Universidad de las Ciencias Informáticas

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

"Subsistema Mensajería SWIFT Fase II"

Quarxo

Autores: *Yadelis Trujillo Martínez*

Radamés Fernández López

Tutor: *Ing. Alain Lazaro Piñero Añon*

La Habana, 2013

DECLARACIÓN DE AUTORÍA

Declaramos ser autores de la presente tesis y reconocemos a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

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

Yadelis Trujillo Martínez

_____

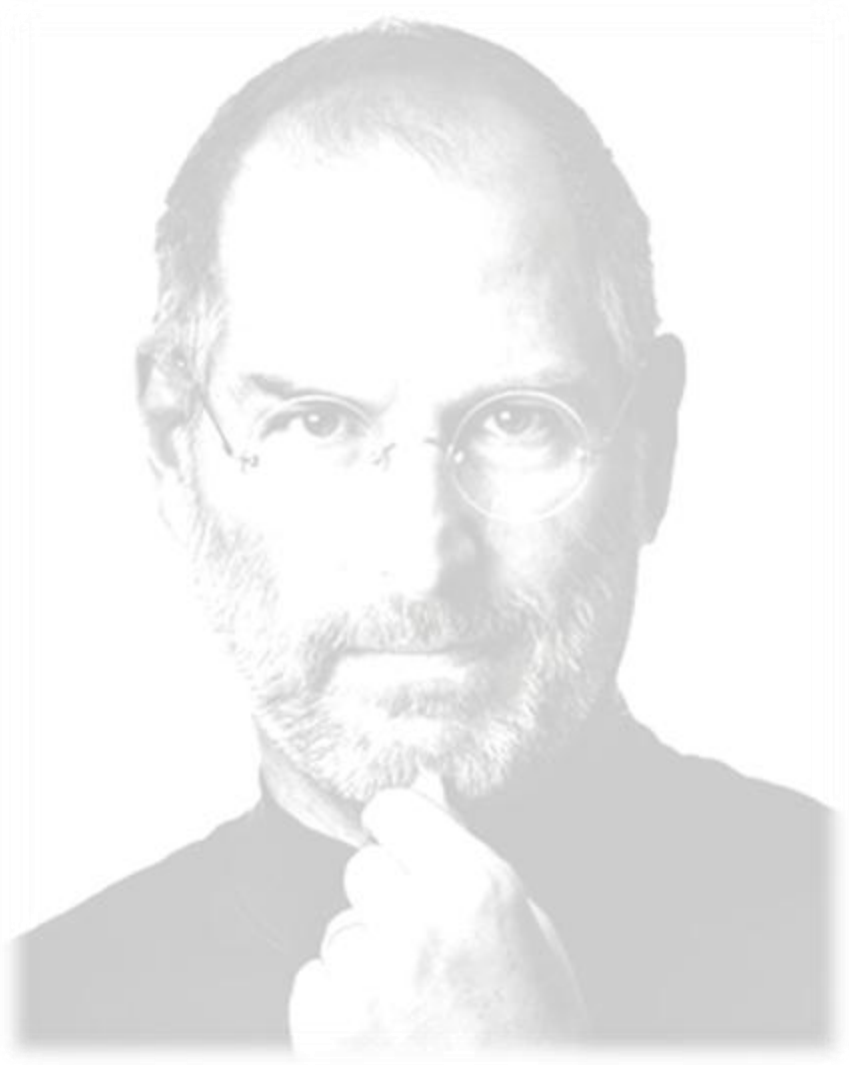
Radamés Fernández López

_____

Alain Lazaro Piñero Añon

_____

FRASE



"Tu tiempo es limitado, de modo que no lo malgastes viviendo la vida de alguien distinto. No quedes atrapado en el dogma, que es vivir como otros piensan que deberías vivir. No dejes que los ruidos de las opiniones de los demás acallen tu propia voz interior. Y, lo que es más importante, ten el coraje para hacer lo que te dicen tu corazón y tu intuición"

Steve Jobs

DEDICATORIA

...A mis Padres por darme todo el apoyo que he necesitado, por su sacrificio y entrega en todo momento, por todo lo que soy en la vida y por lo que pudiera llegar a ser.

...A mi Abuela Nilda que aunque ya no esté conmigo, siempre está en mi pensamiento y en mi corazón.

Yade

…A mi abuela Tina, que le prometí que podía y pude.

…A mis padres Idelmis y Radamés, y mi hermano Raidel, que fueron mi fuerza y mi guía.

…A mi tía loly y mi familia en general que sin Uds. no lo hubiera logrado.

Rada

AGRADECIMIENTOS

…A mi madre, mi padre y mi abuela por toda la confianza que depositaron en mí:

Titina gracias por tener tu sueños junto a los míos.

Papi gracias por apoyarme y ayudarme en cada momento.

Abuela aunque ya no estés aquí conmigo yo te doy las gracias por todo lo que vivimos, por tus cuidados y tus regaños.

…A mi familia, porque de una forma u otra, siempre están a mi lado:

A mi tío Ariel y mi prima Betty por estar siempre presentes.

A mi prima Ana y a mi tía Irene por preocuparse por mí.

…A mis amigas desde el técnico Jessy, Anis y Arianna por su apoyo incondicional y por mantenerse a mi lado a pesar de las adversidades y las distancias.

…A mis amistades de la universidad Liset, Aylen, Mimí, Dania, Diana, Lazaro y Vero:

Gracias Liset por soportarme todo el tiempo y por seguirme en todas mis locuras.

Gracias Vero, a ti y a Nenito por consentirme todo el tiempo.

Y a las que sin estar conmigo aquí siguieron apoyándome Aylin, Lisbet y Nelvis.

…A mis dos grandes amigos Tony y Reiniel por cuidarme y aconsejarme:

Gracias Tony por tu confianza y por ser uno de los dos hermanitos que la vida no me dio.

…A mi otro hermanito y primo: David, por estar siempre ahí me caiga o me levante.

…A mis profesoras Rosalina, Ana Maris, a mi tutor Alain y mi oponente Mailyn por influir positivamente en mi formación profesional.

…A mis compañeros de aula en estos cinco años y a los del proyecto en estos últimos tres añitos.

Yade

…A mis abuelos Tina y Millo, por su educación, por su amor, su cariño, su ternura, su paciencia y porque siempre en mi corazón serán lo más grande, que pueda recordar.

…A mis padres, por su alegría amor y confianza sin límites, que me inspira y me da la fuerza para seguir.

…A mis hermanos, Jorge, Yuliet y Raidel que están preparando su futuro como profesionales y siempre son una motivación y siempre debo ser un ejemplo para ellos, gracias.

…A mis tías loly, milo, y todos en general ya que han sido protagonistas en esta historia por sus consejos, apoyo y por su paciencia, gracias; a mi familia completa por ser partícipe de este momento y siempre estar ahí para mí como estaré yo ahí para uds. Gracias.

…A mis amigos, Osmar, Sergio, los alejandros, los Migue, Medardo, Osvaldito, Arito, Yoan, Idalberto, Arlen a todos ustedes, gracias, por permitirme entrar en sus vidas y formar parte de algo extraordinario, que es tener una buena amistad.

…A Melinda y familia, que han sido extraordinariamente incondicionales conmigo, Muchas Gracias por todo.

…A mis compañeros de estos cinco años, a quienes quiero mucho y nunca voy a olvidar, Luis Carlos, Karel, Silvio, Jorge Fonseca, Jorge Ricardo, Leandro, Yanito, Reiniel, Raul, Camilo, Liliam, Yanet, Lliset, Diana y Yenquiel estos me han ayudado en todo y por eso gracias, no podría dejar de mencionar a Ramón, Enrique, Alain, Felix, Juan Jose, Antonio y a mi compañera de tesis Yadelis, gracias a todos ustedes, por los momentos de alegría, por enseñarme y ayudarme tanto, por compartir esta etapa inolvidable. A todos los profesores del proyecto, Manuel, Bello, mi tutor Alain y mi oponente Mailyn, gracias por ayudarme en la culminación de esta investigación.

…A todos, gracias.

Rada

RESUMEN

Desde hace alrededor de cinco años la Universidad de la Ciencia Informáticas (UCI) desarrolla un software para el Banco Nacional de Cuba (BNC), con el objetivo de informatizar todos los procesos que se realizan dentro de sus gerencias. El sistema actualmente en explotación es nombrado Quarxo y se encuentra dividido en subsistemas, uno de estos subsistemas es Mensajería: encargado del envío de mensajes entre el BNC y entidades bancarias a nivel internacional. Para realizar estas comunicaciones se utilizan los mensajes SWIFT (Sociedad de Telecomunicaciones Financieras Interbancarias Mundiales): mensajes que cumplen con la norma ISO 20022 y cuya estructura está desarrollada en formato XML (Lenguaje de Marcas Ampliable).

Para una primera fase de desarrollo se lograron incorporar un grupo de funcionalidades que permiten confeccionar, modificar o cancelar los distintos tipos de mensajes, así como controlar el proceso de firmas a los que estos deben someterse. Sin embargo el subsistema no permitía el envío de mensajes que no partieran de una transacción bancaria, la modificación de mensajes previamente creados, la gestión de permisos para los mensajes, la elección de imprimir o no mensajes que sean modificados, así como los mensajes necesarios para las negociaciones y los depósitos que entran y salen constantemente en el BNC. Para ello se desarrolló el subsistema Mensajería versión dos con la capacidad de agilizar el envío de todos los mensajes SWIFT utilizados en el BNC, respaldado a través de una adecuada gestión de los permisos de usuario otorgados a los funcionarios del banco.

PALABRAS CLAVES

Entidades bancarias, mensajes, subsistema Mensajería, tecnologías

TABLA DE CONTENIDO

TABLA DE ILUSTRACIONES

INTRODUCCIÓN

Desde principios del siglo XV hasta la actualidad los bancos han venido realizando diversas funciones, como guardar fondos, realizar préstamos o cambiar monedas. (Dülmen, 2002) Con el transcurso de los años se han incorporado diversas tareas o procesos que han hecho efectivas las actividades cotidianas que tienen lugar actualmente en los bancos, con el objetivo de llevar a cabo la gestión económica y financiera de los mismos. Como consecuencia de la evolución de los sistemas bancarios a nivel mundial y del progreso de las tecnologías, se hace necesaria la búsqueda de soluciones que permitan el intercambio de información fluida entre las entidades bancarias. (Channon, 2001)

Con el crecimiento internacional del comercio, tradicionalmente financiado por los bancos, se han incrementado los prestamistas, préstamos, acuerdos y negociaciones entre países de todo el mundo. Por lo cual se hizo indispensable adquirir un sistema para la transmisión de mensajes seguros sobre transacciones financieras internacionales: Sociedad de Telecomunicaciones Financieras Interbancarias Mundiales (SWIFT por sus siglas en inglés). Este sistema inicialmente fue creado con el fin de establecer un enlace de comunicación y un lenguaje común para las transacciones financieras internacionales. Para ello cuenta con una serie de mensajes SWIFT: mensajes que cumplen con la norma ISO 20022 y cuya estructura está desarrollada en formato XML (*Extensible Markup Language*, Lenguaje de Marcas Ampliable). (SWIFT, 2013)

En Cuba se ha venido desplegando un continuo proceso para conquistar el campo tecnológico de la información, a través de la informatización de la sociedad. Este proceso abarca las entidades y empresas más significativas dentro de la sociedad cubana, entre las que se encuentra el Banco Nacional de Cuba (BNC).

Como parte de la modernización del sistema bancario cubano, el BNC le solicitó a la Universidad de Ciencias Informáticas (UCI) el desarrollo de un software que fuera extensible a todos los procesos que se desarrollan dentro de las gerencias del propio banco, permitiendo también la interacción de esos procesos con otros procesos externos que se desarrollan en los demás bancos y entidades bancarias del país. El sistema que está en explotación actualmente en el BNC, no permite el envío de mensajes SWIFT que no partan de una transacción bancaria, lo que dificulta el envío de mensajes

informativos hacia las demás instituciones bancarias, por otra parte el sistema no cuenta con todos los mensajes necesarios que son de vital importancia en el pago de las negociaciones y los depósitos que entran y salen constantemente al BNC, además no permite que el subsistema Mensajería se comporte como un flujo, una vez que se llenan los datos del negocio y se procede a llenar el mensaje es imposible regresar para arreglar cualquier dato de la operación contable sin que se pierdan los datos del mensaje, lo que hace agotador el trabajo por parte de los funcionarios del banco al emitir cualquier información. También es necesario que el sistema permita decidir al usuario si desea imprimir o no un mensaje SWIFT previamente modificado y que ofrezca la posibilidad de actualizar los posibles destinos que pudieran tomar los mensajes creados. Por otra parte el sistema aún no cuenta con una adecuada gestión de permisos de usuarios para el trabajo con los mensajes SWIFT.

En correspondencia con el estudio realizado a la problemática anteriormente expuesta, se define el siguiente **problema a resolver:**

La utilización de dos sistemas para el envío de mensajes SWIFT en el BNC, afecta la realización de dicho proceso.

A partir del problema planteado se define como **objeto de estudio:** El proceso de las trasferencias en entidades financieras bancarias.

Campo de acción: La informatización de los procesos de las trasferencias en el BNC.

El **objetivo general** se centra en Desarrollar el subsistema Mensajería versión 2 que soporte el envío de todos los mensajes SWIFT utilizados en el BNC.

Para dar solución al problema planteado se trazan los siguientes **objetivos específicos:**

1. Elaborar el marco teórico de la investigación.
2. Modelar los procesos de los mensajes SWIFT.
3. Elaborar el levantamiento y especificación de requisitos.
4. Diseñar el subsistema Mensajería bajo la arquitectura propuesta.
5. Implementar el subsistema Mensajería bajo la tecnología propuesta.
6. Validar la solución propuesta.

Tareas:

- Caracterización de los procesos relacionados con la gestión de los mensajes SWIFT en entidades bancarias.
- Valoración de las herramientas para gestionar los mensajes SWIFT en entidades bancarias.
- Caracterización de las tecnologías y herramientas a utilizar en el desarrollo de los módulos del subsistema de Mensajería SWIFT.
- Identificación, especificación y validación de los requisitos funcionales del software.
- Descripción de los elementos arquitectónicos correspondientes al análisis del subsistema de Mensajería SWIFT.
- Elaboración de los diagramas de clases del diseño, diagramas de secuencia y diagrama de despliegue correspondientes al diseño del subsistema de Mensajería SWIFT.
- Validación del diseño aplicando las métricas adecuadas para esta etapa.
- Caracterización, identificación y aplicación de los patrones utilizados en el desarrollo del subsistema de Mensajería SWIFT.
- Confección del modelo de componentes correspondientes a la implementación del subsistema de Mensajería SWIFT.
- Implementación de los módulos del subsistema de Mensajería SWIFT.
- Validación mediante la aplicación de pruebas de caja blanca y pruebas de caja negra a los módulos del subsistema de Mensajería SWIFT.

Para el estudio preliminar de la investigación se utilizaron los siguientes métodos científicos:

Métodos Teóricos:

- Histórico – Lógico: se utilizó para realizar un estudio de las tendencias históricas y actuales de los sistemas bancarios que existen en el mundo y determinar su influencia en el problema actual de la investigación.

- Analítico - Sintético: se utilizó para realizar un análisis de la documentación empleada para el desarrollo de la investigación.

- Modelación: se utilizó para modelar los requerimientos funcionales que se proponen en la solución, su relación con los procesos actuales y el diseño de los nuevos componentes visuales, las entidades relacionales y sus características.

Métodos Empíricos:

- Entrevistas: este método fue de gran utilidad para realizar el estudio de los procesos actuales referentes a la gestión de mensajes SWIFT en el BNC.

Aporte práctico, Resultados: Subsistema Mensajería SWIFT versión 2.

Idea a defender

Si se desarrolla el subsistema Mensajería SWIFT versión 2 se facilitará la creación y procesamiento de los mensajes SWIFT en el BNC simplificando el proceso en una sola aplicación.

Estructura del Informe:

Capítulo I Fundamentación Teórica: En este capítulo se muestra el respaldo teórico de los temas tratados en el informe, expone los fundamentos teóricos relacionados con el objeto de estudio. Se hace referencia a los principales conceptos, patrones, tendencias, técnicas, metodología de desarrollo, lenguaje de desarrollo y herramientas de desarrollo usadas para la solución del problema.

