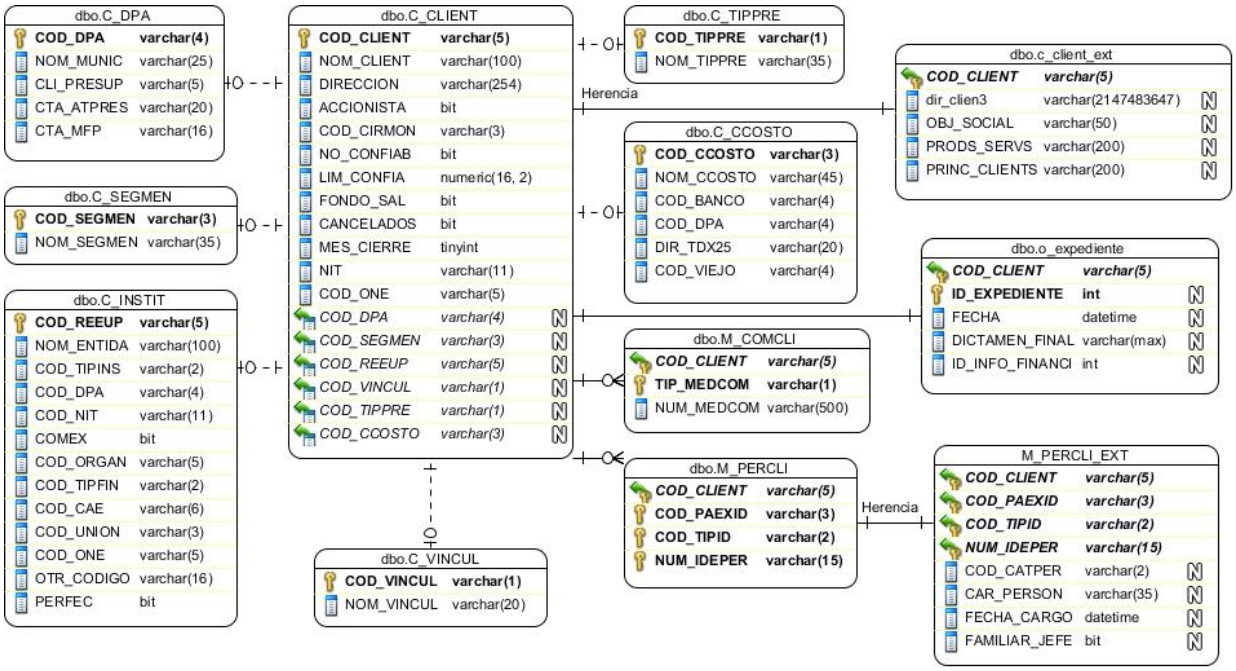


Ilustración 9: Modelo de datos correspondiente al subsistema Monitoreo

El modelo de datos lo conforman 12 tablas: **C_DPA**, **C_SEGMEN**, **C_INSTIT**, **C_CLIENT**, **C_CLIENT_EXT**, **C_VINCUL**, **C_TIPPRE**, **C_CCOSTO**, **M_COMCLI**, **M_PERCLI**, **O_EXPEDIENTE** y **M_PERCLI_EXT**.

La tabla **C_CLIENT** recoge los datos del cliente jurídico una vez que se registran en el sistema y forma parte como padre, de una relación de herencia que incluye como hija a la tabla **C_CLIENT_EXT,** con el propósito de manejar datos adicionales. La tabla **O_EXPEDIENTE** recoge datos específicos de los expedientes de una cuenta de cliente, como la fecha de revisión y las notas relacionadas a la misma. La tabla **M_PERCLI** acopia todas las personas asociadas al cliente jurídico, la cual tiene como extensión a la tabla **M_PERCLI_EXT,** que hereda directamente de ella para manejar información adicional. La tabla **M_COMCLI** acumula todos los medios de comunicación del cliente jurídico. El resto de las tablas manejan datos más generales del cliente jurídico, como la división político administrativa (**C_DPA**), el segmento (**C_SEGMEN**), la institución (**C_INSTIT**), la vinculación (**C_VINCUL**), el tipo de presupuesto (**C_TIPPRE**) y el código de costo (**C_CCOSTO**).

2.6 Patrones de diseño utilizados

El diseño fue elaborado siguiendo patrones basados en la experiencia, que de manera general constituyen soluciones simples y elegantes a problemas específicos o comunes del diseño orientado a

objetos. En el capítulo 1 se describieron de manera general cada uno de los patrones a emplear en el desarrollo de la solución, a continuación se especifica la aplicación de los mismos.

Patrones GRASP:

- **Experto:** Dicho patrón se evidencia en la definición de las clases de acuerdo a las funcionalidades que deben realizar a partir de la información que manejan. Específicamente la clase *ExpedienteDAO*, será la responsable de efectuar las operaciones que conciernen a las funciones: registrar, cancelar y consultar los expedientes de cuentas de cliente. Sobre este mismo principio se realiza el diseño de las restantes funcionalidades.
- **Alta Cohesión:** Este patrón se manifiesta en el diseño del componente de manera general, donde se agruparon las clases en dependencia de los requerimientos a los que se les debía dar respuesta, según la premisa de que cada clase debe implementar las operaciones que estén sobre la misma área funcional.
- **Controlador:** Se utiliza a través de las clases controladoras de los módulos, que tienen la responsabilidad de escuchar y responder las peticiones realizadas por la presentación y de comunicarse con el negocio.
- **Bajo Acoplamiento:** Se maneja a través de las interfaces e implementaciones, como la interfaz *ExpedienteFacade* y su implementación *ExpedienteFacadeImpl,* para convenir que clases como *ConsultarExpedienteMultiActionController,* se relacionen únicamente con ellas para realizar sus operaciones, reduciéndose el impacto de cambios posteriores en el negocio del sistema y lograr que el código sea más fácil de entender, mantener y reutilizar.

Patrones GoF:

- **Fachada:** Se declara mediante la interfaz *ExpedienteClienteFacade*, responsable de la comunicación entre la presentación y el grupo de clases e interfaces más complejas que se encargan de la lógica de negocio y el acceso a datos. Permite reducir la dependencia entre clases y ofrece un punto de acceso al resto de las clases, sin ocultar las mismas. Por medio de su utilización se conocen cuales clases del subsistema son responsables de las peticiones, delegándose así las peticiones realizadas a los objetos correspondientes.
- **Cadena de Responsabilidad:** Se conforma a través del flujo que surge, cuando la vista realiza una petición a alguna información ubicada en la base de datos, siendo atendida por los controladores (*ConsultarExpedienteMultiActionController*), luego manejada por la fachada (*ExpedienteFacade*), seguidamente tratada por el *manager* (*ExpedienteManager*) para finalizar en el DAO (*ExpedienteDAO*).

Patrón Modelo Vista Controlador (MVC):

Se demuestra su aplicación en la arquitectura dividida en tres capas. La capa de presentación con la inclusión de la clase *ExpedienteMultiActionController*. La capa de negocio con la inclusión de las clases *ExpedienteFacade* y *ExpedienteManager*. Por último la capa de acceso a datos, que contiene las interfaces *DAO* por cada funcionalidad y sus implementaciones.

Patrón de Acceso a Datos (DAO):

Se aplica con la definición de las interfaces *DAO*, para disminuir la complejidad de los objetos del dominio, liberándolos de la responsabilidad de manejar la implementación de sus fuentes de datos. Su utilización está centrada en abstraer y encapsular todos los accesos a la fuente de datos.

2.7 Conclusiones parciales

En este capítulo se analizaron los procesos necesarios, para ejecutar el flujo de actividades a realizar en el monitoreo de las cuentas de clientes, comprendido en el campo de acción de la operadora de monitoreo y control dentro del BNC. Se hizo posible precisar las actividades vitales que se realizan en los procesos del negocio, obteniéndose como principal resultado la descripción de los procesos de negocio, asociados al dominio del problema a resolver.

Se definieron los requisitos a implementar en base a las peculiaridades detectadas, con el fin de contribuir al buen funcionamiento del sistema *Quarxo*, estudiándose previamente la arquitectura propuesta para el software. Se crearon los artefactos a ser empleados en la implementación de las nuevas funcionalidades, en este caso los diagramas pertenecientes a la fase de diseño. Todo el material expuesto en este capítulo, conforma los cimientos para lograr un desarrollo acorde a la estructura, que posee actualmente el sistema en explotación.

CAPÍTULO 3 IMPLEMENTACIÓN Y VALIDACIÓN DE LA SOLUCIÓN PROPUESTA

3.1 Introducción

En el siguiente capítulo se aborda la descripción de los elementos esenciales en la etapa de implementación y pruebas del sistema. Se detallan los estándares de codificación definidos para la escritura del código. Contiene además la interacción de los componentes relacionados con los requerimientos detectados, representada mediante el diagrama de componentes. También se muestra parte del código fuente, de uno de los principales métodos con su respectiva descripción. Finalmente se prueba la solución mediante los métodos de caja negra y caja blanca, con el objetivo de encontrar errores que imposibiliten el correcto funcionamiento del producto.

3.2 Implementación

La implementación es un proceso de varias fases que comienza cuando se crea una aplicación en el equipo de un desarrollador y termina cuando está instalada y lista para ejecutarse en el equipo de un usuario. Describe cómo los elementos del modelo de diseño se implementan en términos de componentes, en otras palabras, toma el resultado del modelo de diseño para generar el código final del sistema. Dicho código está determinado por el lenguaje de programación y tiene como objetivo llevar a cabo la implementación de cada una de las clases significativas del diseño.

Estándares de codificación

Los estándares de codificación se definen por el equipo de desarrollo, para lograr estandarización en la programación del software. Engloban todos los aspectos relacionados con la generación de código, de tal manera que sea prudente, práctico y entendible para todos los programadores. Un código fuente completo debe reflejar un estilo armonioso y uniforme, como si un único programador hubiera escrito todo el código.

El desarrollo de una aplicación informática exige que se adopten estándares de codificación y estilo. La práctica de dichos de estándares asegura la legibilidad del código entre distintos desarrolladores, permite la guía para el mantenimiento o actualización del sistema, además de facilitar la portabilidad entre plataformas y aplicaciones. (30)

Convenciones de nomenclatura

En la implementación del sistema *Quarxo* se maneja la notación *PascalCase* para la nomenclatura de las clases e interfaces y la notación *CamelCase* para nombrar las variables y métodos presentes en dichas clases.

***PascalCase*:** Plantea que los identificadores, nombres de variables, métodos y funciones que están compuestos por múltiples palabras juntas, iniciarán cada palabra con letra mayúscula. Ejemplo: ExpedienteController.java, ExpedienteDAO.java

***CamelCase*:** Plantea que los identificadores, nombres de variables, métodos y funciones que están compuestos por múltiples palabras juntas, iniciarán cada palabra con letra mayúscula, excepto la primera palabra. Ejemplo: buscarCuentaXCodCliente(String codCliente), getExpedienteById(int identificador).

Nomenclatura según el tipo de clases

A partir de las notaciones definidas para la codificación de las clases del sistema *Quarxo*, se establece una nomenclatura para las mismas según la capa de la arquitectura a la que pertenezcan, para facilitar su ubicación y el mantenimiento del código. A continuación se muestran las nomenclaturas definidas por capas.

Capa de Presentación:
Las clases que pertenecen a los sub-paquetes *mvc* y *webflow* del paquete *web* tienen la siguiente nomenclatura:

- **Paquete *controller:*** (Nombre de la funcionalidad a la que responde) + (Nombre del controlador de *Spring MVC* que se hereda). Ejemplo: *ExpedienteMultiActionController*.
- **Sub-paquetes:** (Nombre de la funcionalidad a la que responde) + (Nombre del paquete). Ejemplo: *ExpedienteValidator*.

Capa de Negocio:
A las clases contenidas en los paquetes *manager* y *facade* y sus sub-paquetes *impl* se les define la siguiente nomenclatura:

- **Interfaces:** (Nombre del módulo) + (Nombre del paquete). Ejemplo: *ExpedienteManager*.
- **Sub-paquetes impl:** (Nombre del módulo) + (Nombre del paquete) + *Impl*. Ejemplo: *ExpedienteManagerImpl*.