Capítulo II Análisis y Diseño: Este capítulo incluye una caracterización de la arquitectura definida en el proyecto y los principales artefactos del análisis y diseño del subsistema Mensajería: diagramas de clases del análisis, diagramas de colaboración, diagramas de clases del diseño, diagramas de secuencia generados a partir de las funcionalidades identificadas para dar respuesta al problema planteado.

Capítulo III Implementación y Prueba: Este capítulo se enmarca esencialmente en la construcción de la solución exponiendo los aspectos fundamentales de la implementación de la misma. Incluye la descripción de las clases y funcionalidades que engloban los requisitos detectados. También abarca la realización de pruebas de caja blanca y pruebas de caja negra de los módulos del subsistema de Mensajería SWIFT.

CAPÍTULO I. FUNDAMENTACIÓN TEÓRICA

1.1. Introducción

El siguiente capítulo muestra el respaldo teórico de los temas tratados en el informe. Hace referencia a los principales conceptos relacionados con la gestión de los mensajes SWIFT en entidades bancarias. Se describen los sistemas y estándares existentes a nivel mundial para la mensajería SWIFT. Se caracterizan las tecnologías y herramientas a utilizar en el desarrollo de los módulos del subsistema Mensajería SWIFT. El objetivo que persigue el mismo es exponer los fundamentos teóricos relacionados con el objeto de estudio.

1.2. Banco

Los bancos son entidades financieras o instituciones de tipo financiero encargadas de realizar diversas actividades comerciales y operaciones de crédito, cambio, compraventa, gestión de pagos y actividades propias de las cuentas corrientes. (Piñero Añon, 2011)

Entre las funciones de los bancos modernos se pueden encontrar:

- La intermediación de crédito.
- La intermediación de los pagos.
- La administración de los capitales.

1.2.1. Sistema bancario cubano

El sistema bancario cubano actualmente es rectorado por el Banco Central de Cuba y está constituido por 8 bancos comerciales, 18 instituciones financieras no bancarias, 13 oficinas de representación de bancos extranjeros y 4 oficinas de representación de instituciones financieras no bancarias. Dentro de las instituciones financieras que operan en Cuba se encuentra el BNC. (Piñero Añon, 2011)

1.2.2. BNC

El BNC fue creado en diciembre de 1948, bajo la Ley #13, inicialmente como Banco Central del Estado. No pasa a iniciar sus funciones hasta 1950. En febrero de 1998 es liberado de sus funciones de banca central y pasa a atender primordialmente la deuda externa del estado. (Piñero Añon, 2011)

Es una institución del estado encargada de emitir la moneda nacional y velar por su estabilidad, así como obtener y otorgar crédito tanto en moneda nacional como libremente convertible. Entre sus funciones

esenciales se destacan todas las negociaciones referentes a la deuda externa del país. También fija las tasas de interés aplicables a las operaciones que efectúa el Banco, dentro de los límites que establezca el Banco Central de Cuba. Negocia a través de documentos mercantiles como letras de cambio, pagarés y cheques. (Piñero Añon, 2011)

1.3. Mensajería financiera

Sistema de comunicación que permite el intercambio de mensajes (envío y recepción) entre usuarios interconectados a través de una red de ordenadores pertenecientes al sector financiero. Cumpliendo de esta manera con determinadas reglas y estándares definidos en instituciones, órganos y organismos financieros a nivel nacional e internacional. (Channon, 2001)

1.3.1. SWIFT. Definición y surgimiento

Society for Worldwide Interbank Financial Telecommunication o **SWIFT** (en español Sociedad de Telecomunicaciones Financieras Interbancarias Mundiales). Esta organización es una sociedad cooperativa que posee un sistema de mensajería interbancario utilizado por la mayor parte de los bancos del mundo para enviar mensajes. Tiene a cargo una red internacional de comunicaciones financieras entre bancos y entidades financieras a nivel internacional. (SWIFT, 2013)

SWIFT surge en Bruselas en 1973 respaldado por 239 bancos en 15 países, con el objetivo de crear un enlace de comunicaciones y procesamiento de datos compartido para todo el mundo y un lenguaje común para las transacciones financieras internacionales. El 31 de mayo de 2012 alcanza el hito de 20 millones de mensajes enviados por medio de este sistema. Actualmente tiene más de 10.000 miembros ubicados en más de 210 países y envía millones de mensajes a diario actualizando sus sistemas con el trascurso de los años por otros más modernos, rápidos y baratos. Cada banco tiene un código internacional ISO 9632 que lo identifica en el sistema. El código SWIFT de un banco está formado por el código del país, el código del banco y una serie de datos adicionales, como la localización o el tipo de sucursal. (SWIFT, 2013)

1.3.2. SWIFT. Beneficios

En la actualidad con el gran auge que toma la competencia y a medida que avanza la tecnología, se han incrementado considerablemente los sistemas utilizados en todos los campos de la sociedad. En el caso de SWIFT gracias a sus potenciales beneficios ha logrado mantenerse consistentemente en el puntal más

alto de su rama, superándose con el transcurso de los años y avanzando a medida que avanzan las nuevas tecnologías. Entre los beneficios que ofrece SWIFT se destacan (SWIFT, 2013):

- Conexión con todos los bancos.
- Seguridad al enviar información encriptada.
- Independencia con bancos.
- Sin restricciones de horarios.
- Incorporación de nuevos bancos de forma ágil y segura.
- Oportunidad para el monitoreo de transacciones.
- Define normas y estudia soluciones a problemas de interés común.
- Elimina ineficiencias operacionales.

1.3.3. SWIFT Mensaje MT

La red de mensajería SWIFT funciona con una serie de mensajes normalizados. Un mensaje SWIFT es identificado por la palabra MT seguido de un número de tres dígitos. El prefijo "MT" significa "tipo de mensaje", y el número indica uno de los formatos de mensaje normalizado que componen el sistema de mensajería SWIFT. El empleado bancario encargado del envío de los mensajes SWIFT llena los campos correspondientes al tipo de mensaje que necesite enviar. Los campos y sub-campos del mensaje estarán en correspondencia con el tipo de mensaje SWFT seleccionado por dicho funcionario. (Chaviano, 2011)

1.3.4. Ejemplos de mensajes SWIFT

MT-299, MT-199, MT-999, MT-300, MT-103, MT-102, MT-750, MT-754

Existen nueve categorías para mensajes de usuario a usuario, identificadas de 1 a 9, y una categoría separada para mensajes intercambiados entre usuarios y el sistema, identificada como categoría 0:

- System Messages MT categoría 0: utilizados para mensajería general de sistema, notificación de entregas y reportes de mensajes procesados.
- User-to-User Messages MT categorías 1-9: utilizados para la realización de transacciones financieras:
    - 1xx - Transferencias de clientes y cheques.
    - 2xx - Transferencias de instituciones financieras.

- 3xx - Operaciones de cambio extranjero, préstamo/depósito y contratos precio convenido.
- 4xx - Remesas documentarias.
- 5xx - Valores.
- 6xx - Sindicaciones.
- 7xx - Créditos documentarios y garantías.
- 8xx – Travellers o cheques.
- 9xx -Mensajes de estados de las cuentas.

1.3.5. Sistemas utilizados para la mensajería financiera a nivel mundial

B2B Data Transformation:

Utilizado por Natixis, una de las entidades financieras de más rápido crecimiento en Europa con sedes en Paris y Nueva York, con el objetivo de permitir la transformación y entrega automatizadas de datos SWIFT, constituyó la solución para desarrollar un sistema financiero de última generación al ser combinado con una plataforma de integración de datos empresariales. (Piñero Añon, 2011)

Se usa en la extracción de datos de cualquier archivo, documento o mensaje (incluyendo datos estructurados y mensajes SWIFT) con independencia del formato, la complejidad o el tamaño, y los transforma en un formato utilizable. Incluye soporte para los estándares del sector, al permitir a las empresas integrar sin problemas mensajes SWIFT, EDI (*Electronic Data Interchange*, Intercambio Electrónico de Datos) y NACHA (*National Automated Clearing House Association***,** Asociación Nacional de la Cámara de Compensación Automatizada). Los procesadores de documentos facilitan la extracción de datos de archivos Adobe PDF y de documentos de Microsoft Word y Excel. Provee una mejor visibilidad e integridad de los datos gracias a una arquitectura de integración de datos basada en metadatos. (Piñero Añon, 2011)

Incentage Middleware Suite (IMS):

Es un software para la gestión de mensajes financieros, no financieros y archivos entre bancos y entidades financieras con el sistema SWIFT. IMS tiene implementación completamente modular, está desarrollado en Java, lo que provoca ganancias en independencia a nivel de plataforma, utiliza un modelo con base en configuración XML lo que permite una fácil escalabilidad y carga balanceada. Fue laureado con la certificación *SWIFTReady Gold Financial EAI* (*Enterprise*

Aplication Integration, Integración de Aplicaciones Empresariales) 2006 confirmando la calidad del software y la fidelidad con los más altos estándares de SWIFT. (Piñero Añon, 2011)

IMS posee definiciones pre-integradas para todas las categorías y tipos de mensajes, este factor, acelera el proceso de conversión de mensajes SWIFT, incluye módulos de creación, visibilidad y modificación de mensajes, utiliza un tipo de directorio de datos que contiene todas las definiciones de mensajería SWIFT, campos y bloques. Esto permite gran flexibilidad para los usuarios ofreciendo la utilización de datos significativos a la hora de generar los mensajes. (Piñero Añon, 2011)

SWIFT Alliance Integrator:

SWIFT Alliance ayuda a conectar las aplicaciones empresariales a SWIFT, funciona como una pasarela para la validación y la autorización de pagos.

Permite disminuir al mínimo las modificaciones necesarias en las aplicaciones de back-office (parte de las empresas donde se realizan las tareas destinadas a gestionar la propia empresa y con las cuales el cliente no necesita contacto directo), reducir el tiempo y los gastos relacionados con la implementación de las soluciones empresariales a través de SWIFTNet. (SWIFT, 2013)

El *Alliance Integrator* es completamente compatible con todos los servicios de mensajería de SWIFT y además incluye:

- ✓ La definición de las transformaciones requeridas entre sus formatos privados y los formatos de carga útil de SWIFT.
- ✓ Herramientas de configuración empresarial y técnica que abarcan la personalización de los adaptadores y eventos, así como la configuración de usuarios, etapas de procesamiento y transacciones, y el procesamiento relacionado con corresponsales.
- ✓ Medios de supervisión basados en navegador para el estado comercial y para los eventos de *Integrator*. Esta funcionalidad incorpora la posibilidad de controlar el acceso a la información mediante funciones y privilegios.
- ✓ Generación de eventos empresariales y eventos técnicos, que se incluyen en el proceso de supervisión.

> Otras características importantes del software son la interoperabilidad que brinda entre los mensajes MT y MX (mensajes que contienen elementos reusables del negocio para construir XML estándares) a partir de reglas definidas por SWIFT para la conversión automática y la compatibilidad con varias versiones de los estándares de mensajes. (SWIFT, 2013)

Los sistemas presentados para la mensajería financiera a nivel mundial, son considerados soluciones informáticas de alta calidad debido a la gran proporción de funcionalidades que los componen y a la seguridad con que manejan la información, sin embargo por su condición de privativos el estado cubano no puede utilizarlos debido al embargo económico que Estados Unidos mantiene contra Cuba desde hace más de 50 años, por lo que es imposible obtener legalmente las licencias y económicamente tampoco se cuenta con el presupuesto necesario para realizar inversiones en este ámbito. Esto resulta engorroso para las entidades del estado cubano, debido a que los sistemas presentados incluyen tecnología de Estados Unidos y los propietarios están sujetos a sanciones de conformidad con las leyes de Estados Unidos.

A pesar de dicho inconveniente aportan una fructuosa visión en lo referente a tecnologías para el desarrollo del software solicitado por el BNC, ya que sirvieron de base para la identificación de una serie de funcionalidades para el trabajo con mensajes SWIFT. Ejemplo de ello es el manejo de datos que se realiza dentro de estos sistemas y que como solución alternativa en Quarxo se implementa con Hibernate que es un framework libre.

1.4. Sistema alternativo para la mensajería financiera en el BNC

Inicialmente el BNC utilizaba para sus funciones habituales los siguientes sistemas: SABIC (Sistema Automatizado para la Banca Internacional del Comercio), el SISCOM (Sistema de Comunicación para la mensajería SWIFT) y SLBTR (Sistema de Liquidación Bruta en Tiempo Real). Al migrar al sistema Quarxo, el subsistema Mensajería SWIFT perteneciente al mismo, pasó a suplir los servicios que se prestaban con el sistema SABIC.

Quarxo en una primera fase para el subsistema Mensajería SWIFT, incluye entre sus módulos el proceso: Captación de mensajes. Este proceso es el encargado de confeccionar, modificar o cancelar los distintos tipos de mensajes, así como controlar el proceso de firmas a los que estos deben someterse. A pesar de que la aplicación cuenta con las funcionalidades necesarias para el envío de mensajes SWIFT, no permite entre sus opciones integrar el envío de mensajes que no inicien operaciones contables y no posee todos

los mensajes necesarios para las áreas de negociaciones y depósitos; por lo que actualmente se hace necesario el uso del SISCOM para enviar dichos mensajes.

El SISCOM es un producto para el procesamiento y enrutamiento de mensajería SWIFT a través de la red SWIFT FIN. El sistema utiliza como interfaz los servicios de un servidor SWIFT. (SIBANC, 2006)

Este sistema está desarrollado en Fox Pro, procedimientos almacenados en el *SQL Server* 2005 y un componente de comunicación sobre el lenguaje C. Los mensajes que son intercambiados responden a la norma ISO 15022 del estándar SWIFT, la cual define los mensajes, su estructura y las reglas de validación semántica que deben aplicarse. (Chaviano, 2011)

SISCOM está conformado por 4 módulos:

- SACIM (Sistema de Archivo, Captación e Impresión de Mensajes): módulo encargado de crear, archivar, imprimir y firmar los mensajes SWIFT que son utilizados entre las entidades financieras.
- Middleware (Intermediario entre SISCOM y *Alliance Access*): módulo encargado de enrutar y distribuir la mensajería entre los sistemas SISCOM de cada banco y el servidor *Alliance Access* ubicado en el Banco Central de Cuba, garantizando el flujo de mensajes en ambos sentidos para las estaciones de comunicación.
- SwAdmin (Funciones administrativas): módulo encargado de configurar y administrar el sistema, también se ocupa de crear una salva del tráfico diario de mensajes, de la recuperación de los mismos y de la generación de reportes.
- SwMPServer (Interacción con *Alliance Access*): módulo encargado de interactuar con el *Alliance Access*.

A pesar de que el SISCOM funciona correctamente en el envío de mensajes, posee una serie de inconvenientes:

- No brinda las funcionalidades necesarias para soportar los cambios que ocurren en la norma ISO 15022 del estándar SWIFT.
- No se pueden guardar todos los formatos de los mensajes definidos en el estándar.
- Mantiene sus bases de datos en las estaciones de trabajo lo que representa una vulnerabilidad del sistema.
- Existe redundancia en la información que guarda.
- No permite la integración con otras aplicaciones desarrolladas en un lenguaje diferente al Fox Pro.

Por tales deficiencias es necesaria la migración de las funcionalidades que son realizadas en el sistema SISCOM hacia el sistema QUARXO. Ya que a pesar del progreso en el subsistema Mensajería SWIFT todavía hay mensajes que son enviado a través del SISCOM como sistema alternativo.

1.5. Estándares de comunicación

(Tipo, modelo, norma, patrón): los estándares son considerados acuerdos o normas documentados y están conformados por especificaciones técnicas u otros criterios, permitiendo así utilizarlos como reglas o guías para asegurar que la comunicación se ajuste ha determinado propósito. Son las reglas generales confeccionadas y establecidas para que los sistemas o componentes de comunicación sean comunes. (OPENTIA, 2007)

1.5.1. Estándar de comunicación SWIFT: Norma ISO 15022

SWIFT es únicamente un transmisor de mensajes. No posee fondos ni gestiona cuentas en nombre de los clientes, ni tampoco almacena información financiera de forma permanente. Al actuar como transmisor, sirve de vehículo para los mensajes transmitidos entre dos instituciones financieras. Esta actividad implica el intercambio seguro de datos privados, al tiempo que se garantiza su confidencialidad e integridad. (SWIFT, 2013)

La primera norma ISO 7775 fue creada alrededor de los años 80, la misma se encargaba de definir un catálogo de mensajes. Luego surgió la norma ISO 11521 y la norma internacional ISO 15022. Esta última sustituyó a las dos normas anteriores, identificando el mensaje por la palabra MT seguido de un número de tres dígitos. El primer dígito responde a la categoría del mensaje, el segundo al grupo dentro de la categoría y el tercero al número del mensaje dentro del grupo. Existen un total de 10 categorías, desde la categoría 1 hasta la categoría 9 responden a operaciones financieras y una categoría n o 0 para propósito general. (Chaviano, 2011)

Hoy por hoy se está adoptando internacionalmente la norma ISO 20022, cuya estructura está formada en formato XML, por el contrario de las normas antes expuestas que están creadas en texto plano (Miguel Iglesias, 2012).

1.5.2. Estándares para la transmisión y recepción de mensajes a nivel mundial

XBRL:

Extensible Business Reporting Language o **XBLR** (en español Lenguaje Extensible de Informes de Negocios). Surge en 1998 con el objetivo de simplificar la automatización del intercambio de información financiera mediante el uso del lenguaje XML.

Es un lenguaje universal para los reportes y el análisis de la información financiera de las empresas vía Internet, permite la creación de informes financieros personalizados, a bajo costo y es un formato compatible con la mayoría de las aplicaciones informáticas de contabilidad y de análisis de datos. XBRL se deriva de la filosofía del XML por lo que es independiente de la aplicación y puede integrarse con casi todos los sistemas de bases de datos existentes en la actualidad. (Piñero Añon, 2011)

FIX:

Financial Information Exchange o **FIX** (en español Protocolo de Intercambio de Información Financiera). Surge en 1992 con el objetivo de establecer comunicación entre dos empresas, las empresas Fidelity Investments y Salomon Brothers.

Es una serie de especificaciones de mensajería para la comunicación electrónica de datos financieros, incluyendo los mensajes relacionados con las negociaciones. Es un estándar globalmente aceptado de especificaciones de mensajería desarrollado con la colaboración de bancos, corredores, cambios, inversores institucionales y proveedores de tecnología de información de todo el mundo. (Agile software development & services, 2010)

EDIFACT:

Electronic Data Interchange For Administration, Commerce and Transport o **EDIFACT** (en español Intercambio electrónico de datos para la Administración, Comercio y Transporte). Es un estándar de la Organización de las Naciones Unidas para el Intercambio electrónico de datos en el ámbito mundial.

Se compone de un conjunto de normas aprobadas a nivel internacional, de archivos y de directorios para el intercambio electrónico de datos estructurados, en particular estos que

conciernen al comercio y a los servicios, de aplicación en aplicación y entre entidades independientes. Estas reglas son aprobadas y publicadas por la CEE-ONU. (EDIFACTMX)

SWIFT:

El estándar SWIFT solamente se utiliza entre entidades financieras, se rige por unos estrictos códigos de conducta para garantizar la utilización correcta de la transmisión del dinero, está basado en la norma ISO 20022. En 1999 adoptó el lenguaje XML que es una sintaxis para transmitir información de una manera estructurada y es paralelo al HTML (*HyperText Markup Language*, Lenguaje de Marcado Hipertextual). SWIFT debido a sus características y su amplia gama de mensajes fue seleccionado por el Banco Central de Cuba como el estándar para la comunicación financiera en Cuba. (Piñero Añon, 2011)

1.6. Metodología de desarrollo. Modelo de desarrollo

Cada proyecto según sus características particulares, su equipo de desarrollo y recursos debe proceder a utilizar la metodología de desarrollo que se acerque lo más posible a sus necesidades. Su objetivo es estructurar, planificar y controlar el proceso de desarrollo de software mediante herramientas, modelos y métodos. (Areba, 2001)

El Centro de Informatización de la Gestión de Entidades (CEIGE) ha definido para el desarrollo de software un Modelo de Desarrollo de Software predefinido por el propio centro. Donde para el ciclo de vida de los proyectos pertenecientes al mismo se tienen en cuenta las fases y actividades por áreas de procesos que plantea el nivel dos de CMMI (*Capability Maturity Model Integration* o en español Integración de Modelos de Madurez de Capacidades) establecido en la UCI. (CEIGE, 2012)

El modelo definido cuenta con siete fases esenciales: Estudio Preliminar, Modelado del Negocio, Requisitos, Análisis y Diseño, Implementación, Pruebas Internas y Pruebas de Liberación. La realización o no de alguna fase depende de las necesidades particulares del proyecto. Cada fase tiene una serie de actividades o tareas a realizar para darle cumplimiento a los objetivos trazados. (CEIGE, 2012)

1.7. Patrones de diseño y arquitectura

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

patrones arquitectónicos reflejan la estructura global de un sistema, especificando 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, tienen un nivel de abstracción menor, estando más próximos a la implementación final. (Tejada, 2002)

1.7.1. Patrones de diseño: GOF

Los patrones de diseño GOF (*Gang of Four*, Banda de los cuatro) se clasifican en tres categorías esenciales:

- Patrones de creación: referente a la creación de instancias de las clases.
- Patrones estructurales: referente a las relaciones entre clases u objetos.
- Patrones de comportamiento: referente a la interacción y cooperación entre clases.

Patrones estructurales

Facade (Fachada):

El patrón Fachada provee una interfaz unificada simple para acceder a una interfaz o grupo de interfaces de un subsistema. Permite reducir la dependencia entre clases y ofrece un punto de acceso al resto de clases, sin ocultar las clases. Conoce qué clases del subsistema son responsables de qué peticiones, delegando así las peticiones a los objetos correspondientes. Separando al cliente de los componentes del subsistema, se reduce el número de objetos con los que el cliente trata, lo cual facilita el uso del subsistema. (Piñero Añon, 2011)

Patrones de comportamiento

Mediator (Mediador):

El patrón Mediador define un objeto que coordine la comunicación entre objetos de distintas clases, pero que funcionan como un conjunto. Este patrón define un objeto que encapsula la forma en que interactúan un grupo de objetos, promoviendo así un acoplamiento débil al evitar las referencias explícitas entre los objetos. El uso de este patrón Mediator hace que el código sea mucho más legible, ya que favorece la cohesión y ayuda a crear clases desacopladas. (Fonseca, 2011)

Strategy (Estrategia):

El patrón Estrategia encapsula varios algoritmos alternativos en diferentes clases y ofrece una interfaz única para acceder a la funcionalidad ofrecida, o sea definir una familia de algoritmos encapsulando por separado cada uno de ellos y haciéndolos intercambiables. Esto permite a los algoritmos variar con independencia de los clientes que los usan. (Fonseca, 2011)

1.7.2. Patrones de diseño: GRAPS

Los patrones GRASP (*General Responsibility Assignment Software Patterns*, Patrones de Asignación de Responsabilidades) describen los principios fundamentales de la asignación de responsabilidades a objetos, expresados en forma de patrones.

Patrón experto:

Su función esencial se basa en asignar una responsabilidad al experto en información: la clase que cuenta con la información necesaria para cumplir la responsabilidad. El comportamiento se distribuye entre las clases que cuentan con la información requerida, de modo que se obtendrá un diseño con mayor cohesión. La información se mantendrá encapsulada ya que los objetos se valen de su propia información para hacer lo que se les pide, soportando un bajo acoplamiento. (López, 2003)

Patrón creador:

Su función esencial se basa en identificar quién debe ser el responsable de la creación (o instanciación) de nuevos objetos o clases. Asignarle a la clase B la responsabilidad de crear una instancia de la clase A. Un objeto sólo pueda ser creado por el objeto que contiene la información necesaria para ello. Brinda soporte a un bajo acoplamiento. (López, 2003)

Patrón controlador:

Su función esencial se basa en controlar el flujo de eventos del sistema. Es un intermediario entre la interfaz de usuario y el núcleo de las clases donde reside la lógica de la aplicación. El controlador no realiza mucho trabajo por sí mismo; más bien coordina la actividad de otros objetos. Este patrón sugiere que la lógica de negocios debe estar separada de la capa de presentación para aumentar la reutilización de código y a la vez tener un mayor control. (López, 2003)

Patrón alta cohesión:

La cohesión es una medida de cuán relacionadas y enfocadas están las responsabilidades de una clase. Por lo que su función esencial es asignar una responsabilidad de modo que la cohesión siga siendo alta. Trabaja en base a que una clase tenga responsabilidades moderadas en un área funcional y colabora con las otras para llevar a cabo las tareas. Caracteriza a las clases con responsabilidades estrechamente relacionadas que no realizan un trabajo enorme. (López, 2003)

Patrón bajo acoplamiento:

El acoplamiento es una medida de la fuerza con que una clase está conectada a otras clases, con que las conoce y con que recurre a ellas. Por lo que su función esencial es asignar una responsabilidad de modo que el acoplamiento siga siendo bajo. Acoplamiento bajo significa que una clase no depende de muchas clases. Beneficia la reutilización y permite que no se afecten los cambios en otros componentes. (López, 2003)

1.7.3. Patrones J2EE

J2EE es una arquitectura por si misma que involucra otras arquitecturas, incluyendo servlets, JSP (*Java Server Page*) y *Enterprise JavaBeans*, teniendo así su propio conjunto patrones específicos para diferentes aplicaciones empresariales.

Composite view (Vista compuesta)

El patrón de vista compuesta pertenece a la capa de presentación y está basado en el patrón composite, que describe las herencias parte-totalidad cuando un objeto compuesto se compone de varias piezas, todas ellas tratadas como equivalente lógicos. Un objeto vista que está compuesto de otros objetos vista. Por ejemplo, una página JSP que incluye otras páginas JSP y HTML usando la directiva *include* o el *action include* es un patrón *Composite View*. (CiberAula, 2010)

Patrón de diseño DAO (Objeto de Acceso a Datos)

Plantea como solución, utilizar un "*Data Access Object"* (DAO) para abstraer y encapsular todos los accesos a la fuente de datos. Este maneja la conexión con la fuente de datos para obtener y almacenar datos. Encapsula la forma de acceder a la fuente de datos (base de datos, archivos) y oculta el modo de acceso a los datos. Cada DAO implementa una serie de métodos declarados en la interfaz DAO correspondiente, pudiendo incluir también nuevos métodos. Es utilizada en Java para aislar la tecnología

de persistencia como: JDBC (*Java Database Connectivity*), JDO (*Java Data Object*), Hibernate y *Enterprise JavaBeans*, permitiendo actualizar la tecnología subyacente sin cambiar otras partes de la aplicación. (Tejada, 2002)

1.7.4. Patrón arquitectónico: MVC

MVC (Modelo-Vista-Controlador) este patrón se encarga de dividir 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.

1.8. Lenguaje, notación y herramienta de modelado

1.8.1. Herramienta CASE

Computer Aided Software Engineering o **CASE** (en español Ingeniería de Software Asistida por Computación) son diversas aplicaciones informáticas que permiten modelar los procesos de negocios de las empresas y desarrollar sistemas de información gerenciales. Por medio de esta herramienta se puede verificar el uso de todos los elementos en el sistema diseñado, automatizar los diagramas correspondientes a las distintas fases del proceso de desarrollo, también ayuda en la documentación del sistema y la generación de código. Sus objetivos están enfocados en al aumento de la calidad de software a través del mantenimiento de software, la planificación de proyectos, reutilización del software, portabilidad y estandarización de la documentación. (López, 2003)

1.8.1.2. Visual Paradigm (Herramienta de Modelado)

Visual Paradigm es una suite completa de herramientas CASE que ofrece una gama de facilidades para el modelado de aplicaciones con alta calidad y a un menor costo. Está enfocada a la creación de diseños

usando el paradigma de programación orientada a objetos e inclusive admite la creación de prototipos no funcionales para obtener una visión gráfica del sistema en desarrollo. Provee soporte para la generación de código y permite la integración con otros IDE´s como: Eclipse, NetBeans, JBuilder y JDeveloper. Entre sus tantas ventajas se destaca la posibilidad que brinda de realizar ingeniería inversa para aplicaciones realizadas en Hibernate, XML y Java. Estas dos últimas características hacen que sea adecuada para el desarrollo del sistema Quarxo, ya que la vinculan estrechamente con la tecnología utilizada en el proyecto. Además es de vital importancia destacar que brinda facilidades para integrarse con la notación BPMN. (trusted, 2012)

1.8.2. BPMN (Notación para el Modelado de Procesos de Negocio)

Business Process Modeling Notation o **BPMN**: es un estándar que provee una representación gráfica para modelar procesos de negocio, de manera tal que las partes involucradas puedan comunicarse mediante un lenguaje de notación común. Brinda la posibilidad de lograr un mejor entendimiento del negocio, optimizando así los procesos de negocio a desarrollar. Para esto posee varias categorías de elementos gráficos como son: Objetos de Flujo (Eventos, Actividades, *Gateways*), Objetos de Conexión (Asociación, Flujo de Secuencia, Flujo de mensajes), *Swimlanes* y Artefactos (Objetos de Datos, Grupo, Anotación). (Fonseca, 2011)

1.8.3. UML (Lenguaje Unificado de Modelado)

Unified Modeling Language o **UML**: es el lenguaje de modelado de sistemas de software más conocido y utilizado en la actualidad. Captura la idea de cómo debe comportarse un sistema para comunicarla posteriormente a quienes se encarguen de su proceso de desarrollo. Por ello es el lenguaje seleccionado para modelar los nuevos requisitos a incluir al subsistema Mensajería. Permite visualizar, construir y documentar los artefactos que se van generando para la solución del problema en cuestión, involucrando gran cantidad de software y soportando además el paradigma orientado a objetos. Cuenta con una serie de elementos gráficos, donde al ser considerado un lenguaje dichos elementos cumplen ciertas y determinadas reglas, visualmente demostrado por los diagramas que lo componen. (López, 2003)

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.

1.9.1. Plataforma JEE

Java Enterprise Edition (JEE) es una plataforma, creada por *Sun Microsystems*, que ofrece soluciones para el desarrollo, definiendo estándares para desarrollar aplicaciones en el lenguaje Java. La plataforma como tal es una máquina virtual encargada de la ejecución de aplicaciones, y un conjunto de librerías estándar que ofrecen funcionalidad común. Las aplicaciones desarrolladas en esta plataforma tienden a ser portables, escalables, robustas y seguras. (Fonseca, 2011)

1.9.2. Java (Lenguaje de Programación)

Java es un lenguaje de propósito general, y como tal es válido para realizar aplicaciones profesionales. Es un lenguaje de programación intrínsecamente orientado a objetos y fácil de integrar con diferentes frameworks de desarrollo. Permite programar páginas web dinámicas con accesos a bases de datos utilizando XML con cualquier tipo de conexión de red entre cualquier sistema. Posee una gran funcionalidad gracias a sus librerías (clases), proporcionando un conjunto de clases para su uso en aplicaciones de red, permitiendo abrir sockets y establecer conexiones con servidores o clientes remotos. Además de ser poderoso manejando excepciones. (Iglesias, 2010)

1.9.3. Eclipse (Entorno de Desarrollo Integrado IDE)

Eclipse es un entorno de desarrollo integrado (IDE) de código abierto. Es una herramienta multiplataforma y presenta una arquitectura de plugins que lo hace ser bastante flexible y configurable. Permite integrar diversos lenguajes e introducir otras aplicaciones útiles para el proceso de desarrollo, tales como: herramientas UML y editores visuales de interfaces. (Fonseca, 2011)

1.9.4. SQL Manager 2010 for SQL Server (Gestor de Base de Datos)

Microsoft SQL Server es un sistema para la gestión de bases de datos basado en el modelo relacional. Ofrece un conjunto completo de herramientas de bases de datos gratuitas que alcanzan una mayor disponibilidad y un mejor rendimiento en el mantenimiento de datos y administración de las bases de datos. Permite trabajar en modo cliente-servidor, donde la información y datos se alojan en el servidor y los terminales o clientes de la red sólo acceden a la información, además permite administrar información de otros servidores de datos. (Iglesias, 2009)

1.9.5. Tomcat (Contenedor Web)

Tomcat funciona como un contenedor de servlets desarrollado bajo el proyecto Jakarta en la *Apache Software Fundation*. Implementa las especificaciones de los servlets y de los JSP. Fue escrito en Java, lo que le permite funcionar en cualquier sistema operativo que disponga de una máquina virtual Java. Además de ser un servidor web autónomo en entornos de alta disponibilidad y alto nivel de tráfico. (Iglesias, 2010)