Capa de Acceso a Datos:

Las clases que están contenidas por el paquete *dao* y su sub-paquete *impl* presentan la siguiente nomenclatura:

- **Paquete dao:** (Nombre de la entidad) + *DAO*. Ejemplo: *ExpedienteDAO*.
- **Sub-paquete impl:** (Nombre de la entidad) + *DAOImpl*. Ejemplo: *ExpedienteDAOImpl*.

3.2.1 Diagrama de componentes

Los diagramas de componentes describen los elementos físicos del sistema y sus relaciones. Permiten modelar la vista estática del sistema, mostrándose las dependencias lógicas entre un conjunto de componentes de software. Estos pueden ser de código fuente, librerías, binarios o ejecutables. (31)

A continuación se explican de manera general los componentes utilizados.

El componente Web contiene los componentes *mvc* y *webFlow*, los que ofrecen un conjunto de clases que constituyen la lógica de presentación del servidor para *SpringMVC* y para *SpringWebFlow* respectivamente.

Los componentes *Manager* y *Facade* contienen las clases con las implementaciones del negocio y las clases encargadas de proporcionar un punto de entrada a las funcionalidades respectivamente.

En el caso del componente Expediente-Component, engloba un grupo de clases necesarias para resolver todo el negocio, incluyéndose las que conforman el modelo de datos y las clases *DAO*.

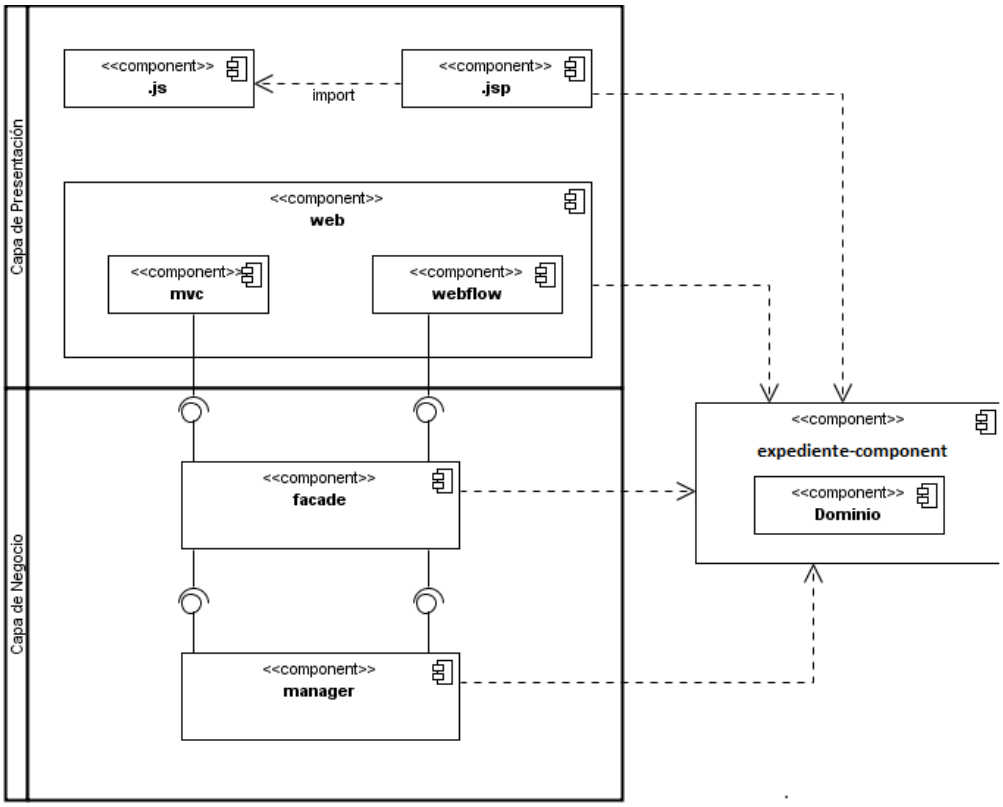


Ilustración 10: Diagrama de Componentes

3.2.2 Diagrama de despliegue

El Diagrama de Despliegue modela los aspectos físicos y la vista de despliegue estática de un sistema, así como la topología del hardware donde se ejecuta el mismo. (31)

A continuación se muestra el Diagrama de Despliegue del sistema *Quarxo*.

Ilustración 11: Diagrama de Despliegue del sistema Quarxo

3.2.3 Descripción de clases y funcionalidades

Teniéndose en cuenta el diseño de las clases, correspondientes al subsistema Monitoreo realizado en el capítulo anterior, a continuación se describen los atributos y métodos de las clases que son más importantes para el sistema desde el punto de vista funcional.

Nombre: ConsultarExpedienteMultiActionController	
Tipo de clase: Controladora	
Atributo	**Tipo**
expedienteFacade	ExpedienteFacade
fotoPath	final String
resourceLoader	org.springframework.core.io.ResourceLoader
Para cada responsabilidad:	
Nombre:	**Descripción**:
consultarExpediente(HttpServletRequest request, HttpServletResponse response)	Permite obtener el objeto Expediente que servirá de modelo a la vista, en caso de que este se vaya a consultar.
getJsonMediosComunicacion(List<MedioComunicacionCliente> medios)	Permite obtener una lista de los medios de comunicación del cliente jurídico que se vaya a mostrar.
getJsonPersonasAsociadas(List<PersonaAsociadaCliente> personas)	Permite obtener una lista de personas asociadas al cliente jurídico que se vaya a mostrar.
getJsonMediosComunicaPersonaAsoc(HttpServletRequest request, HttpServletResponse response)	Permite obtener una lista de los medios de comunicación de las personas asociadas al cliente jurídico que se vayan a mostrar.
buscarFoto(HttpServletRequest request, HttpServletResponse response)	Permite cargar la imagen del carnet de identidad de las personas asociadas al cliente jurídico.

buscarFirma(HttpServletRequest request, HttpServletResponse response)	Permite cargar la imagen de la firma de las personas asociadas al cliente jurídico.
revisarExpediente(HttpServletRequest request, HttpServletResponse response)	Permite agregar un campo de observación para registrar los datos de la revisión de un expediente de cuentas de cliente.
getJsonCuentasClientes(String codCliente)	Permite obtener una lista de las cuentas asociadas al cliente jurídico que se vayan a mostrar.
imprimirExpediente(HttpServletRequest request, HttpServletResponse response)	Permite imprimir los datos del objeto Expediente.

Tabla 5: Descripción de la clase ConsultarExpedienteMultiActionController

Nombre: ExpedienteFacadeImpl	
Tipo de clase: Fachada	
Atributo	**Tipo**
expedienteManager	ExpedienteManager
Para cada responsabilidad:	
Nombre:	**Descripción**:
getExpedienteById(int identificador)	Obtiene un objeto Expediente dado su identificador.
quitarExpedienteSession(Expediente expediente)	Retira los datos de un objeto Expediente dado.
registrarDictamen(int idDictamen, String dictamen)	Registra las observaciones de revisión de un expediente.
buscarCuentaXCodCliente(String codCliente)	Agrupa las cuentas de cliente de un cliente jurídico dado un código de cliente.
imprimirExpediente(int identificador)	Permite imprimir los datos de un expediente dado un identificador.

Tabla 6: Descripción de la clase ExpedienteFacadeImpl

3.3 Validación de la solución propuesta

La calidad de un producto de software es el conjunto de cualidades que lo caracterizan y determinan su utilidad y existencia. Se ha convertido en un elemento estratégico de las grandes organizaciones, debido a su fuerte impacto en la competitividad de las empresas. Durante el proceso de desarrollo de software las posibilidades de errores son múltiples, por lo que se hace necesario detectar a tiempo las imperfecciones e irregularidades y proporcionar una visión objetiva de la madurez y calidad de los procesos asociados. (32)

El aspecto fundamental que rige esta etapa es determinar mediante pruebas, cómo y en qué sentido el subsistema Monitoreo cumple con las expectativas del cliente, a partir de los requisitos establecidos y las restricciones impuestas.

Bajo ese ámbito se trazan los siguientes objetivos:

- Verificar la implementación del subsistema.
- Verificar que todos los requisitos se han implementado correctamente.
- Identificar errores y asegurar que estos sean corregidos de la mejor manera.

Se aplicarán los métodos de pruebas adaptados a este nivel de Caja Blanca, para comprobar los caminos lógicos del software (código) y de Caja Negra para probar que las funciones del software son operativas (interfaces), que la entrada se acepta de forma adecuada y que se produce un resultado correcto, manteniéndose la integridad de la información externa.

3.3.1 Pruebas de unidad

Se desarrollaron pruebas por parte del grupo de desarrollo del proyecto, para verificar el resultado de la implementación. Las pruebas realizadas permitieron identificar posibles errores en la documentación y el software, es decir, requisitos que el producto debiera cumplir, que aún no se cumplían.

Pruebas de Caja Blanca o Estructura

Las pruebas estructurales o de caja blanca del software, se basan en el minucioso examen de los detalles procedimentales. Se comprueban los caminos lógicos del software proponiéndose casos de prueba que ejerciten conjuntos específicos de condiciones y/o bucles. Se examina el estado del programa en varios puntos, para determinar si el estado real coincide con el esperado. (26)

Asociadas a este método existen cuatro técnicas de prueba: Condición, Flujo de Datos, Bucles y Camino Básico, de las cuales será aplicada esta última, debido a que permite obtener una medida de la complejidad lógica del diseño y usar la misma como guía para la definición de un conjunto de caminos básicos, garantizándose que durante la prueba, se ejecute al menos una vez cada sentencia del programa.

Aplicación de las pruebas de Caja Blanca

Estas pruebas estructurales se realizaron mediante el *framework JUnit*, el cual permite la automatización de las pruebas de aplicaciones en *Java,* integrándolo al *IDE Eclipse*. El mismo consta de un conjunto de clases que fueron usadas para construir los casos de prueba y ejecutarlos automáticamente.

A continuación se muestran las imágenes correspondientes a la realización de las pruebas de Caja Blanca, aplicándose el *framework JUnit* a uno de los métodos de la implementación llamado *GetJsonMediosComunicaPersonaAsoc*, el cual se localiza en la clase controladora *ConsultarExpedienteMultiActionController*, dentro del módulo Expediente del subsistema Monitoreo.

```
public void getJsonMediosComunicaPersonaAsoc(HttpServletRequest request, HttpServletResponse response){
    JSONObject json = new JSONObject();
    JSONArray array = new JSONArray();
    if(request.getParameter("id")!=null){
        PersonaNatural persona = globalFacade.obtenerPersonaNatural(request.getParameter("id"));
        List<MedioComunicacionPersona> medios = persona.getMediosComunicacion();
        if(medios!=null){
        for (int i = 0; i < medios.size(); i++) {
            MedioComunicacionPersona medio = medios.get(i);
            JSONObject data = new JSONObject();
            data.put("tipo", medio.getNumero());
            data.put("valor", medio.getTipoMedioComunicacion().getNombre());
            array.add(data);
        }
        }
        json.put("items", array);
    }
    json.put("items", array);
    try {
        ResponseUtil.escribirDatosEnElResponse(response, json.toString());
    } catch (IOException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
}
```

Ilustración 12: Método a probar por el framework JUnit

Para probar dicho método primeramente se crea una clase, nombrada en este caso como *TestJUnit*, la cual debe extender de *TestCase*, como se observa en la ilustración a continuación.

```
public class TestJUnit extends TestCase {

    ConsultarExpedienteMultiActionController consultarExpediente;
    public static final String KEY = "expediente";
    private MockControl controlHttpServlet;
    private HttpServletRequest mockHttpServletRequest;
    private MockControl controlHttpResponse;
    private HttpServletResponse mockHttpServletResponse;
    private MockControl controlHttpSession;
    private HttpSession mockHttpSession;

    @Before
    public void setUp() throws Exception {
        consultarExpediente = new ConsultarExpedienteMultiActionController();

        ApplicationContext context = new ClassPathXmlApplicationContext(
        "classpath:cu/uci/finixubnc/monitoreo/jUnit/monitoreo-context.xml");

        ExpedienteManagerImpl manager = (ExpedienteManagerImpl) context.getBean("ExpedienteManager")
        ExpedienteFacadeImpl facade = new ExpedienteFacadeImpl();
        facade.setExpedienteManager(manager);

        consultarExpediente.setExpedienteFacade(facade);
        controlHttpServlet = MockControl.createControl(HttpServletRequest.class);
        mockHttpServletRequest=(HttpServletRequest) controlHttpServlet.getMock();

        controlHttpResponse = MockControl.createControl(HttpServletResponse.class);
        mockHttpServletResponse =(HttpServletResponse)controlHttpResponse.getMock();

        controlHttpSession = MockControl.createControl(HttpSession.class);
        mockHttpSession = (HttpSession) controlHttpSession.getMock();
        super.setUp();
```

Ilustración 13: Clase TestJUnit

Después de creada la clase de prueba, se crea un objeto de la clase en la cual se encuentra el método a probar y dos simulacros de los parámetros que se le pasan a dicho método, uno de *HttpServletRequest* y otro de *HttpServletResponse*, con sus respectivos *Mock* y *Control* encargados de su control. Esto lo posibilita la librería *EasyMock*, que trabaja en la creación de un proxy dinámico para dicha simulación.