1.9.6. Frameworks

Un framework es un esquema (un esqueleto, un patrón) para el desarrollo o la implementación de una aplicación. Estos son utilizados con el objetivo de promover buenas prácticas de desarrollo como el uso de patrones y la reutilización de código, posibilitando que no sea necesario perder tiempo en reinventar desde cero. Es posible que un framework defina una estructura para una aplicación completa, o bien sólo se centre en un aspecto de ella. (Fonseca, 2011)

1.9.6.1. Capa de presentación

DojoToolkit

Es un framework que contiene APIs (*Aplication Programming Interface*, Interfaz de programación de aplicaciones) y widgets (controles) para ayudar al desarrollo de aplicaciones web, utiliza Ajax para hacer trabajos asíncronos en la página y se compone de varios módulos: Dijit, Dojo y Dojox. Los complementos de Dojo son componentes pre empaquetados de código JavaScript, HTML y CSS que pueden ser usados para enriquecer aplicaciones web. También ofrece funciones para leer y escribir cookies, proporcionando en el lado cliente una abstracción llamada Dojo Storage. Dojo Storage permite a la aplicación web almacenar datos en el lado cliente, persistencia y seguridad. (Piñero Añon, 2011)

Spring WebFlow

Es un framework que permite manejar la navegación de la aplicación Web acoplándose a la plataforma de Spring MVC. Se utiliza cuando el caso de uso que se desarrolla presenta un flujo complejo o no lineal, y provee una definición de un lenguaje declarativo de flujos. Se puede definir como una secuencia de pasos o actividades que se realizan para llevar a cabo una determinada acción. La configuración del desarrollo del flujo es especificada a través de un archivo de configuración XML, permitiendo establecer reglas de navegación complejas de una manera sencilla. (Piñero Añon, 2011)

1.9.6.2. Capa de negocio

Spring MVC

Es un framework que tiene el objetivo de facilitar la construcción de aplicaciones Java, presentando un entorno diseñado para aumentar la productividad, liberando al desarrollador de tareas repetitivas y ayudándolo a hacer diseños más consistentes. Es utilizado para atender las peticiones web simples con navegación lineal a través de clases controladoras. Está diseñado como una serie de módulos que pueden trabajar independientemente uno de otro. Además intenta mantener un mínimo acoplamiento entre la aplicación y el propio framework. Brinda una limpia y clara separación entre las 3 capas arquitectónicas: Acceso a Datos, Negocio y Lógica de Presentación. (Fonseca, 2011)

1.9.6.3. Capa de acceso a dato

Hibernate

Es un framework dedicado al mapeo objeto-relacional para la plataforma Java, y a su vez un generador de sentencias SQL (*Standard Query Language*). Busca solucionar el problema de la diferencia entre el modelo orientado a objetos y el usado en las bases de datos modelo relacional mediante archivos declarativos. Permite la ejecución de consultas SQL y además posee un potente lenguaje de consulta de datos llamado HQL (*Hibernate Query Language*), que permite realizar consultas basándose en los objetos del negocio y no en las tablas de la base de datos. Brinda la posibilidad de generar bases de datos en cualquiera de los entornos soportados: PosgtreSQL, Oracle y MySql. (Piñero Añon, 2011)

1.10. Conclusiones parciales

En este capítulo con el análisis referente a los sistemas de mensajería financiera entre bancos mundiales se llegó a la conclusión de que resulta necesario desarrollar el sistema bajo la filosofía de software libre, para que así pueda ser utilizado por el estado cubano ya que el pago de licencias suele ser engorroso en temas económicos para Cuba. Con el estudio del modelo de desarrollo de software se podrán seguir detenidamente cada una de las fases establecidas en CEIGE para obtener un producto que responda a las necesidades del banco y que tenga la calidad requerida, utilizando las herramientas y tecnologías establecidas desde la primera fase de desarrollo del sistema Quarxo.

CAPÍTULO II: ANÁLISIS Y DISEÑO

2.1. Introducción

El siguiente capítulo muestra una descripción de los elementos esenciales a tener en cuenta para desarrollar el producto. Hace referencia a los principales diagramas relacionados con los procesos necesarios para llevar a cabo la gestión de los mensajes SWIFT, como es el caso de los diagramas de clases del análisis, diagramas de colaboración, diagramas de clases del diseño y diagramas de secuencia. También se describe la arquitectura previamente propuesta para el sistema Quarxo. De manera general se abarca todo el análisis y el diseño realizado para dar solución a la problemática inicialmente propuesta en el subsistema Mensajería SWFT.

2.2. Análisis

En esta etapa se realiza una descripción del entorno software que se quiere obtener, estableciendo los requisitos de todos los elementos del sistema. El objetivo de este flujo de trabajo es refinar y estructurar los requisitos obtenidos y traducirlos a una especificación que describe cómo implementar el sistema. El análisis consiste en obtener una visión del sistema que se preocupa de ver qué hace (Sommerville, 2005).

2.2.1. Modelado del negocio

Para obtener garantías de que el software desarrollado cumpla con su propósito, se llevó a cabo un estudio que permitiera comprender cómo funcionan los procesos de negocio en el área de Mensajería SWIFT perteneciente al BNC. Para lo cual se identificaron dos procesos específicos a analizar para su posterior desarrollo.

El primer proceso nombrado Actualizar encabezado a un mensaje SWIFT tiene como objetivo fundamental permitirle a los usuarios de la entidad bancaria, modificar el destino de los mensajes SWIFT previamente registrados en el sistema. Esta funcionalidad actualmente no se encuentra disponible en el sistema Quarxo, por lo que hace engorroso el trabajo de los clientes cuando al tener la necesidad de hacer algún cambio en el posible destino de los mensajes SWIFT deben volver a crear el mensaje y pierden los datos previamente insertados.

El segundo proceso nombrado Registrar mensaje SWIFT está enfocado en la incorporación de nuevos mensajes SWIFT al sistema como son: MT-299, MT-199, MT-999, MT-300, MT-103, MT-102, MT-750, MT-

754. También incluye el envío de mensajes SWIFT que no partan de alguna operación bancaria, con el objetivo de permitirles a los trabajadores de la entidad enviar mensajes informativos a otras instituciones bancarias. Con la incorporación de estos nuevos mensajes el sistema contaría con los mensajes SWIFT necesarios en el pago de las negociaciones y depósitos que entran y salen constantemente del BNC.

2.2.1.1. Diagrama de procesos de negocio

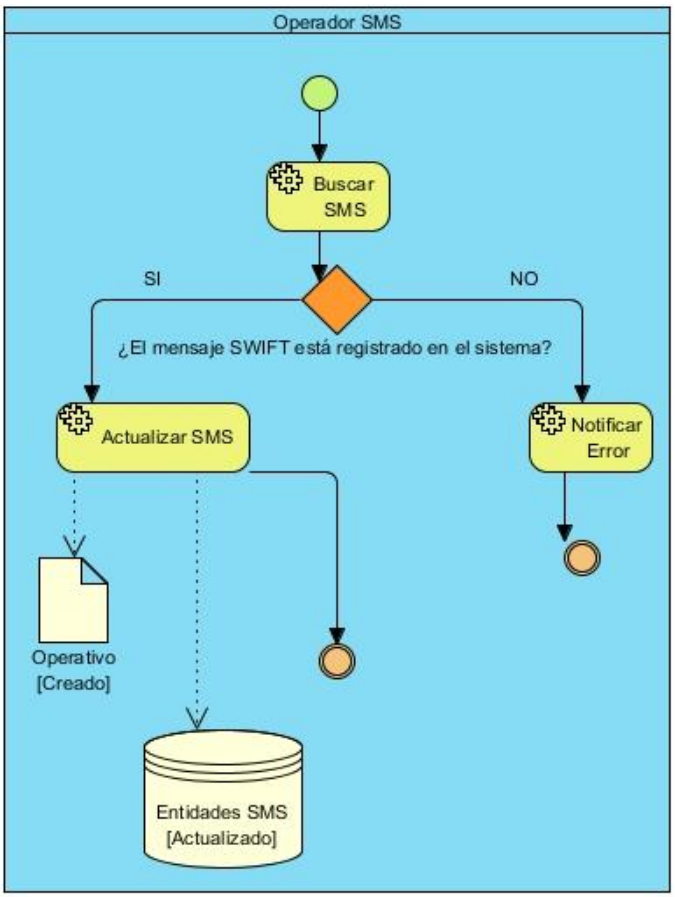


Ilustración 1: Diagrama correspondiente al proceso Actualizar encabezado a un mensaje SWIFT.

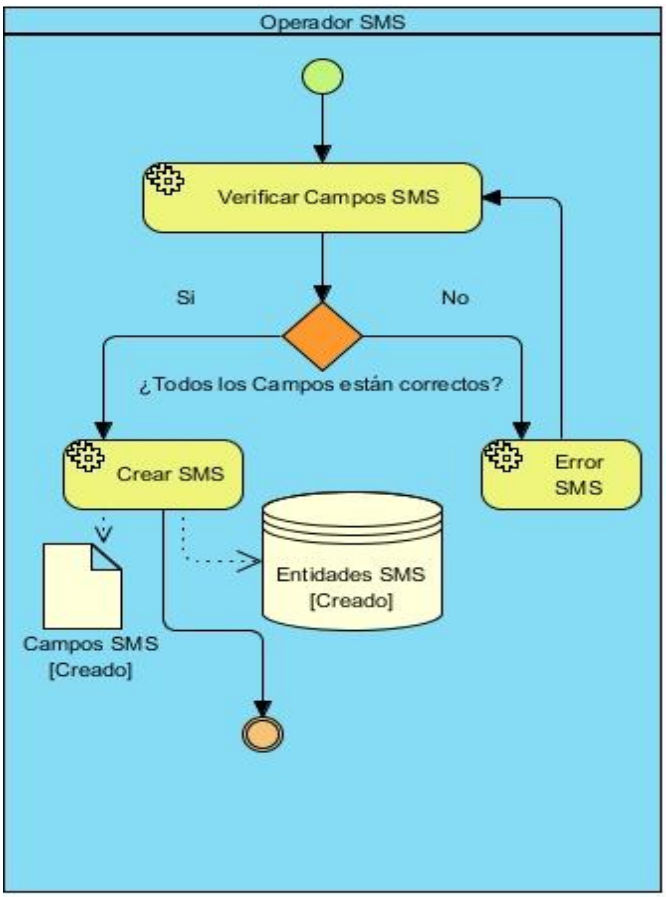


Ilustración 2: Diagrama correspondiente al proceso Registrar mensaje SWIFT

2.2.2. Requisitos del sistema

Definen el comportamiento que deberá tener el sistema, minimizando la posibilidad de errores y especificando de forma clara lo que el cliente desea. Mostrando así las cualidades, características, funciones y atributos que debe cumplir el sistema que se está desarrollando.

2.2.2.1. Requisitos funcionales

Para describir todas las interacciones que efectuarán los usuarios con el sistema de acuerdo a las necesidades existentes en el subsistema Mensajería SWIFT se definieron los siguientes requisitos funcionales:

RF1: Actualizar encabezado a un mensaje SWIFT.

El sistema permitirá actualizar el encabezado (destino) de los mensajes mediante la siguiente descripción de requisitos:

Precondiciones	El usuario se ha identificado y autenticado ante el sistema y tiene permisos para ejecutar esta acción. El mensaje debe de estar previamente registrado en el sistema.
Flujo de eventos	
Flujo básico Actualizar encabezado de un mensaje SWIFT	
1	Se buscar el mensaje SWIFT definido por alguno de los siguientes criterios de búsqueda: Identificador del mensaje SWIFT. Número del mensaje SWIFT. Importe Fecha
2	El usuario selecciona el mensaje y solicita actualizar encabezado.
3	El sistema muestra los campos del encabezado del mensaje seleccionado.
4	El usuario introduce los cambios y solicita aceptar.
5	El sistema valida que los datos sean correctos y muestra el mensaje de confirmación.
6	El usuario confirma el mensaje.
7	El sistema actualiza el encabezado del mensaje.
8	Concluye el requisito.
Pos-condiciones	
1	Se actualizó el encabezado del mensaje SWIFT seleccionado.

Flujos alternos		
Flujo alterno 5.a Información incompleta		
1	El sistema señala los campos vacíos y permite corregirlos.	
2	Volver al paso 4 del flujo básico.	
Pos-condiciones		
1	N/A.	
Flujo alternativo 4.a El usuario cancela la acción		
1	Concluye el requisito.	
Pos-condiciones		
1	No se registran los datos.	
Validaciones		
1	Se validan los datos.	
Relaciones	**Requisitos Incluidos**	N/A.
	Extensiones	N/A.
Conceptos	**Mensaje SWIFT**	Atributos del concepto que se utilizan en el requisito: Encabezado Utilizados internamente: N/A.
Requisitos especiales	N/A.	
Asuntos pendientes	N/A.	

Tabla 1: Descripción correspondiente al requisito Actualizar encabezado a un mensaje SWIFT.

RF2: Registrar mensaje SWIFT.

El sistema permitirá enviar mensajes que no inicien de operaciones contables.

Incluyendo los mensajes: 300 (depósitos), 750,754, 299, 999, 199, 103, 102 (negociaciones).

(Ver anexo 1).

RF3: Imprimir o no el mensaje SWIFT.

El sistema permitirá al usuario que cuando se arregle el mensaje, muestre la opción de imprimir o no, por si en ese instante se quedó algo por arreglar, ahorrando papel.

RF4: Permitir que mensajería se comporte como un flujo.

El sistema permitirá regresar al negocio para cualquier cambio que pueda surgir, sin que al volver al mensaje se pierdan los datos que no fueron modificados.

RF5: Gestionar permisos de usuarios a los mensajes.

El sistema permitirá asignar las funcionalidades requeridas a los usuarios autentificados en cuestión, o sea se le darán los permisos correspondientes a los usuarios para acceder a los requisitos que su rol especifique.

2.2.2.2. Requisitos no funcionales

Para describir las cualidades o propiedades con las que el sistema debe de contar se definieron los siguientes requisitos no funcionales:

RNF1: Usabilidad

- El sistema permitirá visualizar una descripción textual en los mensajes de error.
- El sistema permitirá ordenar las etiquetas de las funcionalidades en el menú según las dependencias de ejecución de negocio.
- Las etiquetas de cada funcionalidad y los campos de cada interfaz tendrán títulos asociados a su función de negocio.

RNF2: Adaptabilidad

- El sistema podrá ser operado desde el navegador web Mozilla Firefox 3.6.

RNF3: Madurez

- El sistema no permitirá la entrada de datos incorrectos.
- El sistema impondrá campos obligatorios para garantizar la integridad de la información que se introduce por el usuario.

RNF4: Atracción

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

2.2.2.3. Validación de requisitos

Para validar los requisitos se emplearon tres técnicas de validación de requisitos, las mismas son:

- Prototipos
- Revisiones de requisitos
- Generación de casos de prueba.

Para esto se efectuó la construcción de prototipos por cada requisito funcional, de forma tal que se visualizaran los diferentes datos que tendría cada una de las interfaces. Así el cliente pudo tener una idea de la estructura de la interfaz final del sistema.

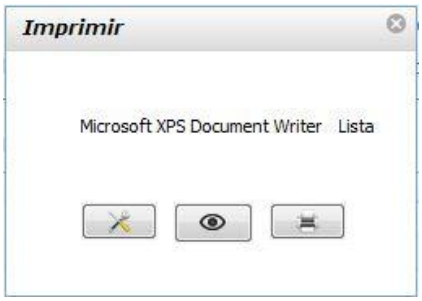


Ilustración 3: Prototipo para permitir Imprimir o no el mensaje SWIFT después de ser actualizado.