Posteriormente en el método *setUP* se inicializan todos los atributos anteriormente creados, además de un contexto que contenga todos los *beans*, de las clases necesarias para la ejecución de la operación que se va a probar. Debido a la arquitectura del subsistema, se debe crear un objeto por cada capa a la que se le haga énfasis en el método a probar. Estos pasos se van a configurar en el fichero *expediente-context.xml*, el cual va a ser el encargado de declarar y mapear los objetos creados por cada nivel de la aplicación.

```
@After
public void tearDown() throws Exception {
    controlHttpServlet.verify();
}

@Test
public void testGetJsonMediosComunicaPersonaAsoc() {
    mockHttpServletRequest.getParameter("id");
    controlHttpServlet.setReturnValue("87022508102");

     controlHttpServlet.replay();

    assertNotNull(mockHttpServletResponse);
}
```

Ilustración 14: Métodos TearDown y testGetJsonMediosComunicaPersonaAsoc

El método *tearDown* es el encargado de informar la existencia de los errores en la realización de las pruebas. Ahí es donde se verifican si las llamadas a los métodos se realizaron correctamente. Las verificaciones en este método se realizan de forma automática, para todos los test definidos en la clase creada, en este caso, *TestJUnit*.

Por último se preparan los parámetros de pruebas necesarios al método *getJsonMediosComunicaPersonaAsoc* y el identificador de cada uno, para ser pasados por el *mockHttpServletRequest*. Posteriormente se realiza la condición de prueba a través del método *assertNotNull,* el cual valida que el objeto pasado no sea nulo. Se ejecuta el procedimiento que se va a analizar y se espera a que el *framework* termine el análisis.

JUnit muestra en una ventana los resultados obtenidos de las pruebas realizadas, representándolos con una línea de color verde si son satisfactorios, o una de color rojo en caso contrario. A continuación se muestra el caso favorable para dicha prueba.

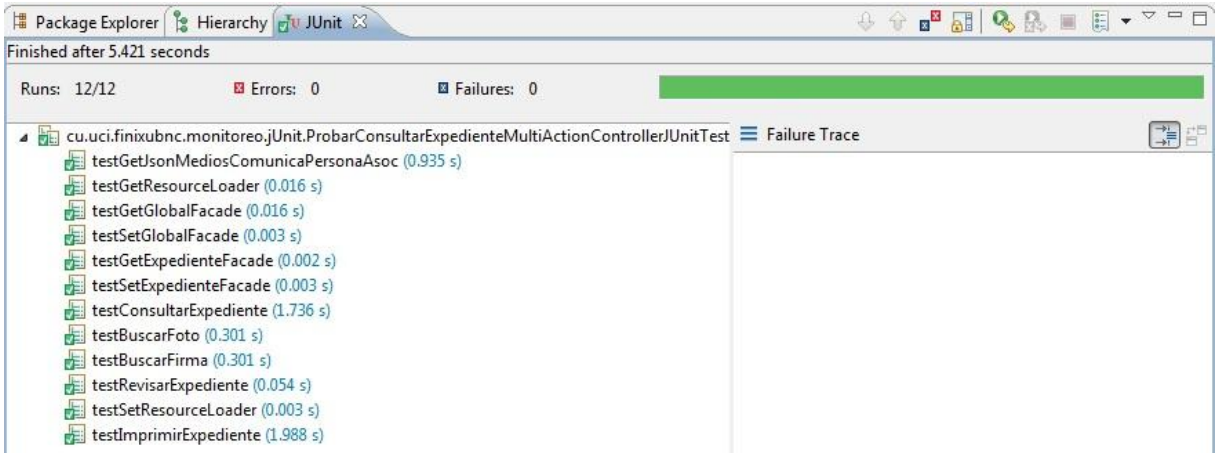


Ilustración 15: Resultados de las pruebas con JUnit

Pruebas de Caja Negra o Funcional

Las pruebas funcionales o de caja negra se centran en los requisitos funcionales del software. Permiten al ingeniero de software obtener conjuntos de condiciones de entrada, que ejerciten completamente todos los requisitos funcionales de un programa. Todas las pruebas se realizan sobre la interfaz de usuario. (26)

La prueba de Caja Negra no es una alternativa a las técnicas de prueba de la Caja Blanca, sino un enfoque complementario que intenta descubrir diferentes tipos de errores, a los encontrados en los métodos de Caja Blanca.

A continuación se mencionan algunos de los errores que detecta:

- Funciones incorrectas o ausentes.
- Errores de interfaz.
- Errores en estructuras de datos o en accesos a la base de datos externa.
- Errores de rendimiento.
- Errores de inicialización y terminación.

Para el correcto desarrollo de las pruebas existen diferentes técnicas, como la Partición de equivalencia, Análisis de valores límites y Grafos de Causa-Efecto**.**

Aplicación de las pruebas de Caja Negra

Para la realización de las pruebas de caja negra se empleó la técnica Partición de Equivalencia. Dicha técnica es una de las más efectivas para examinar los valores válidos e inválidos de las entradas existentes en el software. Dirige la definición de casos de pruebas que descubren clases de errores, reduciéndose el número de casos de prueba a desarrollar.

<table>
<tr><th colspan="2">Caso de Prueba del Requisito Funcional Actualizar Cliente Jurídico</th></tr>
<tr><th colspan="2">Descripción General:</th></tr>
<tr><td colspan="2">El sistema debe permitir actualizar un cliente jurídico, para ello se deben buscar los datos del mismo. De cancelarse esta operación no se efectuará la actualización del cliente jurídico.</td></tr>
<tr><th colspan="2">Condiciones de Ejecución:</th></tr>
<tr><td colspan="2">1. Se debe identificar y autenticar ante el sistema y tener los permisos para ejecutar esta acción.
2. Debe estar creado el cliente jurídico a actualizar.
3. Se debe seleccionar el subsistema Clientes y dentro de este el módulo “Gestionar cliente jurídico”.
4. Se debe seleccionar al desplegarse el menú, la opción: Buscar cliente jurídico.</td></tr>
<tr><th>Escenarios de pruebas</th><th>Flujo del escenario</th></tr>
<tr><td>EP 1.1: Actualizar un cliente jurídico correctamente.</td><td>– Se seleccionan los filtros de búsqueda y se llenan los datos para encontrar el cliente jurídico a actualizar.
– Se presiona el botón Buscar.
– Se selecciona el cliente jurídico a actualizar en la tabla de resultados de la búsqueda y se presiona el botón Actualizar cliente jurídico.
– Se introducen los datos correctamente.
– Se presiona el botón Aceptar.
– Se muestra el mensaje emitido por el sistema con el texto: “Operación realizada satisfactoriamente.”.
– Se presiona el botón Aceptar del mensaje.</td></tr>
<tr><td>EP 1.2: Actualizar un cliente jurídico introduciendo datos inválidos.</td><td>– Se seleccionan los filtros de búsqueda y se llenan los datos para encontrar el cliente jurídico a actualizar.
– Se presiona el botón Buscar.
– Se selecciona el cliente jurídico a actualizar en la tabla de resultados de la búsqueda y se presiona el botón Actualizar cliente jurídico.</td></tr>
</table>

	– Se introducen los datos correspondientes, insertando algunos datos inválidos. – Se presiona el botón *Aceptar***.** – El sistema señala los datos introducidos de forma incorrecta. – El usuario corrige los datos. – Se presiona el botón *Aceptar***.**
EP 1.3: Actualizar un cliente jurídico dejando campos requeridos en blanco.	– Se seleccionan los filtros de búsqueda y se llenan los datos para encontrar el cliente jurídico a actualizar. – Se presiona el botón *Buscar***.** – Se selecciona el cliente jurídico a actualizar en la tabla de resultados de la búsqueda y se presiona el botón *Actualizar cliente jurídico*. – Se introducen los datos correspondientes, dejándose al menos uno de los campos requeridos en blanco. – Se presiona el botón *Aceptar* – El sistema señala los campos obligatorios que faltan por llenar. – El usuario llena los datos. – Se presiona el botón *Aceptar*.
EP 1.4: Actualizar un medio de comunicación correctamente.	– Se selecciona el botón *Actualizar medio de comunicación*. – Se introducen los datos correctamente. – Se presiona el botón *Aceptar*.
EP 1.5: Actualizar un medio de comunicación introduciendo datos inválidos.	– Se selecciona el botón *Actualizar medio de comunicación*. – Se introducen los datos correspondientes, insertando algunos datos inválidos. – Se presiona el botón *Aceptar***.** – El sistema señala los datos introducidos de forma incorrecta. – El usuario corrige los datos.

	– Se presiona el botón *Aceptar***.**
EP 1.6: Actualizar un medio de comunicación dejando campos requeridos en blanco.	– Se selecciona el botón *Actualizar medio de comunicación*. – Se introducen los datos correspondientes dejándose al menos, uno de los campos requeridos en blanco. – Se presiona el botón *Aceptar* – El sistema señala los campos obligatorios que faltan por llenar. – El usuario llena los datos. – Se presiona el botón *Aceptar*.
EP 1.7: Actualizar una persona asociada correctamente.	– Se selecciona el botón *Actualizar personas asociadas*. – Se introducen los datos correctamente. – Se presiona el botón Aceptar.
EP 1.8: Actualizar una persona asociada introduciendo datos inválidos.	– Se selecciona el botón *Actualizar personas asociadas*. – Se introducen los datos correspondientes, insertando algunos datos inválidos. – Se presiona el botón *Aceptar***.** – El sistema señala los datos introducidos de forma incorrecta. – El usuario corrige los datos. – Se presiona el botón *Aceptar***.**
EP 1.9: Actualizar una persona asociada dejando campos requeridos en blanco.	– Se selecciona el botón *Actualizar personas asociadas*. – Se introducen los datos correspondientes dejándose al menos, uno de los campos requeridos en blanco. – Se presiona el botón *Aceptar* – El sistema señala los campos obligatorios que faltan por llenar. – El usuario llena los datos. – Se presiona el botón *Aceptar*.

EP 1.10: Cancelar actualización de un cliente jurídico.	– Se seleccionan los filtros de búsqueda y se llenan los datos para encontrar el cliente jurídico a actualizar. – Se presiona el botón *Buscar***.** – Se selecciona el cliente jurídico a actualizar en la tabla de resultados de la búsqueda y se presiona el botón *Actualizar cliente jurídico*. – Se selecciona el botón *Cancelar*. – Se muestra el mensaje: "¿Está seguro que desea abandonar esta operación?" – Si se acepta el mensaje concluye el requisito y no se realiza ninguna operación.