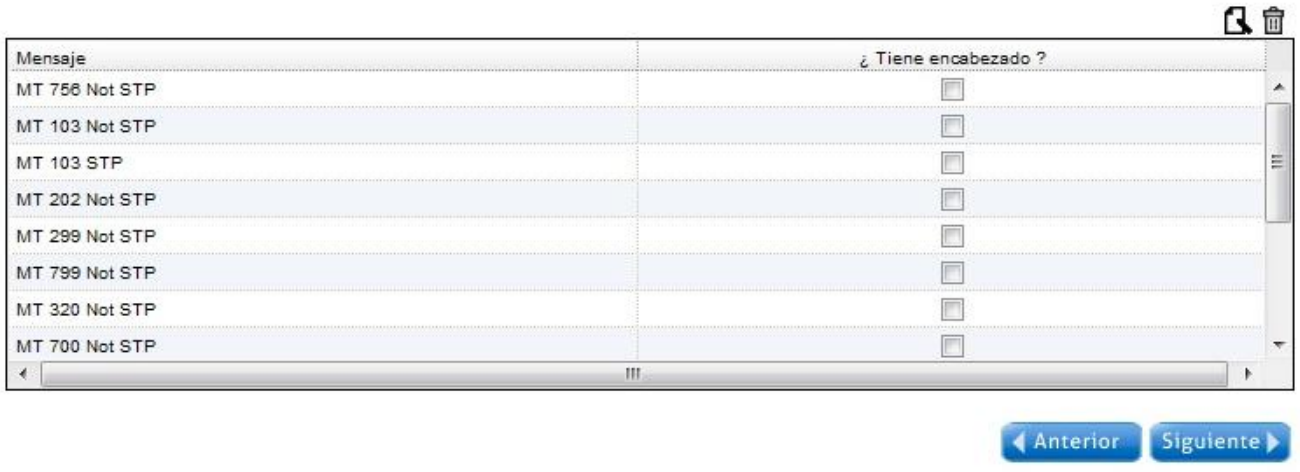


Ilustración 4: Prototipo de Interfaz de usuario para el requisito Registrar mensaje SWIFT.

La segunda técnica fue empleada al realizarse la descripción de cada uno de los requisitos funcionales, donde los clientes con ayuda de los desarrolladores, revisaron los requerimientos definidos para confirmar que realmente respondían a las necesidades y definen si el producto cumple con las expectativas esperadas, firmando un acta de conformidad que avala el cumplimiento satisfactorio de los objetivos del requisito (Ver anexo 2 y anexo 3).

La tercera y última técnica fue empleada al generarse casos de pruebas por cada uno de los requisitos funcionales (Ver anexo 4), donde a través de un conjunto de posibles escenarios se crearon varios juegos de datos a probar para validar que el requisito cumpliese con el propósito para el cual fue desarrollado. Por este medio se documentaron una serie de no conformidades en los artefactos creados, que ya han sido corregidas (Ver anexo 5).

2.2.3. Modelo de Datos

El almacenamiento de datos es considerado el corazón de un sistema de información, ya que la información obtenida mediante los datos almacenados normalmente es utilizada para administrar, planear, controlar o tomar decisiones en cualquier organización. Por ello se hace necesario realizar un diseño de datos equilibrado y con calidad que permita gestionar la información referente a los mensajes SWFT. Al usuario interactuar con los procesos que se llevan a cabo dentro del subsistema Mensajería, en determinadas ocasiones se efectúan cambios en alguno que otro campo del mensaje o se registran nuevos mensajes, lo que conlleva a interactuar directamente con las tablas guardadas en la base de datos y con las relaciones existentes entre ellas. A continuación se muestra la descripción de las tablas y el modelo de datos utilizado para lograr la interacción de los requisitos identificados con los datos almacenados en el sistema:

Las tablas o_formato_abstracto_SWIFT, o_secuencia_formato_SWIFT, o_campo_formato_SWIFT, o_grupo_componente_formato_SWIFT, o_subcampo_formato_SWIFT, o_funcion_formato_SWIFT, o_nodo_formato_SWIFT, o_bloque_formato_SWIFT y o_formato_SWIFT se encargan de responder por el almacenamiento del formato de los mensajes SWIFT. Por otra parte las tablas o_mensaje_SWIFT, o_numero_mensaje y o_estado_mensaje_swiftFIN se encargan de almacenar los mensajes SWIFT conformados y el estado en que se encuentran. Por último la tabla n_tag119 funciona como un identificador de los mensajes, pudiendo ser STP (Sistema de Transferencias y Pagos) y NOT STP.

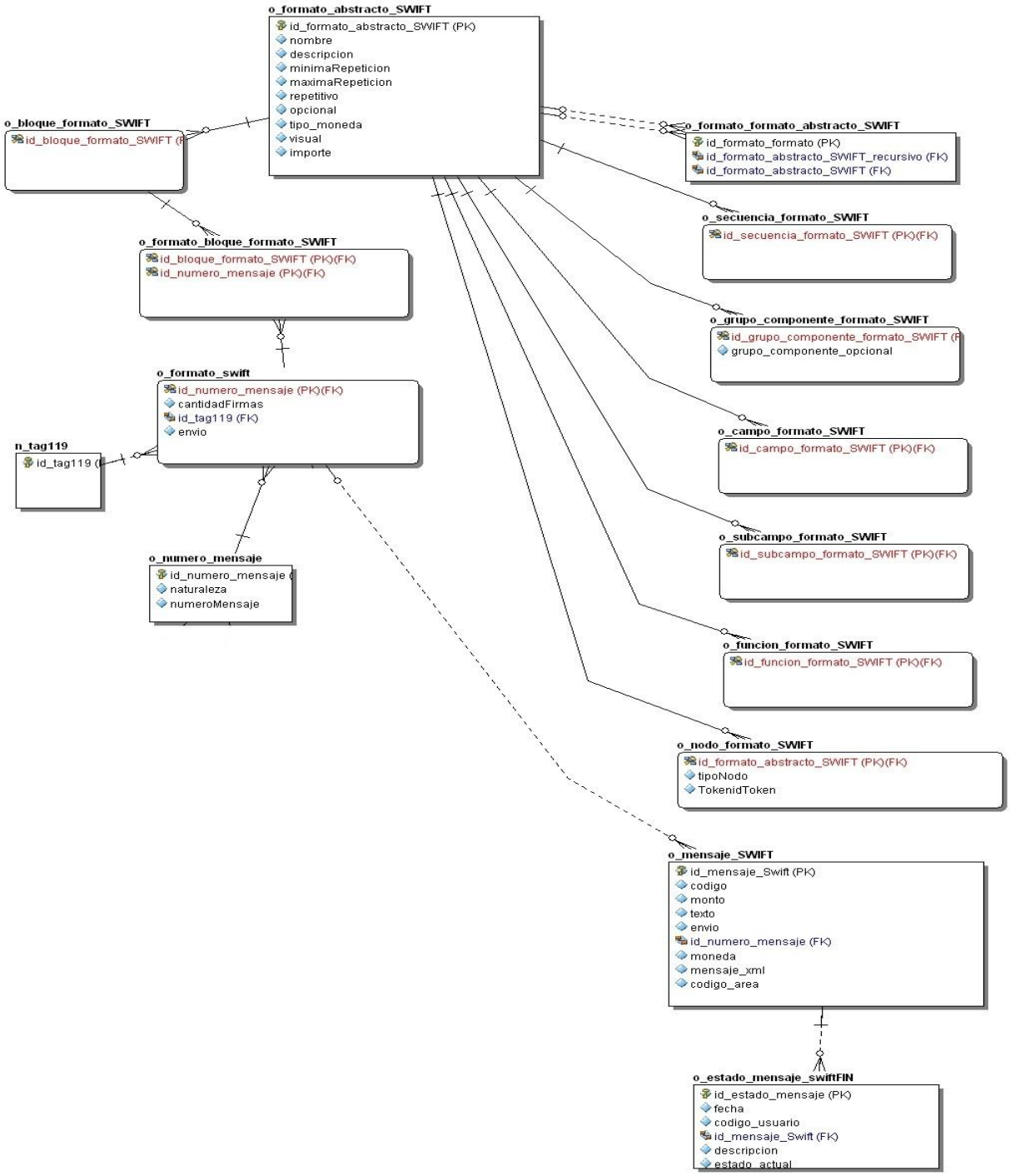


Ilustración 5: Modelo de datos correspondiente al subsistema Mensajería SWIFT.

2.3. Modelo de diseño

Es un modelo de un sistema que describe un grupo de objetos que interactúan entre sí, centrándose en cómo los requisitos según sus descripciones tienen impacto en el sistema. Es frecuentemente utilizado como entrada en las actividades de la implementación, ya que permite abstraerse en el contexto funcional del sistema para tener una idea plena de qué se va a hacer y cómo se va a hacer. En este se generan una serie de artefactos que actúan como una guía realizada por el diseñador e interpretada por el programador, para llevar a cabo la implementación del software basada en todas las peticiones especificadas por el cliente. (López, 2003)

2.3.1. Arquitectura propuesta para el sistema Quarxo

La arquitectura es el resultado de conformar con cierto número de elementos arquitectónicos una estructura adecuada que permita satisfacer las funcionalidades y requerimientos necesarios para el buen desempeño de un sistema. En este caso el sistema QUARXO cuenta con una arquitectura compuesta por tres capas básicas: Capa de presentación, Capa de negocio y Capa de acceso a datos, también contiene una capa transversal a las otras con los objetos del dominio. Donde se organizan jerárquicamente las capas, de manera que cada capa provee servicios a la capa superior y es servido por la capa inferior. (Iglesias, 2009)

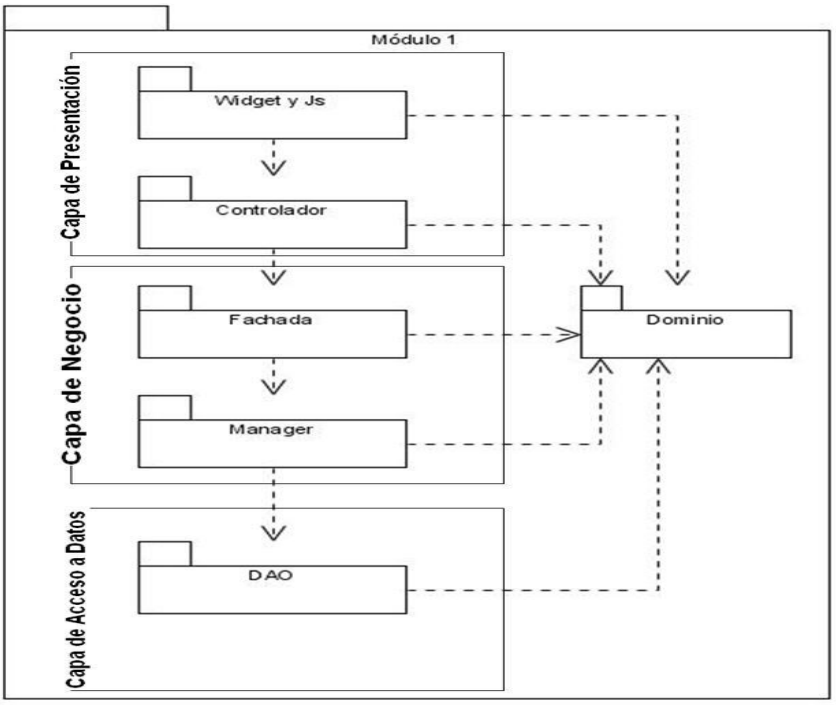


Ilustración 6: Estructura de las capas lógicas del sistema QUARXO.

2.3.1.1. Capa de presentación

Para el trabajo en la capa de presentación es utilizado Spring MVC (para recibir, controlar y enviar respuestas a las peticiones realizadas desde el cliente), bajo las adaptaciones realizadas al framework por el proyecto SIGEP, se utiliza además, Spring Web Flow (para representar y controlar los flujos complejos y reutilizables de la aplicación) y la librería Java Script Dojo (para generar las interfaces que interactuarán con el usuario) y Spring Security. La capa de presentación estará relacionada con la capa de negocios y la capa de dominio. (Iglesias, 2009)

2.3.1.2. Capa de negocio

En la capa del negocio se utilizan algunos módulos del framework Spring (para declarar y representar las relaciones de dependencia de cada una de las clases) tales como Spring Transaction, Spring AOP (*Aspect-Oriented Programming*, Programación Orientada a Aspectos) (para ejecutar las transacciones y la auditoría de cada uno de los métodos del negocio) y Spring Security (para asegurar las invocaciones a los métodos). Esta capa está divida en dos subcapas: **Facade** y **Manager**. Facade se encarga del

intercambio entre la capa de presentación y la capa de negocio, y no posee lógica de negocio. Manager por su parte tendrá la jerarquía de clases suficiente para implementar el negocio de la aplicación, utilizando la capa de acceso a datos para obtener los datos persistidos y la capa de dominio para generar las entidades del negocio. (Iglesias, 2009)

2.3.1.3. Capa de acceso a datos

En la capa de acceso a datos para la persistencia de los datos en la aplicación se utiliza el framework Hibernate (para realizar las operaciones básicas con la base de datos) y Spring JDBC (para interactuar con los procedimientos almacenados). Para desarrollar esta capa se utiliza el patrón DAO. Las interfaces de los DAOs contienen básicamente las operaciones de inserción, modificación, eliminación y localización de un objeto a partir de la llave primaria. (Iglesias, 2009)

2.3.1.4. 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; quedando estructurado de la siguiente forma:

- Subsistema: agruparán un conjunto de módulos que estén relacionados con los procesos que ejecutan.
- Módulo: agrupan un conjunto de casos de uso 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.

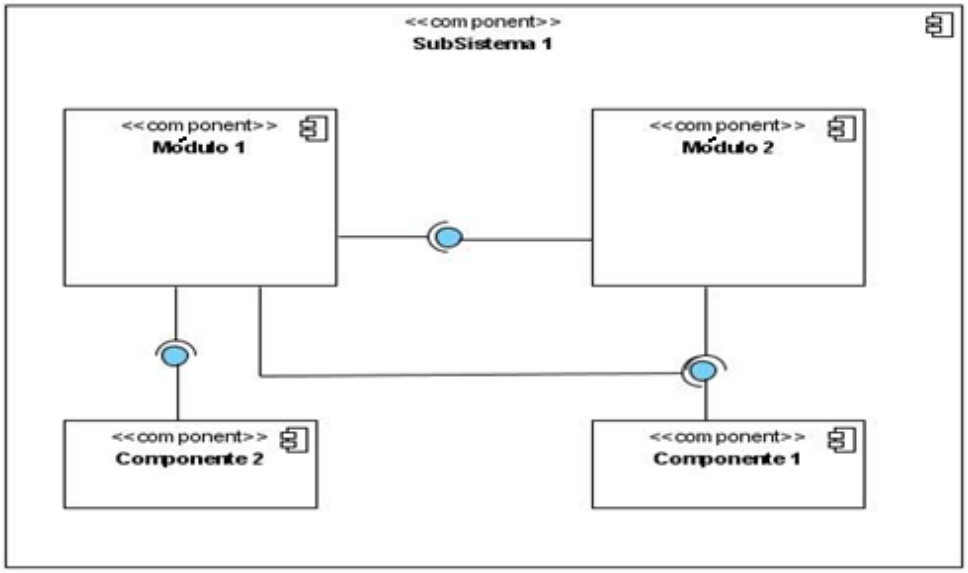


Ilustración 7: Arquitectura de un subsistema QUARXO.

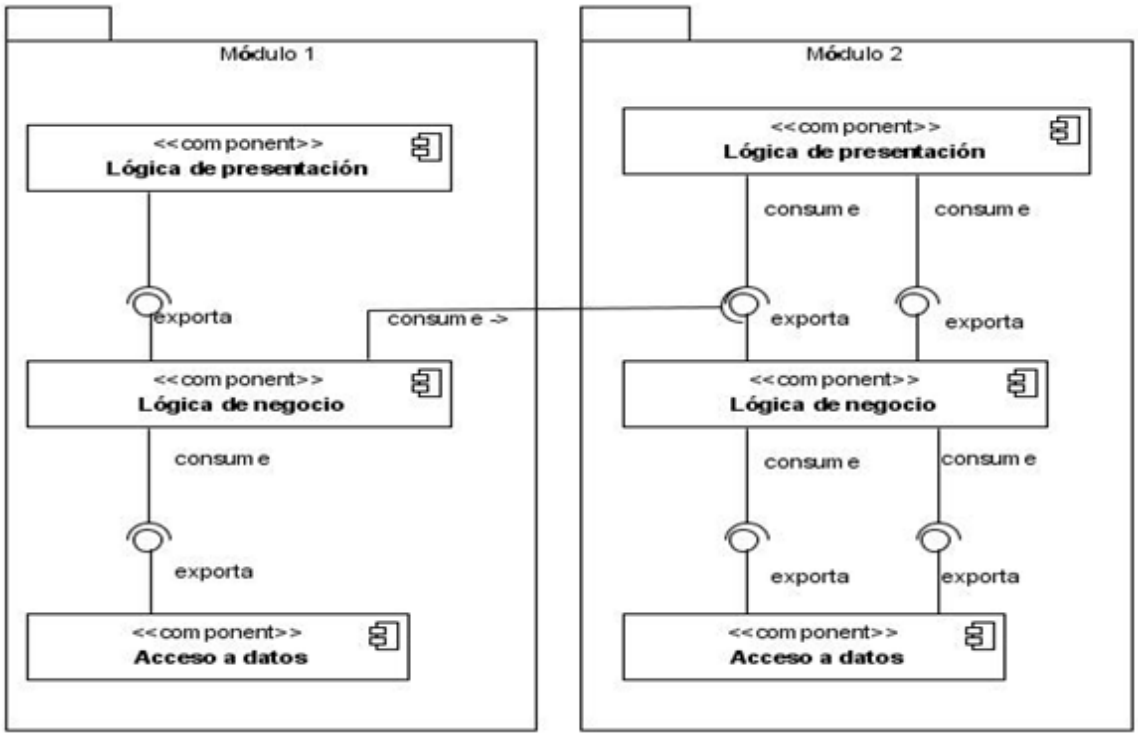


Ilustración 8: Arquitectura de los módulos en el sistema QUARXO.