Tabla 7: Escenarios de Prueba del RF Actualizar Cliente Jurídico

No	Nombre de campo	Tipo	Válido	Inválido
1	Nombre	Campo de texto	Admite datos de tipo alfanumérico y caracteres	No puede ser nulo.
2	Tipo de entidad	Lista desplegable	Valores predeterminados. Se cargan los tipos de entidad existentes en el sistema, para la	Todos los valores que no estén en la lista predeterminada.
3	Código NIT	Campo de texto	Sólo admite datos de tipo numérico. No puede exceder de 11 caracteres.	No admite caracteres especiales, ni alfabéticos. Más de 11 caracteres.
4	Código ONE	Lista desplegable	Valores predeterminados. Se cargan los códigos ONE existentes en el sistema, para la	Todos los valores que no estén en la lista predeterminada.
5	Provincia	Lista desplegable	Valores predeterminados. Se cargan las provincias existentes en el sistema, para la selección	Todos los valores que no estén en la lista predeterminada.

6	Municipio	Lista desplegable	Valores predeterminados. Se cargan los municipios existentes en el sistema, para la selección	Todos los valores que no estén en la lista predeterminada.
7	Vinculación	Lista desplegable	Valores predeterminados. Se cargan los tipos de vinculación existentes en el sistema, para la	Todos los valores que no estén en la lista predeterminada.
8	Límite de confianza	Campo de texto	Sólo admite datos de tipo numérico.	No admite caracteres especiales, ni
9	No confiable	Caja de		
10	Código REEUP	Lista desplegable	Valores predeterminados. Se cargan los códigos REEUP existentes en el sistema, para la	Todos los valores que no estén en la lista predeterminada.
11	Dirección	Campo de texto	Admite datos de tipo alfanumérico y caracteres	No puede ser nulo.
12	Objeto social	Campo de texto	Sólo admite datos de tipo alfanumérico.	No puede ser nulo. No admite caracteres
13	Productos y servicios	Campo de texto	Sólo admite datos de tipo alfanumérico.	No admite caracteres especiales.
14	Principales clientes	Campo de texto	Sólo admite datos de tipo alfanumérico.	No admite caracteres especiales.
15	Tipos de medios de comunicación	Lista desplegable	Valores predeterminados. Se cargan los tipos de medios de comunicación existentes en el sistema, para la selección de uno de estos.	Todos los valores que no estén en la lista predeterminada.
16	Campos de los medios de comunicación	Campo de texto	Admite datos de tipo alfanumérico y caracteres especiales.	

17	País	Lista desplegable	Valores predeterminados. Se cargan los países existentes en el sistema, para la selección de uno de estos.	Todos los valores que no estén en la lista predeterminada. No puede ser nulo.
18	Tipo de identidad	Lista desplegable	Valores predeterminados. Se cargan los tipos de identidad existentes en el sistema, para la selección de uno de estos.	Todos los valores que no estén en la lista predeterminada. No puede ser nulo
19	No. de identidad personal	Campo de texto	Sólo admite datos de tipo alfanumérico.	No puede ser nulo. No admite caracteres

Tabla 8: Descripción de las variables de entrada

Id del escenario	EP 1.1	EP 1.2	EP 1.3	EP 1.4
Escenario	Registrar un cliente jurídico.	Registrar un cliente jurídico introduciendo datos inválidos.	Registrar un cliente jurídico dejando campos requeridos en blanco.	Cancelar el registro de un cliente jurídico.
Nombre	V(UCI)	I()	V(UCI)	V(UCI)
Tipo de entidad	V(Organismo estatal)	I(Orsmo 6352ftff)	V(Organismo estatal)	V(Organismo estatal)
Código NIT	V(12345678911)	I(365365356fffdghdyd)	V(12345678911)	V(12345678911)
Código ONE	V(00153-Instituto Nacional de Reservas Estatales)	I(454quehjgfGGG)	V(00153-Instituto Nacional de Reservas Estatales)	V(00153-Instituto Nacional de Reservas Estatales)
Provincia	V(La Habana)	I(LaaHbbb)	V(La Habana)	V(La Habana)
Municipio	V(Habana Vieja)	I(HabViejt)	V(Habana Vieja)	V(Habana Vieja)
Vinculación	V(Nacional)	I(nacTT)	V(Nacional)	V(Nacional)
Límite de	V(10)	I(tt43)	V(10)	V(10)

confianza				
Código REEUP	V(00153-Instituto Nacional de Reservas Estatales)	I(3399quehjgfGE)	V(00153-Instituto Nacional de Reservas Estatales)	V(00153-Instituto Nacional de Reservas Estatales)
Dirección	V(Carretera San Antonio, La Lisa)	I()	V(Carretera San Antonio, La Lisa)	V(Carretera San Antonio, La Lisa)
Objeto social	V(Venta de software)	I(&%#$#/)	V(Venta de software)	V(Venta de software)
Productos y servicios	V(Software y Soporte)	I(&%$%#)	V(Software y Soporte)	V(Software y Soporte)
Principales clientes	V(MINSAP, MINED)	I()	V(MINSAP, MINED)	V(MINSAP, MINED)
Tipos de medios de comunicación	V(Teléfono)	I(#terlefo)	V(Teléfono)	V(Teléfono)
Campos de los medios de comunicación	V(7656565)	I()	V(7656565)	V(7656565)
País de la persona asociada	V(Cuba)	I(Cub343)	V(Cuba)	V(Cuba)
Tipo de identidad	V(Carné de identidad)	I(Carn4535 de iden)	V(Carné de identidad)	V(Carné de identidad)
No. de identidad personal	V(87022508102)	I(87rser976673666666)	I()	V(87022508102)
Respuesta del sistema	El sistema registra al nuevo cliente jurídico y emite el mensaje de notificación: “Operación	El sistema señala los datos introducidos incorrectamente.	El sistema señala los campos de llenado obligatorio.	El sistema muestra el mensaje de alerta: “¿Está seguro que desea

	realizada satisfactoriamente.”.			abandonar esta operación?”.
Resultado de la prueba	Satisfactoria	Satisfactoria	Satisfactoria	Satisfactoria

Tabla 9: Juegos de datos a probar

Los resultados obtenidos a través de las pruebas realizadas al subsistema Monitoreo, fueron satisfactorios desde el punto de vista interno y funcional. El mismo se encuentra listo para pasar a las pruebas de Integración y Aceptación que serán aplicadas en el BNC, para posteriormente ser avalado a través del certificado de aceptación emitido por el cliente.

Pruebas de generación de alertas

Para validar el correcto funcionamiento del módulo Alerta, se realizaron una serie de pruebas sobre la base de datos, con el objetivo de verificar si los procedimientos almacenados que se implementaron para las alertas definidas, cumplen su función de manera adecuada. Se obtuvieron resultados satisfactorios en todas las pruebas, como la expuesta a continuación en el siguiente ejemplo.

Para probar la alerta que se genera al exceder los valores máximos o mínimos, de los créditos y débitos correspondientes a las operaciones de una cuenta, se tomó la tabla *M_CUEMAY_EXT:* tabla que recoge parte de los datos pertenecientes a las cuentas de clientes junto con la tabla *M_CUEMAY*. Se introdujeron datos de prueba, acotándose los rangos permisibles de los montos por operaciones.

Se escogió la cuenta de cliente 01 1210 2 0115000, fijándose el crédito por un mínimo de 2 300 y un máximo de 250 000, así como el débito con un mínimo de 1 000 y un máximo de 500 000.

COD_MON	CUE_SUB	TIP_CON	COD_CON	conciliable	fecha_ll	credito_min	credito_max	debito_min	debito_max	cod_usu
01	1210	2	0115000	☑	06/06/2013	2,300	250,000	1,000	500,000	00001
01	1210	2	0115001	☑	07/06/2013	0	0	0	0	00005
01	1210	2	0137000	☑	07/06/2013	0	0	0	0	00001
01	1210	2	0137002	☑	07/06/2013	0	0	0	0	00001
01	1210	2	2660005	☑	08/06/2013	0	0	0	0	00011
01	1210	2	3002000	☑	08/06/2013	0	0	0	0	00001
01	1212	2	2166000	☑	08/06/2013	0	0	0	0	00001
01	1220	2	2660000	☑	09/06/2013	0	0	0	0	00001
01	1270	2	0138000	☑	09/06/2013	0	0	0	0	00005
02	1210	2	0115002	☑	10/06/2013	0	0	0	0	00005
02	1210	2	0137000	☑	10/06/2013	0	0	0	0	00005
02	1210	2	0137001	☑	10/06/2013	0	0	0	0	00001

Ilustración 16: Datos de prueba de las cuentas de clientes

Según los datos de la tabla *H_HISTOR:* tabla que recoge todas las operaciones contables realizadas en las cuentas de clientes, la cuenta 01 1210 2 0115000 tenía registradas las siguientes operaciones.

imp_asient	fec_acthis	FEC_CONTAB	COD_MONEDA	CUE_SUBCUE	TIP_CONTRA	COD_CONTRA
500000	08/11/2005	08/11/2005	01	1210	2	0115000
-5607356.96	08/11/2005	08/11/2005	01	1210	2	0115000
-3316.96	08/11/2005	08/11/2005	01	1210	2	0115000
-2257.48	08/11/2005	08/11/2005	01	1210	2	0115000
-28317.51	08/11/2005	08/11/2005	01	1210	2	0115000
-21182.52	08/11/2005	08/11/2005	01	1210	2	0115000
-470200	08/11/2005	08/11/2005	01	1210	2	0115000
-173850	08/11/2005	08/11/2005	01	1210	2	0115000
6700000	08/11/2005	08/11/2005	01	1210	2	0115000
666.28	08/11/2005	08/11/2005	01	1210	2	0115000
255482.1	08/11/2005	08/11/2005	01	1210	2	0115000
200000	08/11/2005	08/11/2005	01	1210	2	0115000
-24516	08/11/2005	08/11/2005	01	1210	2	0115000
-2700	08/11/2005	08/11/2005	01	1210	2	0115000
-4617.85	08/11/2005	08/11/2005	01	1210	2	0115000
-9100000	08/11/2005	08/11/2005	01	1210	2	0115000
-244496.84	08/11/2005	08/11/2005	01	1210	2	0115000
-70718.96	08/11/2005	08/11/2005	01	1210	2	0115000

Ilustración 17: Operaciones contables de la cuenta 01 1210 2 0115000

Como se puede apreciar en la ilustración anterior, se muestran los importes de cada operación ejecutada en el campo *imp_asient*, registrándose valores negativos para los créditos y positivos para los débitos.

Para detectar las operaciones con un importe fuera del rango establecido, se ejecutó el procedimiento almacenado llamado *alertaMontos*, devolviéndose los siguientes datos.

imp_asie	fec_acthis	FEC_CONTAB	COD_MONEDA	CUE_SUBCUE	TIP_CONTRA	COD_CONTRA
-5607356.96	08/11/2005	08/11/2005	01	1210	2	0115000
-2257.48	08/11/2005	08/11/2005	01	1210	2	0115000
-470200	08/11/2005	08/11/2005	01	1210	2	0115000
6700000	08/11/2005	08/11/2005	01	1210	2	0115000
666.28	08/11/2005	08/11/2005	01	1210	2	0115000
-9100000	08/11/2005	08/11/2005	01	1210	2	0115000

Ilustración 18: Resultado del procedimiento almacenado alertaMontos

A partir de los resultados obtenidos de la prueba realizada se pudo comprobar, que el procedimiento detectó exitosamente todas las operaciones con un importe fuera de la norma fijada.

3.4 Conclusiones parciales

En este capítulo se lograron desarrollar las funcionalidades propuestas para darle solución a la problemática existente en el BNC. La validación del subsistema estuvo dada por las pruebas de unidad realizadas por el equipo del proyecto. Para llevar a cabo las pruebas de caja blanca se utilizó el *framework JUnit*, el cual facilitó su ejecución. Las pruebas de caja negra se realizaron manualmente apoyadas en los casos de pruebas descritos. Durante las pruebas aplicadas se recogieron una serie de no conformidades que fueron analizadas y resueltas en su totalidad.

CONCLUSIONES GENERALES

Al término del presente trabajo de diploma, se concluye que se desarrollaron todas las tareas propuestas, a fin de cumplir los objetivos establecidos y para ello:

- Se examinaron las características de diferentes sistemas informáticos, tanto nacionales como internacionales vinculados al monitoreo y control, donde la principal dificultad para su utilización resultó ser su poca adaptabilidad al negocio del BNC, evidenciándose la necesidad de desarrollar como solución, un subsistema para el monitoreo incorporado al sistema Quarxo.
- Se realizó el análisis, diseño e implementación del subsistema Monitoreo a partir de los procesos y requisitos identificados, lográndose monitorear las operaciones sobre las cuentas de clientes del BNC y brindar solución a las deficiencias del sistema en explotación.
- Se evaluó la viabilidad del subsistema Monitoreo, a través de pruebas de software efectuadas al nivel de unidad, las cuales arrojaron resultados favorables, demostrándose el cumplimiento de las funcionalidades previstas para el mismo y de esta forma acatar el objetivo general planteado.

La propuesta presenta un valor técnico, destacándose la asociación de principios por los que se mide la factibilidad de un diseño de software, como lo es el empleo de patrones. Esto posibilita la reutilización, sostenibilidad y mantenimiento del sistema. La solución es además novedosa, su importancia radica en la realización de los procesos referentes al monitoreo en el BNC, a través de subsistemas que funcionan rápido y correctamente, contribuyéndose al ahorro de tiempo y recursos, así como al control eficiente de los cambios y la información manipulada.

RECOMENDACIONES

- Aprovechar los estudios realizados en esta investigación, para mejorar la solución presentada en futuras versiones que se realicen del sistema.
- Incorporar nuevas alertas al módulo Alerta del subsistema Monitoreo de Operaciones en Cuentas de Clientes, que contribuyan a la seguridad del BNC.
- Incrementar el uso de las tecnologías y herramientas presentadas en la investigación, para el desarrollo de otros sistemas de gestión bancaria.

REFERENCIAS BIBLIOGRÁFICAS

1. **SSI.** Subdirección de Seguridad de la Información. *Información y servicios de seguridad en cómputo.* [En línea] 7 de Septiembre de 2011. [Citado el: 5 de Noviembre de 2012.] http://www.seguridad.unam.mx/noticias/?noti=4848.

2. **OLAF.** Comisión Europea. *Oficina Europea de Lucha contra el Fraude.* [En línea] 5 de Noviembre de 2012. [Citado el: 5 de Noviembre de 2012.] http://ec.europa.eu/anti_fraud/index_es.htm.

3. **BCC.** Banco Central de Cuba. [En línea] 2006. [Citado el: 5 de Noviembre de 2012.] http://www.bc.gov.cu/Manual/Cap%C3%ADtulo%2006.%20Regulaciones%20de%20Supervisi%C3%B3n%20Bancaria/6.07%20%20Instrucci%C3%B3n%20No.%2026%20de%2013%20Nov.%202006.%20GUIA%20PARA%20PREVENIR%20LAS%20OPERACIONES%20DE%20LAVADO%20DE%20DINERO.pdf.

4. **Auditool.** Auditool. *Red de conocimientos en Auditoría y Control interno.* [Online] 2012. [Cited: Noviembre 7, 2012.] http://www.auditool.org/index.php?option=com_content&view=article&id=682:auditoria-de-monitoreo-continuo-de-las-operaciones&catid=38:restaurants&Itemid=57.

5. **Eniac.** Eniac. [Online] 2012. [Cited: Noviembre 7, 2012.] http://www.eniac.com/productos/acl.htm.

6. **Catálogo de Software.com.** Catálogo de Software.com. *Portal Especializado de Software y Servicios Relacionados para el Sector Empresarial.* [Online] Junio 21, 2012. [Cited: Noviembre 7, 2012.] http://www.catalogodesoftware.com/producto-bideab-1152.

7. **EcuRed.** EcuRed. *Conocimiento con todos y para todos.* [Online] Noviembre 2012. [Cited: Noviembre 7, 2012.] http://www.ecured.cu/index.php/Versat_Sarasola.

8. **Rodas XXI.** Rodas XXI. *Sistema Integral Económico Administrativo.* [Online] Noviembre 2012. [Cited: Noviembre 7, 2012.] http://www.rodasxxi.cu/rodasxxi.php.

9. **The GBM Journal.** Busisness Transformation - The GBM Journal. [Online] Mayo 2006. [Cited: Noviembre 7, 2012.] http://www.gbm.net/bt/bt32/hss/ibm_anuncia_soluciones_blade_center.php.

10. **Obregón, Ing. William González.** *Modelo de desarrollo de software.* 1.0. 2012. p. 109.

11. **Larman, C.** *UML y Patrones.* Segunda Edición. s.l. : Prentice Hall, 1999.

12. **Visual Paradigm.** Visual Paradigm. *UML CASE tool for software development.* [Online] Noviembre 2012. [Cited: Noviembre 8, 2012.] http://www.visual-paradigm.com/product/vpuml/.

13. **I. Architecture Working Group.** *IEEE Recomended Practice for Architectural Description of Software-Intensive Systems.* IEEE Standard IEEE STD 1471-2000. s.l. : IEEE Standards Association, 2000.

14. **Marquina, E.** *Guía de Patrones, Prácticas y Arquitecturas .NET.* s.l. : Organización Contoso, 2008.

15. **Oracle.** Oracle Technology Network for Java Developers. [Online] 2009. [Cited: Noviembre 8, 2012.] http://www.oracle.com/technetwork/java/dataaccessobject-138824.html.
16. **Jaworski, J.** *Java Developer's Guide.* s.l. : Sams.net, 1996. Vol. 201.
17. **Astarita, E.** Manual XHTML. [Online] 2006. [Cited: Noviembre 8, 2012.] http://manual-xhtml.blogspot.com/2006/05/primer-documento-xhtml.html.
18. **Global Mentoring.** Global Mentoring. *Experiencia y conocimiento para tu vida.* [Online] Mayo 6, 2011. [Cited: Noviembre 9, 2012.] http://www.globalmentoring.com.mx/portal/es/blog/item/que-es-un-framework.html.
19. **Spring Source Community.** SpringSource.org. [Online] 2012. [Cited: Noviembre 9, 2012.] http://www.springsource.org/.
20. **EcuRed.** EcuRed. *Conocimiento con todos y para todos.* [Online] 2010. [Cited: Noviembre 9, 2012.] http://www.ecured.cu/index.php/Hibernate.
21. **Rusell, M.** *Dojo: The definitive guide.* s.l. : O'Reilly Media, Inc, 2008.
22. **Geer, D.** *Eclipse becomes the dominant Java IDE.* s.l. : Computer, 2005. pp. 16-18.
23. **V. Chopra, S. Li, y J. Genender.** *Professional Apache Tomcat 6.* s.l. : Wrox, 2011.
24. **Elster, J.** *Sour grapes: Studies in the subversion of rationality.* s.l. : Cambridge Univ Pr, 1985.
25. **Delaney, K.** *Inside microsoft® sql server™ 2005: query tuning and optimization.* s.l. : Microsoft Press, 2007.
26. **Pressman, Roger S.** *Ingeniería del software. Un enfoque practico. Cuarta edición.* 1998.
27. **Ivar Jacobson, Grady Booch, James Rumbaugh.** *El Proceso Unificado de Desarrollo de Software.* s.l. : Pearson Educación , 2000.
28. **Gutiérrez, Damián.** *UML Diagramas de Paquetes.* s.l. : Universidad de los Andes, 2009.
29. **Vidal, Mari Carmen Otero.** *Diagramas de Clases .* s.l. : Dpto. de Lenguajes y Sistemas Informáticos, ITESCAM.
30. **Dirección General de Gobierno Digital.** *Estándares de codificaciones de sistemas .* s.l. : Ministerio de Coordinación de Gabinete, Gobierno del Chubut , 2006.
31. **Universidad de Castilla-La Mancha. Escuela Politécnica Superior de Albacete.** *Ingeniería de Software.* s.l. : [En línea] http://www.dsi.uclm.es/asignaturas/42530/pdf/M2tema12.pdf, 2011.
32. **Thames, Juan Pablo Bustos.** *Verificación y Validación del Diseño .* 2011.

BIBLIOGRAFÍA CONSULTADA

Design Patterns. Addison Wesley, 1995. Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides.

Dojo: The Definitive Guide. O'Reilly Media Inc., 2008. Matthew A. Russell.

El Proceso Unificado De Desarrollo De Software. PERSON EDUCACIÓN.S.A. 2000. Ivan Jacobson, Grady Booch, James Rumbaugh.

INGENIERÍA DEL SOFTWARE: **Un enfoque práctico**. McGraw-Hill Interamericana, 2002. Roger S. Pressman.

JUnit in Action. Manning Publications Co, 2010. Peter Tahchiev, Felipe Leme, Vincent Massol y Gary Gregory.

UML y Patrones. Prentice Hall, 1999. Craig Larman.

Control de Versiones Usando SubVersion. Galíndez, Rodrigo. [En línea] http://www.rodrigogalindez.com/files/14.pdf.

Desarrollo Web. Arquitectura cliente-servidor. [En línea] http://www.desarrolloweb.com/articulos/arquitectura-cliente-servidor.html.

Debug Mode On. El patrón MVC. [En línea] http://es.debugmodeon.com/articulo/el-patron-mvc.

Introducción a JavaScript. 2008. Pérez, Javier Eguíluz.

Apache: Documentación del Servidor HTTP Apache 2.0. [En línea] http://httpd.apache.org/docs/2.0/es/.

Ingeniería de Software: Un Enfoque Práctico. 1998. Pressman, Roger.

Mastering Dojo. Pragmatic Bookshelf, 2008. Rawld Gill, Craig Riecke, Alex Russell.

Spring in Action. Manning Publications Co, 2005. Craig Walls, Ryan Breindenbach.

Spring in Action: Second Edition. Manning Publications Co, 2008. Craig Walls, Ryan Breidenbach.

Hibernate in Action. Manning Publications Co, 2005. Christian Bauer, Gavin King.

Java Servlet Programing. O'Reilly & Associates, 1998. Jason Hunter, Willian Crawford.

Pruebas de Software. [En línea] http://lsi.ugr.es/~ig1/docis/pruso.pdf.

BIZAGI. Bizagi Process Modeler. [En línea] http://www.bizagi.com/esp/descargas/BPMNbyExample.pdf.

GLOSARIO DE TÉRMINOS

A

Algoritmo: Cualquier procedimiento computacional bien definido mediante un conjunto de reglas, que da solución a instancias de un problema (entrada) produciéndose un valor o conjunto de valores (salida).

API: Interfaz de Programación de Aplicaciones o *API* (*Application Programming Interface*). Es el conjunto de funciones, procedimientos o métodos, en la programación orientada a objetos que ofrece cierta librería para ser utilizado por otro software como una capa de abstracción.

B

Beans: Son componentes hechos en software que se pueden reutilizar y que pueden ser manipulados visualmente por una herramienta de programación en lenguaje *Java*.

C

CASE: Siglas en inglés: *Computer Aided Software Engineering*, en español: Ingeniería de Software Asistida por Computadora. Son diversas aplicaciones informáticas destinadas a aumentar la productividad en el desarrollo de software reduciéndose el costo de las mismas en términos de tiempo y dinero.

Código abierto: en inglés: *Open Source*, es el término con el que se conoce al software distribuido y desarrollado libremente. El código abierto tiene un punto de vista más orientado a los beneficios prácticos de compartir el código.

E

Entidades: 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.

F

Framework: Es una estructura de soporte definida, mediante la cual otro proyecto de software puede ser organizado y desarrollado. Típicamente, puede incluir soporte de programas, bibliotecas y un

lenguaje interpretado entre otros software para ayudar a desarrollar y unir los diferentes componentes de un proyecto.

H

HTML: Lenguaje para la elaboración de páginas web, es usado para describir la estructura y el contenido en forma de texto, así como para complementar el texto con objetos tales como imágenes.

I

IDE: Entorno de desarrollo integrado (en inglés *Integrated Development Environment*). Es un programa informático compuesto por un conjunto de herramientas de programación.

J

JavaBeans: Son un modelo de componentes creado por *Sun Microsystems* para la construcción de aplicaciones en *Java*.

JavaScript: Es un lenguaje de programación interpretado. Se define como orientado a objetos, basado en prototipos, imperativo, débilmente tipado y dinámico. Se utiliza principalmente en su forma del lado del cliente, implementado como parte de un navegador web para permitir mejoras en la interfaz de usuario y páginas web dinámicas.