2.3.2. Modelo de paquetes

Los paquetes permiten adquirir una organización estructural en los sistemas que se desarrollan, posibilitando descomponer los sistemas de gran tamaño en componentes más pequeños y manuables. También se encarga de representar las dependencias existentes entre estas descomposiciones lógicas para tener una clara idea de cómo interactúan entre sí. A través de este modelo se puede obtener y entender la jerarquía lógica y física que complementa el funcionamiento adecuado de los sistemas.

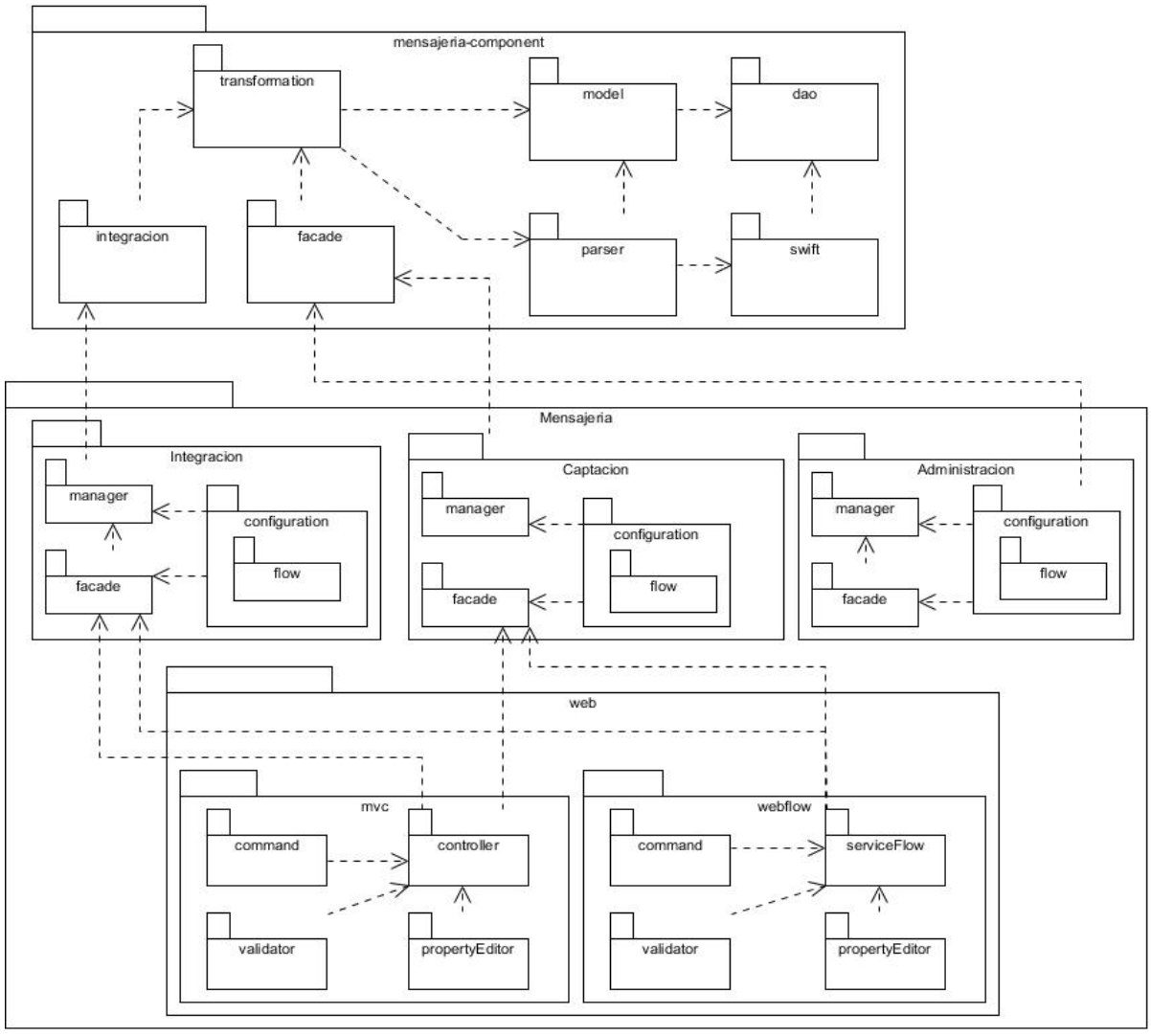


Ilustración 9: Modelo de paquetes correspondiente al subsistema Mensajería SWIFT.

Dentro de la estructura del subsistema Mensajería el paquete Mensajería-Component es el encargado de resolver todo el negocio (transformación y parseo de los mensajes, validación semántica y sintáctica de los mensajes), también contiene las clases que constituyen el modelo de datos y las clases DAO. Y presenta un número de clases encargadas de proporcionar puntos de entrada a las funcionalidades que brinda dicho componente, ubicadas en los paquetes Integración y Facade.

Para ayudar a la fácil comprensión de la aplicación se decidió organizar por criterios, en paquetes, las clases y ficheros de un módulo, agrupando las mismas según la función que cumplen.

A continuación se mencionan las clases que componen cada paquete.

Paquete configuration: en este paquete están 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: en este paquete están los flujos.xml del módulo correspondiente.

Paquete manager: en este paquete se encuentran las interfaces e implementaciones del negocio del módulo correspondiente.

Paquete facade: en este paquete se encuentran 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 la 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.

2.3.2.1. Diagrama de paquetes del módulo Captación

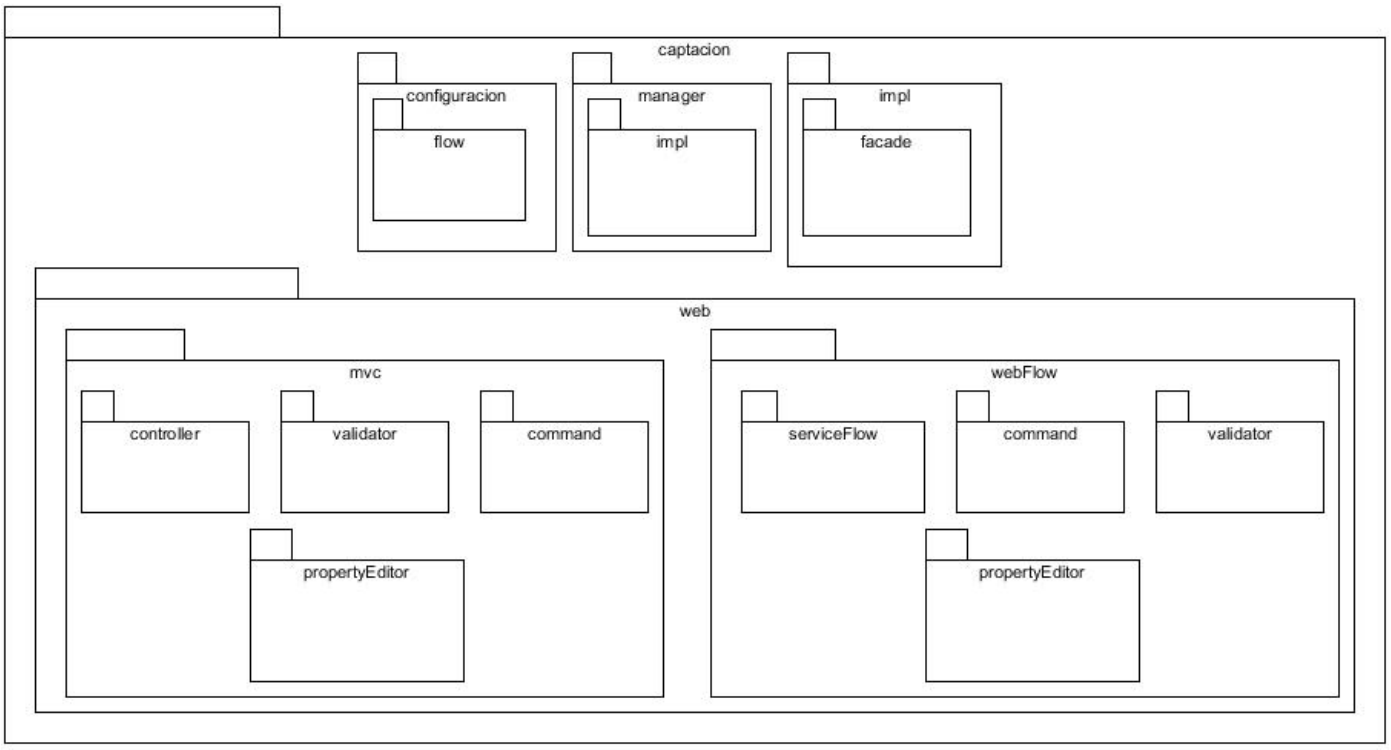


Ilustración 10: Modelo de paquetes correspondiente al módulo Captación.

En el caso del módulo Captación se decidió incorporar la funcionalidad Actualizar encabezado de un mensaje SWIFT. Utilizando para ello los paquetes para la web: mvc y webflow, para la lógica de presentación del servidor de ambos paquetes. Así como los paquetes configuración (para los ficheros XML de configuración), manager (para las interfaces e implementaciones del negocio) y facade (para las interfaces y la implementación de las clases que brindan un punto de entrada a las funcionalidades que se gestionan en el módulo).

El módulo Integración se encarga de la gestión de la funcionalidad Registrar mensaje SWIFT. Utilizando para ello los paquetes para la web: mvc y webflow, para la lógica de presentación del servidor de ambos paquetes. Así como los paquetes configuración (para los ficheros XML de configuración), manager (para las interfaces e implementaciones del negocio) y facade (para las interfaces y la implementación de las clases que brindan un punto de entrada a las funcionalidades que se gestionan en el módulo) (Ver anexo 6).

El módulo Administración por su parte se encargará de los recursos necesarios para la gestión de los permisos de usuarios según el rol que cumpla cada usuario en el sistema. Utilizando para ello los paquetes para la web: mvc y webflow, para la lógica de presentación del servidor de ambos paquetes. Así como los paquetes configuración (para los ficheros XML de configuración), manager (para las interfaces e implementaciones del negocio) y facade (para las interfaces y la implementación de las clases que brindan un punto de entrada a las funcionalidades que se gestionan en el módulo) (Ver anexo 7).

2.3.3. Diagramas de clases del diseño

Las clases de diseño reflejan un mayor nivel de detalle, se conciben para satisfacer los requisitos funcionales y no funcionales, teniendo en consideración la tecnología en la cual se implementará.

2.3.3.1. Diagrama de clases del diseño del módulo Captación

A continuación se muestra una parte del diagrama de clases del diseño del módulo Captación, en los anexos se podrá ver el diagrama completo, debido a que el mismo es muy extenso (Ver anexo 8). En este diagrama se evidencia la relación de las páginas clientes y los formularios con las clases que atienden sus peticiones.

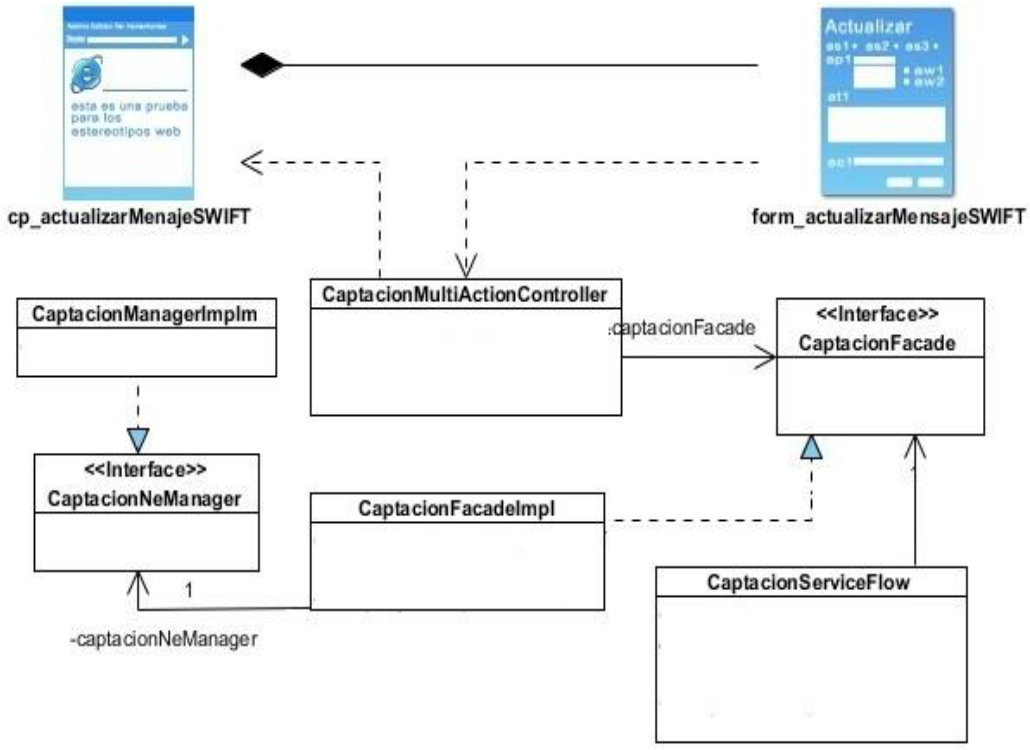


Ilustración 11: Diagrama de clases correspondiente al módulo Captación.

Para manejar las peticiones que se realizan desde la página del cliente, correspondientes a la actualización del encabezado de los mensajes se utiliza la clase **CaptacionMultiActionController**. Dicha entidad posee una serie de método usados para dar solución a la problemática planteada.

- ✓ **+buscarBanco** (request, response) : void
  Mediante este método se buscan los posibles bancos destinos a los que se puede enviar el mensaje.
- ✓ **+actualizarEncabezado** (request, response) : void
  Mediante este método se actualiza o modifica el destino al que se enviará el mensaje.

2.3.3.2. Diagrama de clases del diseño del módulo Integración

A continuación se muestra una parte del diagrama de clases del diseño del módulo Integración, en los anexos se podrá ver el diagrama completo, debido a que el mismo es muy extenso (Ver anexo 9). En este

diagrama se evidencia la relación de las páginas clientes y los formularios con las clases que atienden sus peticiones.

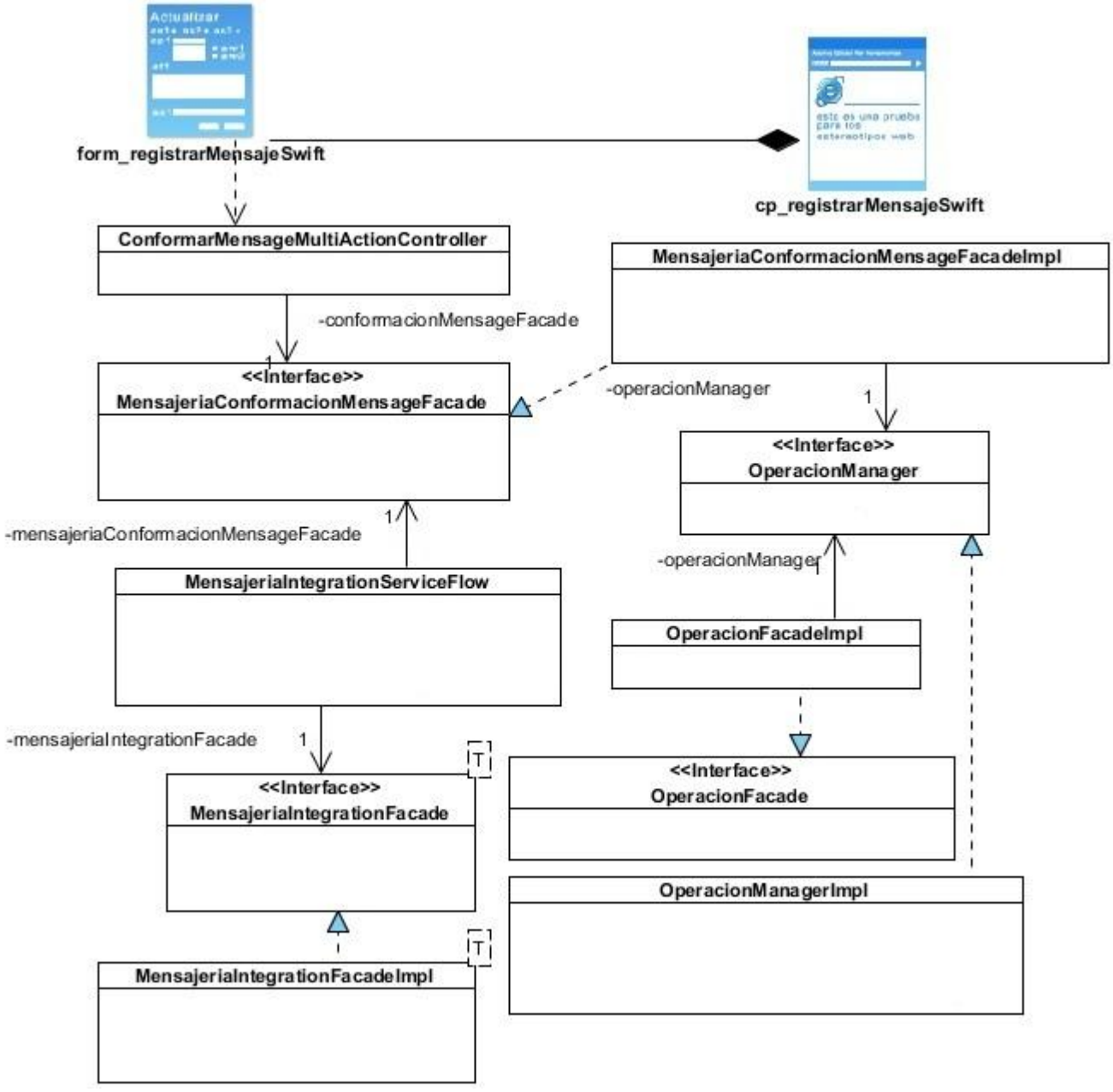


Ilustración 12: Diagrama de clases correspondiente al módulo Integración.

Para manejar las peticiones que se realizan desde la página del cliente, correspondientes a la creación de los mensajes SWIFT se utiliza la clase **MensajeriaIntegrationServiceFlow** y la clase **ConformarMensageMultiActionController**. Dicha entidad posee una serie de método usados para dar solución a la problemática planteada:

- ✓ **+procesarMessage** (context, datosNegocio : Object) : void
  Mediante este método se crea el nuevo mensaje SWIFT, según los datos introducidos por el usuario.
- ✓ **+prepararMensajes** (context) : void
  Este método se ejecuta cuando ocurre un error en el parseo del mensaje.
- ✓ **+prepararEncabezado** (context) : void
  Mediante este método se crea el encabezado del mensaje a través del banco destino seleccionado por el usuario.
- ✓ **+eliminarEncabezado** (context) : void
  Mediante este método se elimina el encabezado del mensaje previamente creado.
- ✓ **+parserMiniMessageText** (context) : void
  Mediante este método se parsean los datos introducidos en los campos del mensaje.
- ✓ **+buscarBanco**(request, response) : void
  Mediante este método se buscan los posibles bancos preestablecidos en el sistema.
- ✓ **+posiblesValoresDelCampoMoneda** (request, response) : void
  Mediante este método se buscan los posibles valores que puede tomar el campo moneda en el mensaje.
- ✓ **+posiblesValoresDelCampoAEditar** (request, response) : void
  Mediante este método se buscan los posibles valores que puede tomar el campo código (determinado por cuatro letras) y también los posibles valores del campo Crédito_Débito (determinado por C: para los Créditos y D: para los Débitos).

2.3.4. Diagramas de interacción (Secuencia)

Los diagramas de secuencia de un sistema permiten describir la interacción entre los objetos de la aplicación y comprender los mensajes enviados y recibidos entre dichos objetos. Son considerados herramientas valiosas para presentar y examinar las operaciones, permitiendo 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.

2.3.4.1. Escenarios

Describe la secuencia de pasos de éxito que se produce en las posibles interacciones, proporcionando diferentes tipos de información en dependencia de los niveles de detalle del sistema. Son técnicas para obtener requerimientos para los puntos de vista debido a que se centran en las interacciones, permitiendo tener una vista adecuada del procedimiento que se sigue en cada requerimiento.

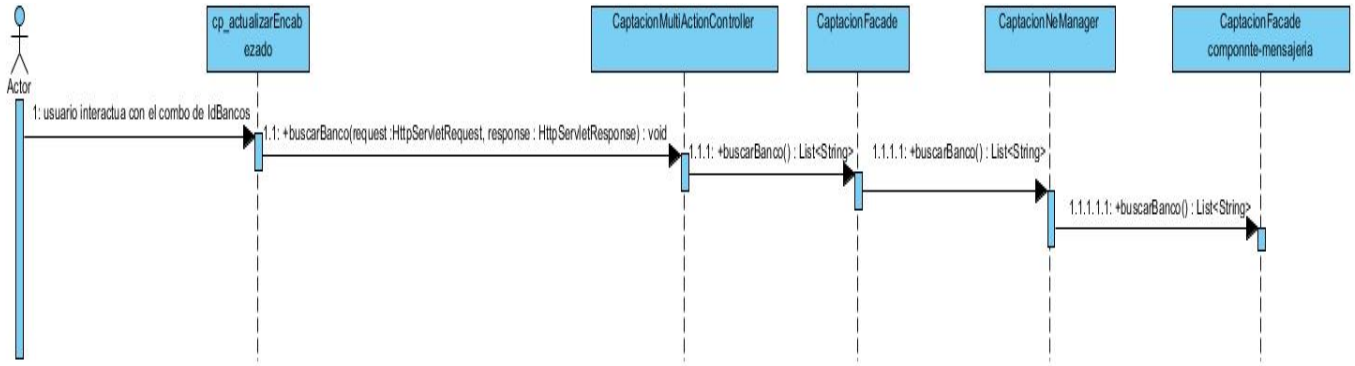


Ilustración 13: Diagrama de secuencia de la operación Actualizar encabezado. Primer escenario.

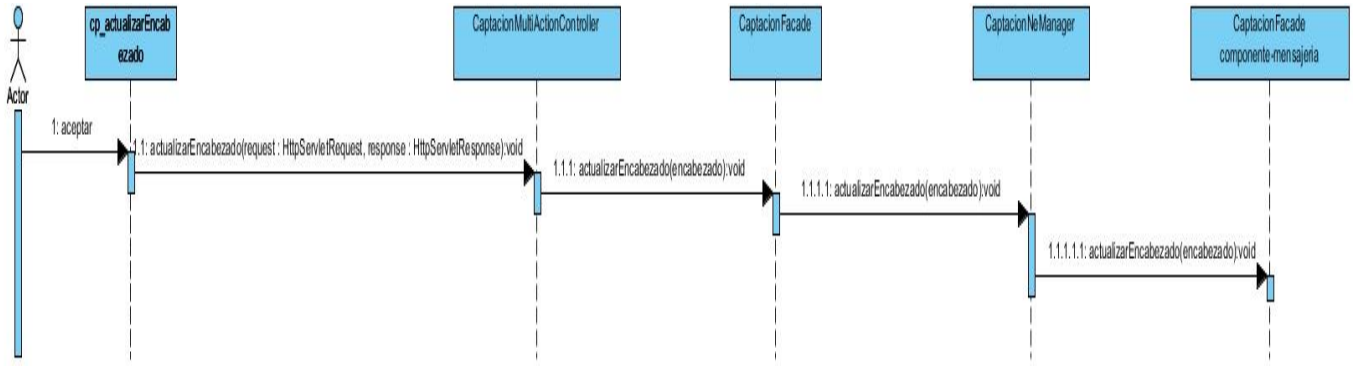


Ilustración 14: Diagrama de secuencia de la operación Actualizar encabezado. Segundo escenario.

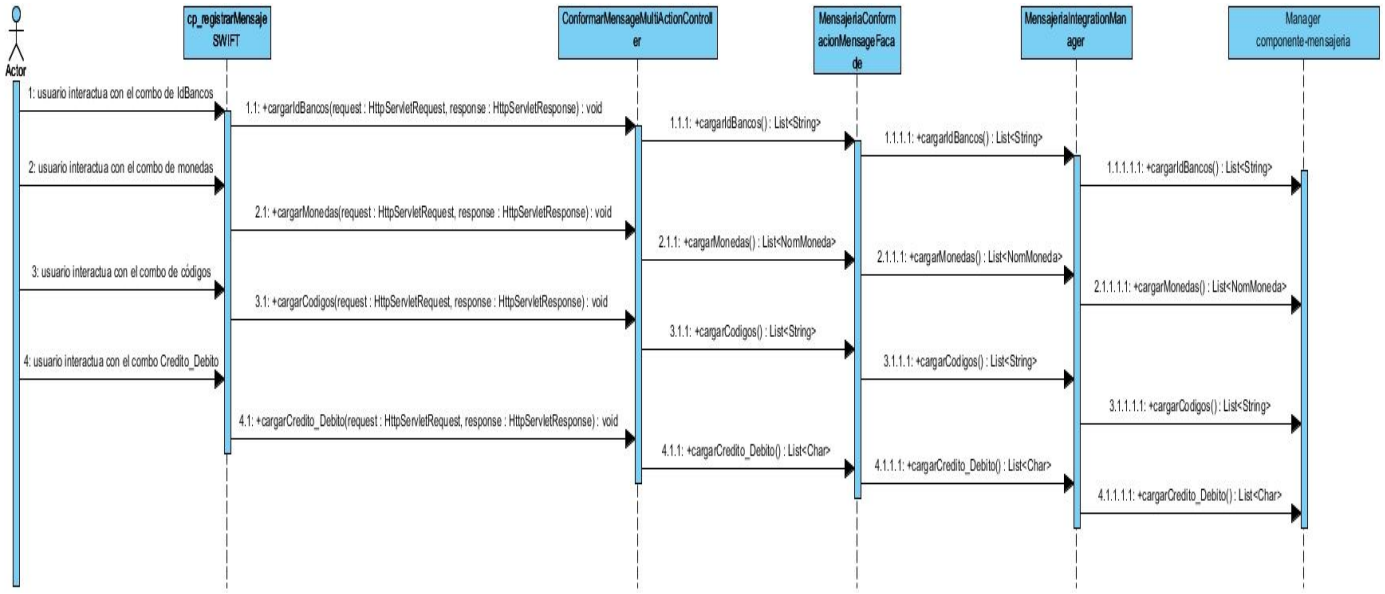


Ilustración 15: Diagrama de secuencia de la operación Registrar mensaje SWIFT. Primer escenario.

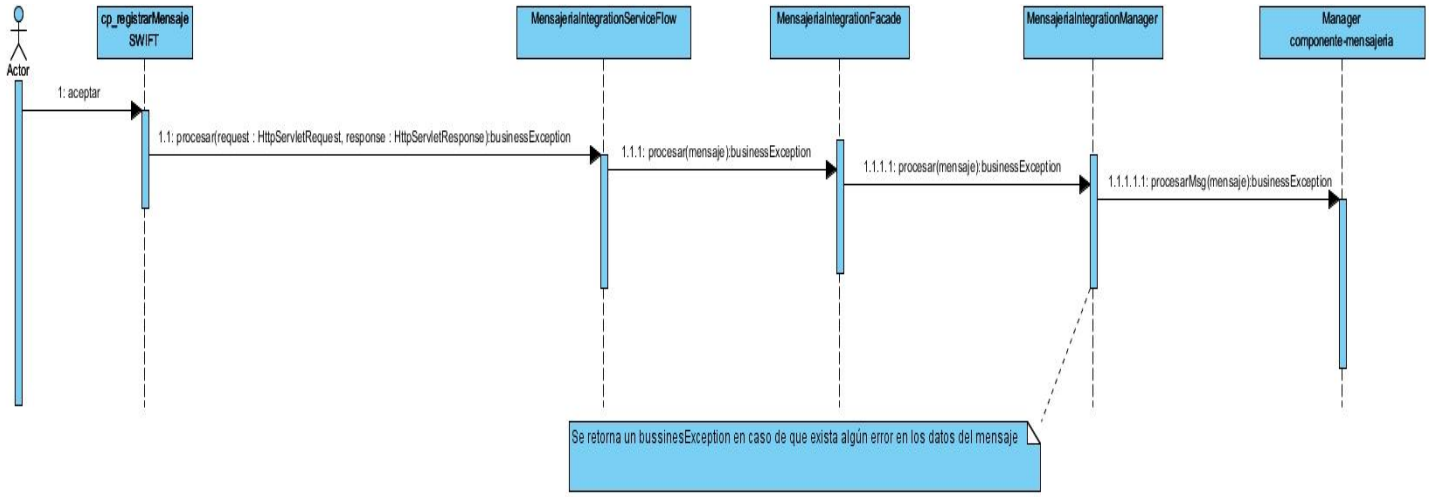


Ilustración 16: Diagrama de secuencia de la operación Registrar mensaje SWIFT. Segundo escenario.

2.3.5. Patrones de diseño utilizados

2.3.5.1. Patrones GRAPS

Controlador: se utiliza a través de las clases controladoras de los módulos, teniendo la responsabilidad de escuchar y responder a las peticiones realizadas por la presentación y de comunicarse con el negocio.

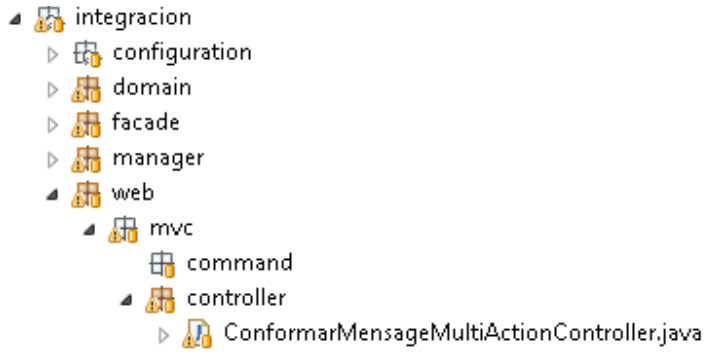


Ilustración 17: Ejemplo del controlador en el módulo Integración.

Bajo Acoplamiento: se utiliza a través de las interfaces e implementaciones, como puede ser la interfaz *CaptacionFacade* y su implementación *CaptacionFacadeImpl* permitiendo que clases como *CaptacionMultiActionController* se relacionen únicamente con ellas para realizar sus operaciones, reduciendo el impacto de cambios posteriores en el negocio del sistema, logrando que el código sea más fácil de entender, mantener y reutilizar.

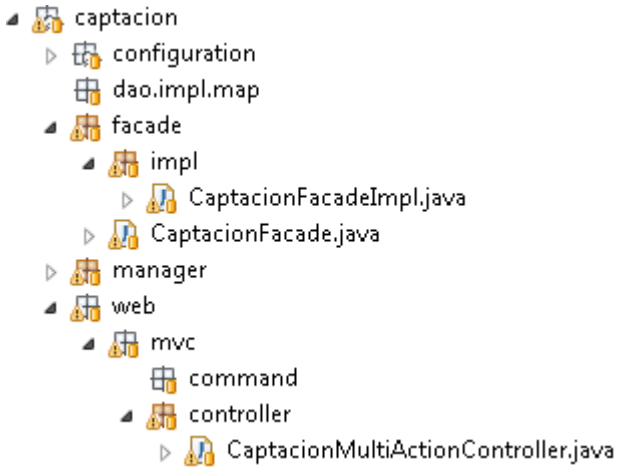


Ilustración 18: Ejemplo del bajo acoplamiento en el módulo Captación.