JDBC: (en inglés: *Java Data Base Connectivity)*. Permite la ejecución de operaciones sobre bases de datos desde el lenguaje de programación *Java*.

JSP: Es una tecnología *Java* para crear contenido dinámico para *web* en forma de documentos *HTML* o *XML*.

M

Mapear: Hace referencia a la acción de realizar un mapeo, (del inglés *mapping*), que significa hacer corresponder en elemento con otro en el sistema.

Multiplataforma: Es un término utilizado frecuentemente en informática para indicar la capacidad, o características de poder funcionar o mantener una interoperabilidad de forma similar en diferentes sistemas operativos o plataformas.

N

Navegador: Software que permite al usuario recuperar y visualizar documentos de hipertexto desde servidores web a través de internet

P

Plugin: También conocido como (*plug-in*, *add-in*, *addon*, *add-on*, *snap-in* o *snapin*), es un pequeño programa de computadora que extiende las capacidades de otro programa de mayor tamaño, adicionándole nuevas funcionalidades.

S

Script: Conjunto de instrucciones escritas en un lenguaje *script* ejecutadas por un intérprete de comandos.

Servlet: Es utilizado para generar páginas web de forma dinámica, a partir de parámetros de una petición de un navegador web, utilizando para este objetivo el acceso a bases de datos, flujos de trabajo y otros recursos.

X

XML: Es un metalenguaje extensible de etiquetas desarrollado por el *World Wide Web Consortium,* el cual permite definir la gramática de lenguajes específicos. *XML* no es realmente un lenguaje en particular, sino una manera de definir lenguajes para diferentes necesidades.

XHTML: Siglas en inglés: *eXtensible HyperText Markup Language*. Es básicamente HTML expresado como XML válido. Es más estricto a nivel técnico, pero esto permite que posteriormente sea más fácil al hacer cambios o buscar errores entre otros.

ANEXOS

Anexo 1: Especificación de requisito Buscar Expediente.

Precondiciones	El usuario se ha autenticado ante el sistema y tiene permisos para ejecutar esta acción.
Flujo de eventos	
Flujo básico Buscar expediente.	
5.	Se eligen los criterios de búsqueda.
6.	Se insertan los datos de la búsqueda.
7.	Se seleccionan las columnas que desea ver en el resultado de la búsqueda.
8.	Se selecciona el botón “Buscar”.
9.	El sistema muestra los expedientes encontrados de la búsqueda.
10.	Concluye el requisito.
Pos-condiciones	
1	Se muestran los datos de los expedientes encontrados.
Flujos alternativos	
Flujo alternativo 4.a No se encuentran datos	
1	El sistema muestra un mensaje alertando que no se encontró ningún expediente asociado a los parámetros de búsqueda.
Pos-condiciones	
1	N/A
Flujo alternativo 4.a Información errónea	
1	El sistema señala los datos erróneos.
2	El usuario corrige los datos.
3	Volver al paso 4 del flujo básico.
Pos-condiciones	
1	N/A.
Flujo alternativo 4.b Información incompleta	
1	El sistema señala los datos vacíos.
2	El usuario llena los datos.
3	Volver al paso 4 del flujo básico.
Pos-condiciones	
1	N/A.

Validaciones		
N/A.		
Relaciones	**Requisitos Incluidos**	
	Extensiones	N/A.
Conceptos	N/A	
Requisitos especiales	N/A.	
Asuntos pendientes	N/A.	

Anexo 2: Especificación de requisito Registrar Cliente Jurídico

Precondiciones	El usuario se ha autenticado ante el sistema y tiene permisos para ejecutar esta acción. Debe estar registrada al menos una persona natural.
Flujo de eventos	
Flujo básico Registrar cliente jurídico	
11. Se introducen los datos: Nombre Tipo de entidad Código NIT Código ONE Provincia Municipio Vinculación Límite de confianza No confiable Código REEUP Dirección Objeto social Productos y servicios Principales clientes Tipo de medio de comunicación Campo del medio de comunicación	

	Cargo de la persona asociada Fecha del nombramiento del cargo de la persona asociada Categoría de la persona asociada Relación familiar con el jefe inmediato
12.	El usuario solicita aceptar.
13.	El sistema valida los datos introducidos.
14.	Si los datos son correctos el sistema los registra.
15.	El sistema confirma el registro de los datos y muestra el mensaje “Operación realizada satisfactoriamente”.
16.	Concluye el requisito.
Pos-condiciones	
2.	Se registran los datos del cliente jurídico en la base de datos.
Flujos alternativos	
Flujo alternativo 3.a Información errónea	
4	El sistema señala los datos erróneos y permite corregirlos.
5	El usuario corrige los datos.
6	Volver al paso 2 del flujo básico.
Pos-condiciones	
2	N/A.
Flujo alternativo 3.b Información incompleta	
4	El sistema señala los datos vacíos y permite completarlos.
5	El usuario completa los datos.
6	Volver al paso 2 del flujo básico.
Pos-condiciones	
2	N/A.
Flujo alternativo 2.a El usuario cancela la acción	
1	El sistema muestra el mensaje ¿Estás seguro que desea cancelar la operación?
2	El usuario acepta el mensaje.
3	Concluye el requisito.
Pos-condiciones	
1	No se registran los datos.
Validaciones	

1		
2		
Relaciones	**Requisitos Incluidos**	N/A.
	Extensiones	N/A.
Conceptos	N/A	
Requisitos especiales	N/A.	
Asuntos pendientes	N/A.	

Anexo 3: Caso de Prueba del requisito Registrar Cliente Jurídico

Nombre del requisito	**Descripción general**	**Escenarios de pruebas**	**Flujo del escenario**
1: Registrar cliente jurídico.	El sistema debe permitir registrar un cliente jurídico, para ello se deben llenar los campos correspondientes a su definición, de cancelarse esta operación, no se efectuará el registro del cliente jurídico.	EP 1.1: Registrar un cliente jurídico correctamente.	– Se introducen los datos correctamente. – Se presiona el botón *Aceptar*. – Se muestra el mensaje emitido por el sistema con el texto: “Operación realizada satisfactoriamente.”. – Se presiona el botón *Aceptar* del mensaje**.**
		EP 1.2: Registrar un cliente jurídico introduciendo datos inválidos.	– Se introducen los datos correspondientes, insertando algunos

		datos inválidos. – Se presiona el botón *Aceptar***.** – El sistema señala los datos introducidos de forma incorrecta. – El usuario corrige los datos. – Se presiona el botón *Aceptar***.**
	EP 1.3: Registrar un cliente jurídico dejando campos requeridos en blanco.	– Se introducen los datos correspondientes, dejándose al menos uno de los campos requeridos en blanco. – Se presiona el botón *Aceptar* – El sistema señala los campos obligatorios que faltan por llenar. – El usuario llena los datos. – Se presiona el botón *Aceptar*.
	EP 1.4: Registrar un medio de comunicación correctamente.	– Se selecciona el botón *Registrar medio de comunicación.* – Se introducen los datos correctamente. – Se presiona el botón

		Aceptar.
	EP 1.5: Registrar un medio de comunicación introduciendo datos inválidos.	– Se selecciona el botón *Registrar medio de comunicación*. – Se introducen los datos correspondientes, insertando algunos datos inválidos. – Se presiona el botón *Aceptar***.** – El sistema señala los datos introducidos de forma incorrecta. – El usuario corrige los datos. – Se presiona el botón *Aceptar***.**
	EP 1.6: Registrar un medio de comunicación dejando campos requeridos en blanco.	– Se selecciona el botón *Registrar medio de comunicación*. – Se introducen los datos correspondientes dejándose al menos, uno de los campos requeridos en blanco. – Se presiona el botón *Aceptar* – El sistema señala los campos obligatorios

		que faltan por llenar. – El usuario llena los datos. – Se presiona el botón *Aceptar*.
	EP 1.7: Registrar una persona asociada correctamente.	– Se selecciona el botón *Registrar personas asociadas*. – Se introducen los datos correctamente. – Se presiona el botón *Aceptar*.
	EP 1.8: Registrar una persona asociada introduciendo datos inválidos.	– Se selecciona el botón *Registrar personas asociadas*. – Se introducen los datos correspondientes, insertando algunos datos inválidos. – Se presiona el botón *Aceptar***.** – El sistema señala los datos introducidos de forma incorrecta. – El usuario corrige los datos. – Se presiona el botón *Aceptar***.**
	EP 1.9: Registrar una persona asociada dejando campos requeridos en blanco.	– Se selecciona el botón *Registrar personas asociadas*. – Se introducen los

		datos correspondientes dejándose al menos, uno de los campos requeridos en blanco. – Se presiona el botón *Aceptar* – El sistema señala los campos obligatorios que faltan por llenar. – El usuario llena los datos. – Se presiona el botón *Aceptar*.
	EP 1.10: Cancelar el registro de un cliente jurídico.	– Se selecciona el botón *Cancelar*. – Se muestra el mensaje: "¿Está seguro que desea abandonar esta operación?" – Si se acepta el mensaje concluye el requisito y no se realiza ninguna operación.

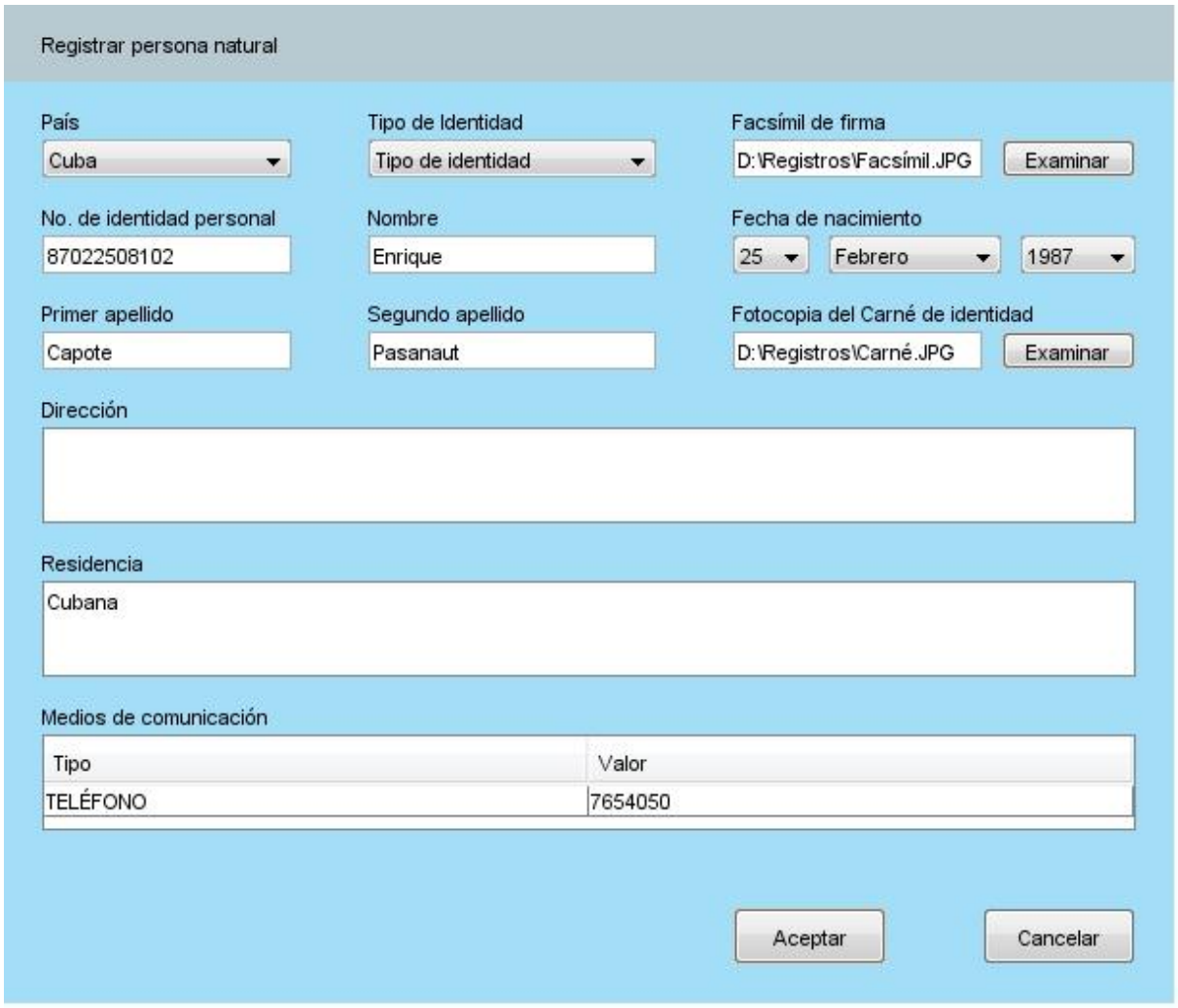
Anexo 4: Juego de datos a probar para Registrar Cliente Jurídico

Id del escenario	EP 1.1	EP 1.2	EP 1.3	EP 1.4
Escenario	Registrar un cliente jurídico.	Registrar un cliente jurídico introduciendo datos inválidos.	Registrar un cliente jurídico dejando campos	Cancelar el registro de un cliente jurídico.

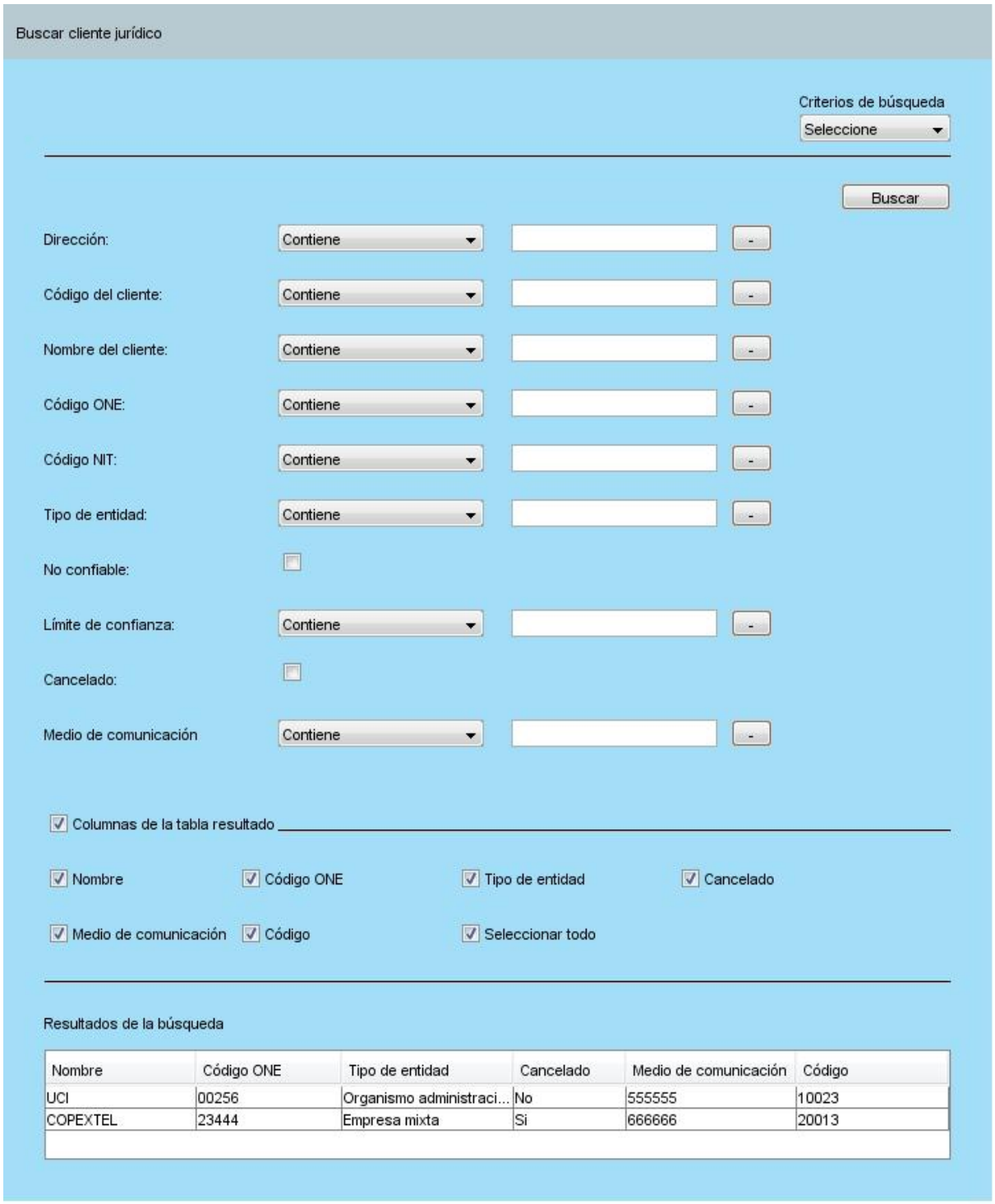
			requeridos en blanco.	
Nombre	V(UCI)	I()	V(UCI)	V(UCI)
Tipo de entidad	V(Organismo estatal)	I(Orsmo 6352ftff)	V(Organismo estatal)	V(Organismo estatal)
Código NIT	V(12345678911)	I(365365356fffdghdyd)	V(12345678911)	V(12345678911)
Código ONE	V(00153-Instituto Nacional de Reservas Estatales)	I(454quehjgfGGG)	V(00153-Instituto Nacional de Reservas Estatales)	V(00153-Instituto Nacional de Reservas Estatales)
Provincia	V(La Habana)	I(LaaHbbb)	V(La Habana)	V(La Habana)
Municipio	V(Habana Vieja)	I(HabViejt)	V(Habana Vieja)	V(Habana Vieja)
Vinculación	V(Nacional)	I(nacTT)	V(Nacional)	V(Nacional)
Límite de confianza	V(10)	I(tt43)	V(10)	V(10)
Código REEUP	V(00153-Instituto Nacional de Reservas Estatales)	I(3399quehjgfGE)	V(00153-Instituto Nacional de Reservas Estatales)	V(00153-Instituto Nacional de Reservas Estatales)
Dirección	V(Carretera San Antonio, La Lisa)	I()	V(Carretera San Antonio, La Lisa)	V(Carretera San Antonio, La Lisa)
Objeto social	V(Venta de software)	I(&%#$#/)	V(Venta de software)	V(Venta de software)
Productos y servicios	V(Software y Soporte)	I(&%$%#)	V(Software y Soporte)	V(Software y Soporte)
Principales clientes	V(MINSAP, MINED)	I()	V(MINSAP, MINED)	V(MINSAP, MINED)
Tipos de medios de comunicación	V(Teléfono)	I(#terlefo)	V(Teléfono)	V(Teléfono)

Campos de los medios de comunicación	V(7656565)	I()	V(7656565)	V(7656565)
País de la persona asociada	V(Cuba)	I(Cub343)	V(Cuba)	V(Cuba)
Tipo de identidad	V(Carné de identidad)	I(Carn4535 de iden)	V(Carné de identidad)	V(Carné de identidad)
No. de identidad personal	V(87022508102)	I(87rser976673666666)	I()	V(87022508102)
Respuesta del sistema	El sistema registra al nuevo cliente jurídico y emite el mensaje de notificación: "Operación realizada satisfactoriamente.".	El sistema señala los datos introducidos incorrectamente.	El sistema señala los campos de llenado obligatorio.	El sistema muestra el mensaje de alerta: "¿Está seguro que desea abandonar esta operación?".

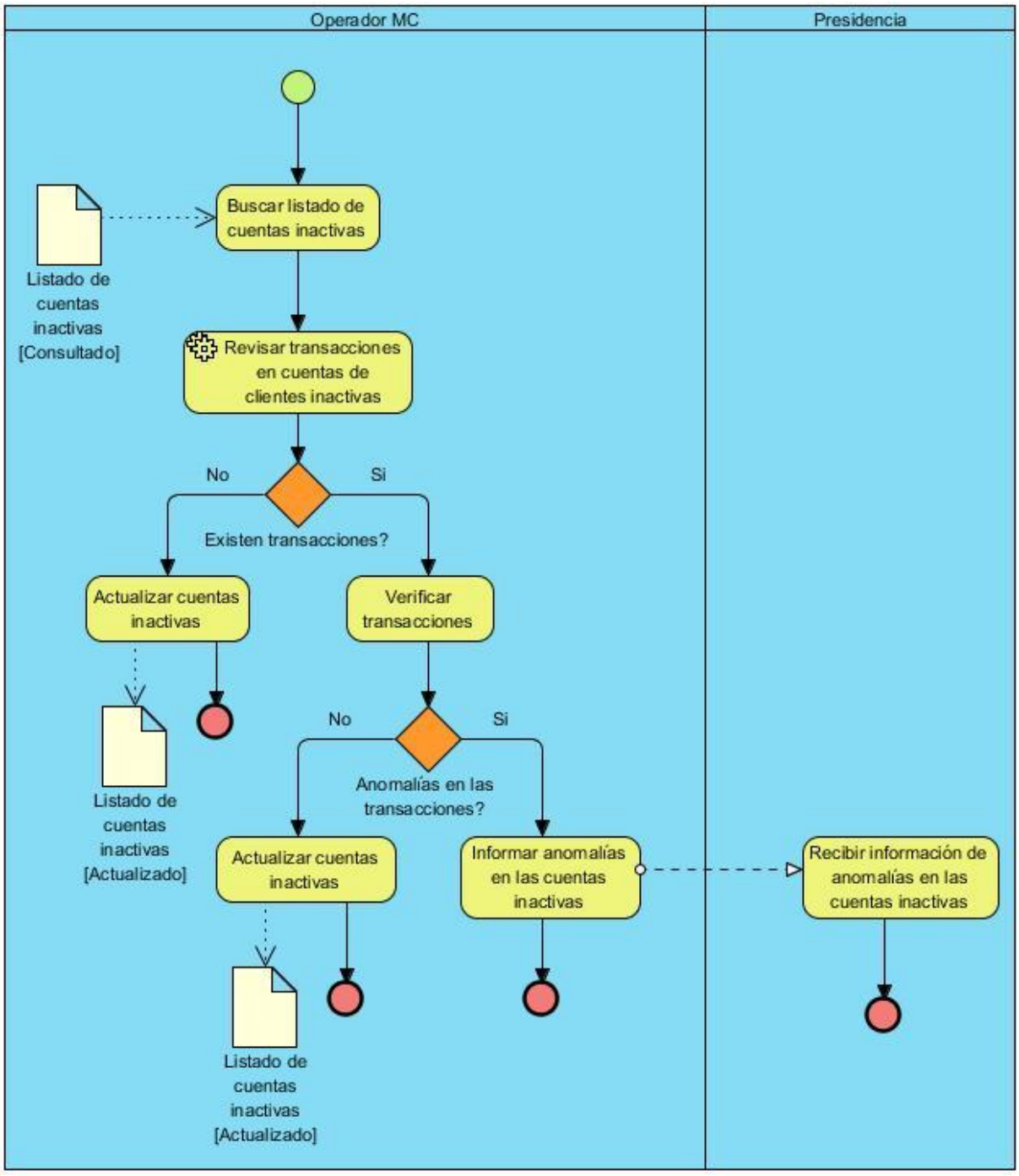
Anexo 5: Prototipo de diseño Registrar Cliente Jurídico



Anexo 6: Prototipo de Diseño Buscar Cliente Jurídico



Anexo 7: Diagrama de Proceso Monitorear Cuentas Inactivas



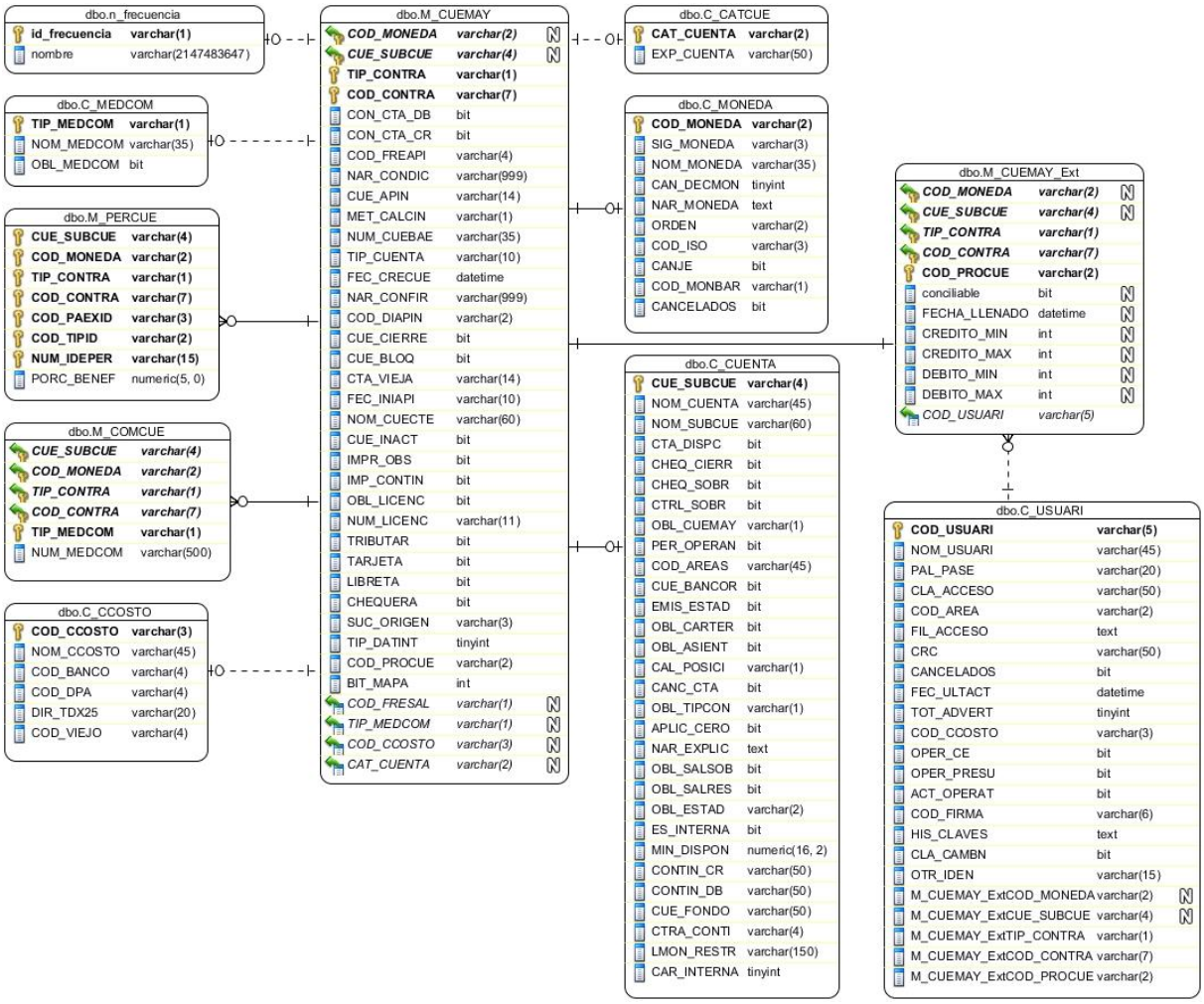
Anexo 8: Reglas de Negocio Monitoreo de Expedientes

No	Tipo	Descripción

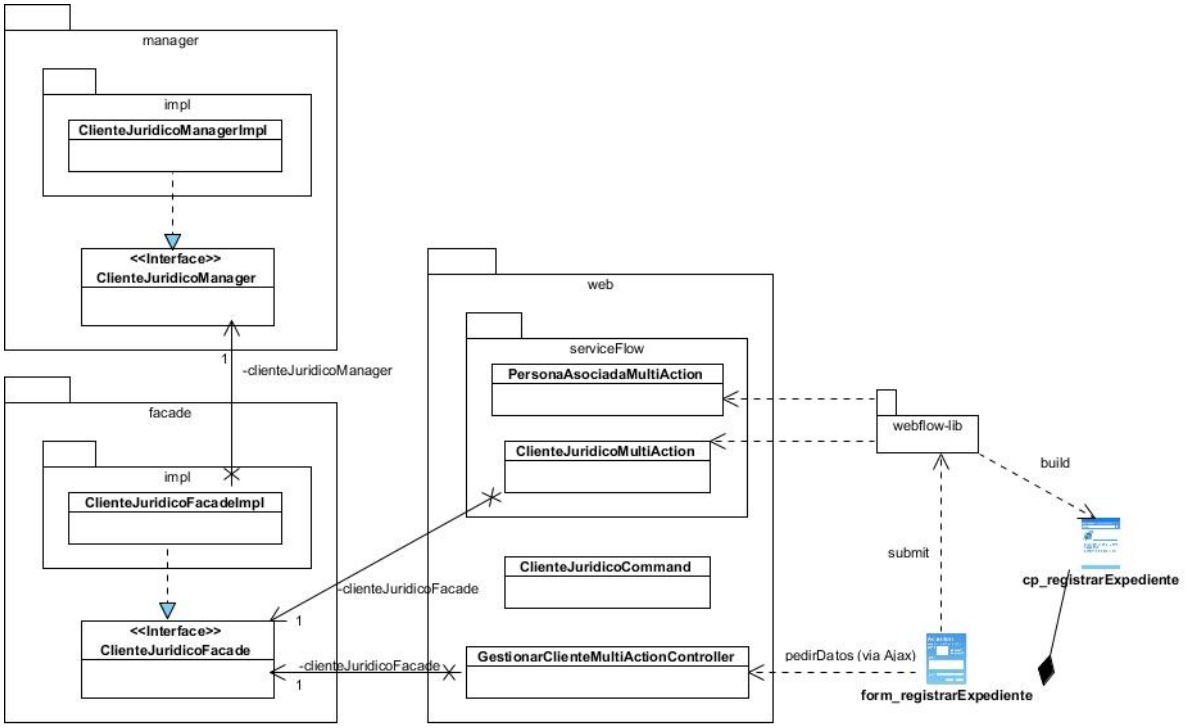
1	**Reglas Textuales**	Para la creación de una cuenta, el cliente deberá mostrar una licencia de BCC a la DGN (si es la primera cuenta), si tiene creada alguna cuenta con licencia, llega directo a cuentas corrientes (la Resolución 76 del 2000 abarca la apertura de cuentas).
2	**Reglas de Relación**	Para la creación de las cuentas 3241 y 3210 es necesaria una licencia del BCC.
3	**Reglas Textuales**	Cada expediente es chequeado por la Operadora MC al menos una vez al año.
4	**Reglas de Relación**	Por cada cuenta que se crea con códigos 3241 y 3210 se define un expediente con toda la información necesaria.
5	**Reglas Textuales**	A las cuentas con código 3221 solo se les creará un expediente si se conoce la necesidad de que el representante deba firmar en lo adelante. Ejemplo, si antes de la emisión de una CC se recibe el dinero del cliente, se crea una 3221 para el monto y como se tiene de antemano todo el dinero, el BNC no necesita crear un expediente para esa cuenta; por el contrario, si

		llega solo parte del dinero antes de la emisión y se conoce que en lo adelante se deberá exigir la otra parte al cliente, entonces sí se crea un expediente para posibilitar un seguimiento
6	**Reglas Textuales**	En los casos relacionados con la creación de cuentas con código 3221 para Cuenta Única no se requiere la creación del expediente.

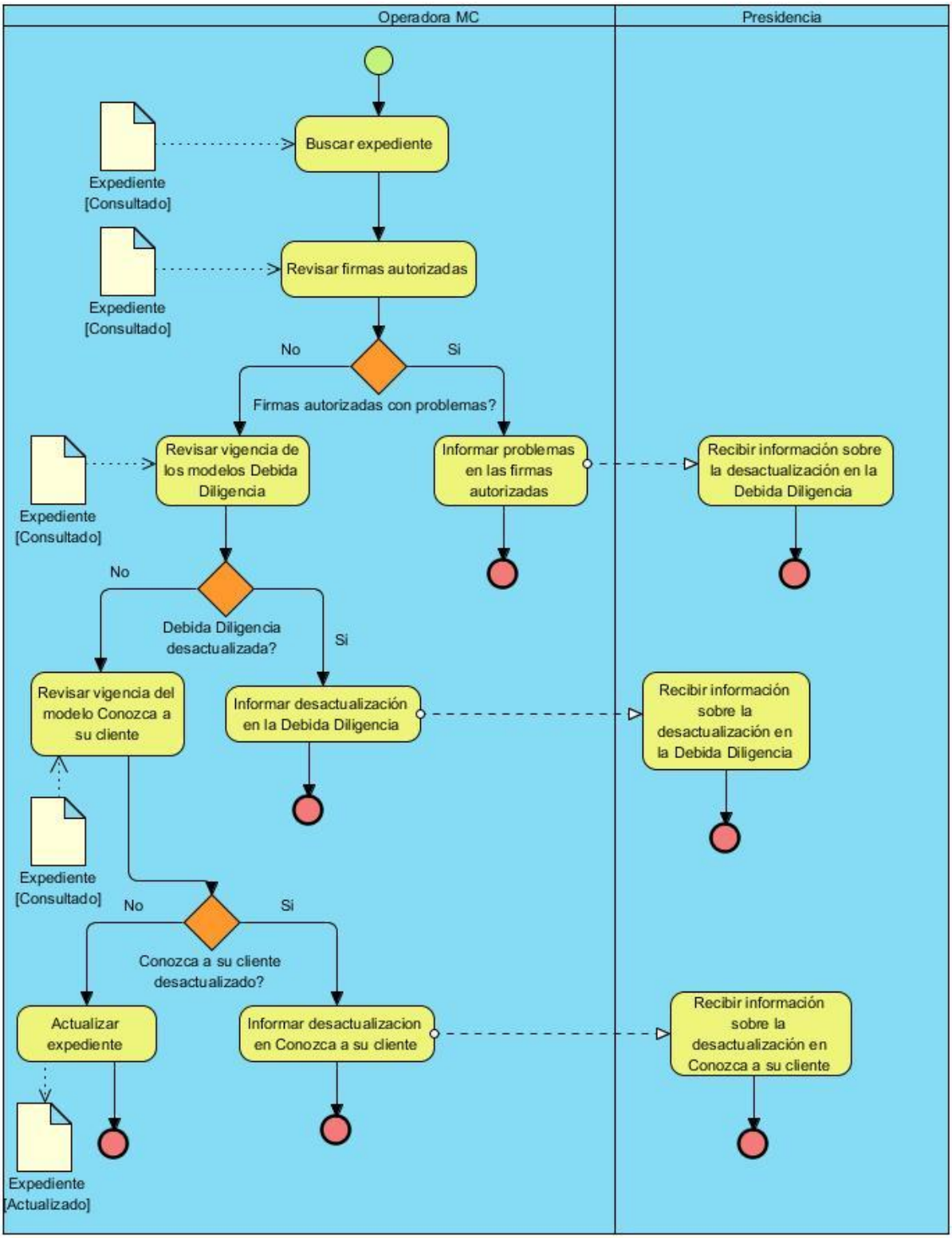
Anexo 9: Modelo de Datos de Cuentas Clientes



Anexo 10: Diagrama de Clases Gestionar Cliente Jurídico



Anexo 11: Diagrama de proceso de negocio Monitoreo de expedientes.


Operadora MC
Presidencia
Buscar expediente
Expediente
[Consultado]
Revisar firmas autorizadas
Expediente
[Consultado]
No
Si
Firmas autorizadas con problemas?
Revisar vigencia de los modelos Debida Diligencia
Expediente
[Consultado]
Informar problemas en las firmas autorizadas
Recibir información sobre la desactualización en la Debida Diligencia
No
Debida Diligencia desactualizada?
Si
Revisar vigencia del modelo Conozca a su cliente
Informar desactualización en la Debida Diligencia
Recibir información sobre la desactualización en la Debida Diligencia
Expediente
[Consultado]
No
Si
Conozca a su cliente desactualizado?
Actualizar expediente
Informar desactualizacion en Conozca a su cliente
Recibir información sobre la desactualización en Conozca a su cliente
Expediente
[Actualizado]