2.3.5.2. Patrones GOF

Fachada: se utiliza a través de la interfaz *CaptacionFacade* 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. Permitiendo reducir la dependencia entre clases y ofreciendo un punto de acceso al resto de clases, sin ocultar las clases. Por medio de su utilización se conoce qué clases del subsistema son responsables de qué peticiones delegando así las peticiones a los objetos correspondientes.

Cadena de Responsabilidad: se utiliza a través del flujo que surge cuando la vista realiza una petición a alguna información que se encuentran en la base de datos, siendo atendida posteriormente por los controladores, luego la fachada, seguidamente el manager y por último el DAO.

2.3.5.3. Patrones DAO

Para abstraer y encapsular todos los accesos a la fuente de datos se utilizan los DAOs que se muestran a continuación. Implementando cada uno de los DAOs una serie de métodos declarados en la interfaz DAO correspondiente e incluyendo también nuevos métodos necesarios para manejar la conexión con la fuente de datos.

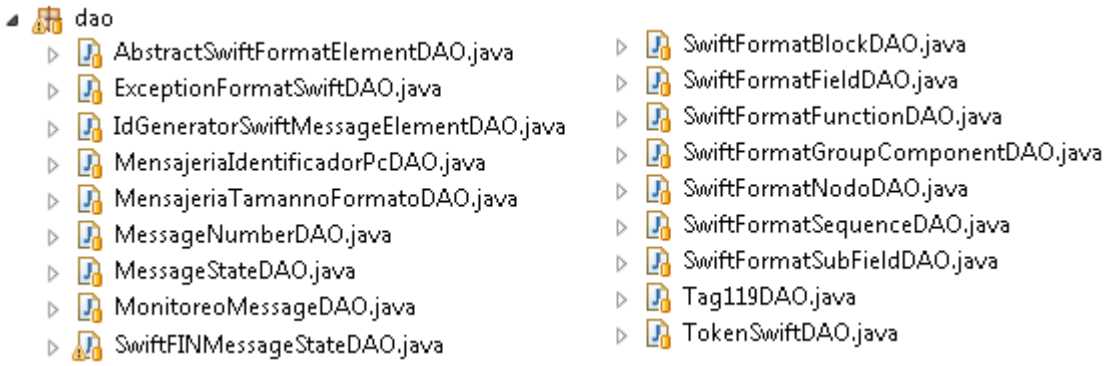


Ilustración 19: Aplicación del patrón DAO.

2.3.6. Validación del diseño

Las métricas definen la calidad durante el desarrollo del software a través de la medición de ciertas características o atributos como costo, tiempo y tamaño, que son inherentes a aspectos del software. Esto permite al ingeniero evaluar la calidad durante el desarrollo del sistema.

Para la validación del diseño correspondiente a la validación del sistema se empleó la métrica TOC (Tamaño Operacional de las Clases) para evaluar los siguientes atributos de calidad:

Responsabilidad: consiste en la responsabilidad asignada a una clase en el marco de modelado de la problemática propuesta.

Complejidad de implementación: consiste en el grado de dificultad que tiene implementado un diseño de clases determinado.

Reutilización: consiste en el grado de reutilización presente en una clase o estructura de clase.

Para la evaluación de cada atributo se establece un rango de valores que determinan la complejidad en la aplicación del TOC. A continuación se muestran los rangos de valores y la complejidad por cada uno de los atributos que se analizan:

	Categoria	Criterio
Responsabilidad	Baja	< =Prom.
	Media	Entre Prom. y 2* Pom.
	Alta	> 2* Prom.
Complejidad implementación	Baja	< =Prom.
	Media	Entre Prom. y 2* Pom.
	Alta	> 2* Prom.
Reutilización	Baja	> 2*Prom.
	Media	Entre Prom. y 2* Pom.
	Alta	<= Prom.

Ilustración 20: Rango de valores para la evaluación de los atributos

Para la aplicación de la métrica se utilizaron seis clases con un total de ochenta y cuatro procedimientos, lo que arrojó un promedio de procedimientos de catorce como se observa en las siguientes imágenes:

$$\frac{Total\ de\ Clases}{Cantidad\ de\ Procedimientos} = \frac{84}{6} = 14$$

Total de clases	**6**
Promedio de procedimientos	**14**

No	Módulo	Clase	Cantidad de Procedimientos
1	integracion	ControllerMensajeriaIntegrationServiceFlow	43
2	integracion	ConformarMensageMultiActionController	21
3	integracion	OperacionManagerImpl	5
4	integracion	AbstractMensajeriaIntegrationTemplateManager	10
5	integracion	MensajeriaIntegrationFlowHandler	2
6	integracion	MensajeriaIntegrationWebflowUtils	3
	Total		84

Ilustración 21: Datos de validación del diseño.

Al probar los datos presentados se obtuvieron los siguientes resultados en cada uno de los atributos de calidad:

Reutilizació	Cantidad de clases	Promedio
Alta	4	66,66666667
Media	2	33,33333333
Baja	0	0

Ilustración 22: Resultado de Reutilización.

Responsabilidad	Cantidad de clases	Promedio
Baja	4	66,66666667
Media	2	33,33333333
Alta	0	0

Ilustración 23: Resultado de Responsabilidad.

Complejidad	Cantidad de clases	Promedio
Baja	2	33,33333333
Media	1	16,66666667
Alta	0	0

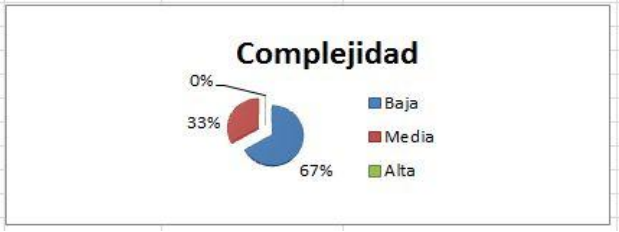


Ilustración 24: Resultado de Complejidad.

Los resultados obtenidos demuestran que los atributos de calidad de las clases se encuentran en un nivel satisfactorio; de manera que se puede confirmar la elevada reutilización con un 67% y cómo se reducen la responsabilidad y la complejidad de implementación. Lo que contribuye a un bajo acoplamiento ya que existen pocas dependencias entre las clases debido a que hay una baja responsabilidad en el atributo de calidad. Por otra parte también se concibe una alta cohesión ya que las responsabilidades están acordes a la información que controlan debido a que existe una alta reutilización.

2.4. Conclusiones parciales

En este capítulo se analizaron los procesos necesarios para complementar el flujo de las actividades que se realizan por medio de los mensajes SWIFT dentro de la gerencia del banco, permitiendo así visualizar cómo funcionan los procesos Actualizar encabezado a un mensaje SWIFT y Registrar mensaje SWIFT. El análisis de la arquitectura del proyecto QUARXO y el diseño de los módulos Captación, Integración y Administración realizado en este capítulo, facilitó la comprensión de la estructura de paquetes existentes en el subsistema, así como un mejor entendimiento del diseño del mismo. Todo el material expuesto en este capítulo conforma las bases a seguir para lograr una implementación acorde a la estructura que posee actualmente el sistema en explotación e incorporando las necesidades surgidas para dar solución al problema en cuestión

.

.

CAPÍTULO III: IMPLEMENTACIÓN Y VALIDACIÓN DE LA SOLUCIÓN PROPUESTA

3.1. Introducción

El siguiente capítulo muestra descripción de los elementos esenciales en la etapa de implementación y pruebas del sistema. Se describirán los estándares de codificación definidos para la escritura del código. Contiene la interacción de los componentes relacionados con los requerimientos detectados, representada mediante el diagrama de componentes. También se muestra el código fuente de uno de los principales métodos con la descripción del mismo. Finalmente se probará la solución mediante pruebas de caja negra y caja blanca, en aras de encontrar errores que imposibiliten el correcto funcionamiento del producto.

3.2. Implementación

La implementación es el principal flujo de trabajo en la fase de construcción. Describe cómo los elementos del modelo de diseño se implementan en términos de componentes, o sea toma el resultado del modelo de diseño para generar el código final del sistema. 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. (Piñero Añon, 2011)

3.2.1. Diagrama de despliegue

El diagrama de despliegue permite mostrar la arquitectura en tiempo de ejecución del sistema respecto a hardware y software, muestra los recursos necesarios a la hora de poner en marcha el sistema para que todo funcione correctamente.

Para desplegar el sistema Quarxo se necesita una PC Cliente que tenga instalada una máquina virtual de Java, que se encuentre conectada a una Impresora mediante USB y al Servidor de aplicación mediante HTTPS (*Hypertext Transfer Protocol Secure*, Protocolo Seguro de Transferencia de Hipertexto). Este servidor de aplicaciones debe tener instalada una máquina virtual de Java y el Apache Tomcat 6.0, además debe estar conectada mediante el protocolo TCP/IP al servidor de Base de Datos Microsoft SQL Server 2005. (Iglesias, 2009) A continuación se muestra el diagrama de despliegue predefinido para el sistema.

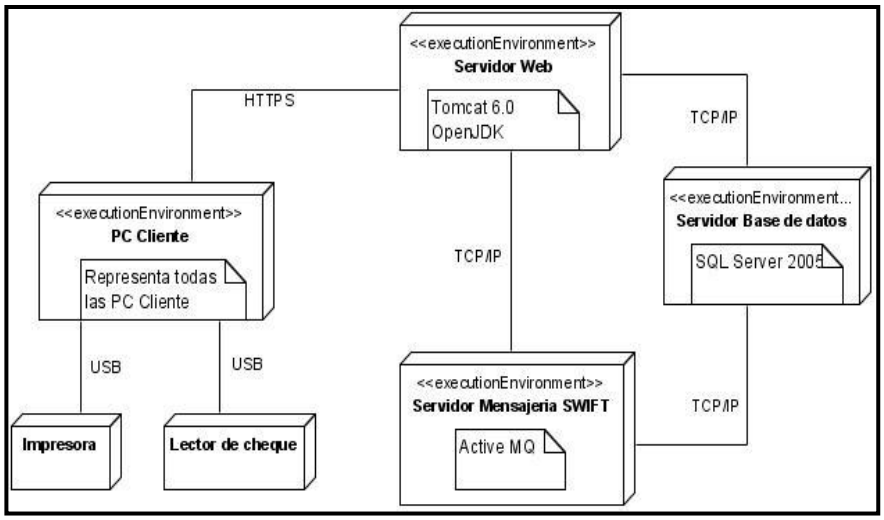


Ilustración 25: Modelo de despliegue.

3.2.2. Diagrama de componentes

Los diagramas de componentes describen los elementos físicos del sistema y sus relaciones. Permite modelar la vista estática del sistema mostrando las dependencias lógicas entre un conjunto de componentes de software. Los componentes pueden ser de código fuente, librerías, binarios o ejecutables. (Fonseca, 2011)

A continuación se explican de manera general los componentes utilizados:

El componente **Web** contiene los componentes **mvc** y **webFlow** ofreciendo un conjunto de clases que constituyen la lógica de presentación del servidor para Spring MVC y para Spring WebFlow respectivamente.

En el caso del componente **Mensajería-Component** engloba un grupo de clases necesarias para resolver todo el negocio (transformación y parseo de los mensajes, validación semántica y sintáctica de los mensajes), también contiene las clases que constituyen el modelo de datos y las clases DAO.

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.

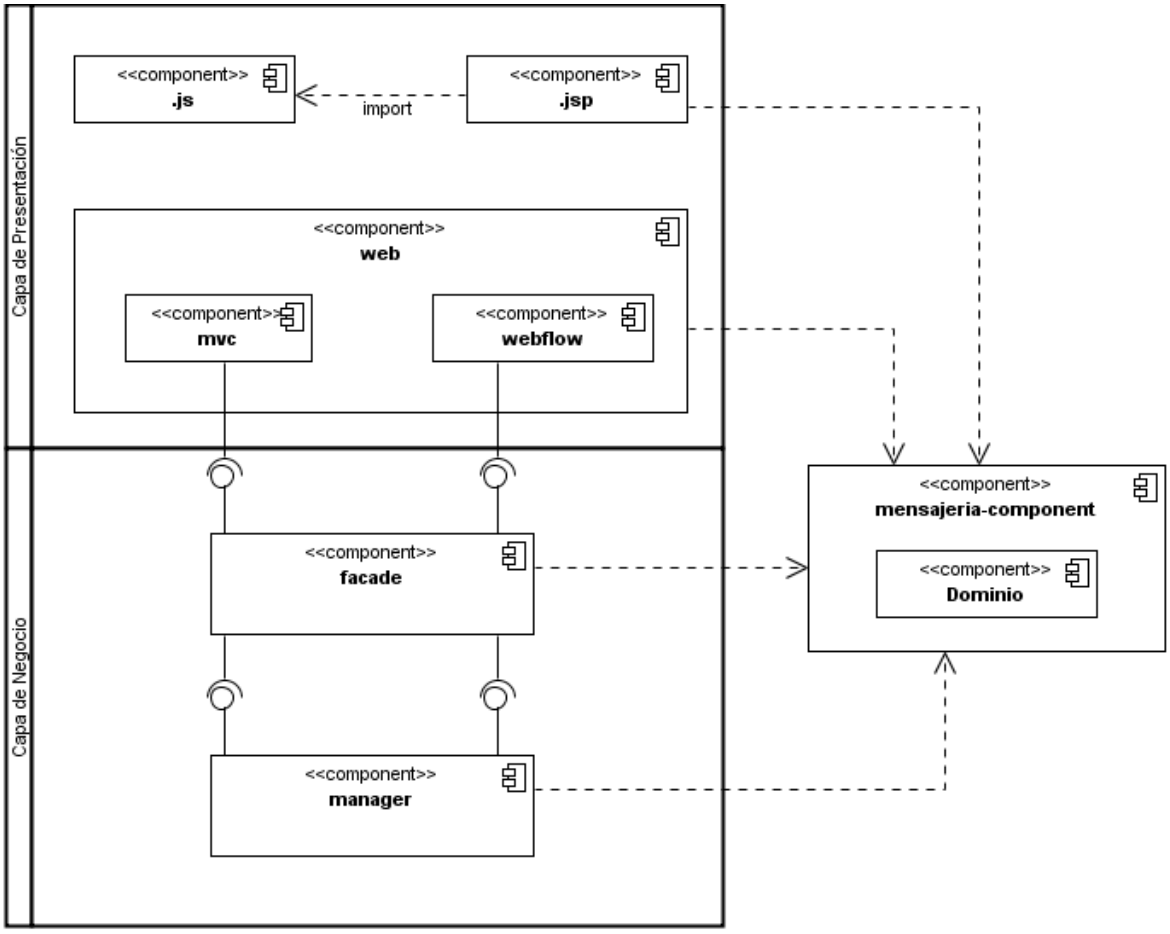


Ilustración 26: Diagrama de componentes.

3.2.3. Estándares de codificación

Un estándar de codificación comprende 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 estilos. El uso de estándares asegura la legibilidad del código entre distintos desarrolladores, permite la guía para el mantenimiento o actualización del sistema, además de que facilita la portabilidad entre plataformas y aplicaciones (Piñero Añon, 2011).

Notaciones

Se utiliza en la implementación del sistema Quarxo, la notación ***PascalCasing*** para la nomenclatura de las clases y la notación ***CamelCasing*** para nombrar las variables y métodos presentes en cada una de ellas.

PascalCasing: Plantea que los identificadores y nombres de variables, métodos y funciones que están compuestos por múltiples palabras juntas, iniciarán cada palabra con letra mayúscula.

CamelCasing: Plantea que los identificadores y nombres de variables, métodos y funciones que están compuestos por múltiples palabras juntas, iniciarán cada palabra con letra mayúscula excepto la primera palabra.

3.2.3.1. Convenciones de Nomenclatura

Los nombres de las clases comienzan con la primera letra en mayúscula y el resto con minúscula, en caso de ser una palabra compuesta se empleará la notación PascalCasing.

Ejemplo: CaptacionManager.

Los nombres de los métodos y los atributos de las clases, así como los nombres de ficheros de código javascript y sus funciones y variables internas comienzan con la primera letra en minúscula, en caso de que sea un nombre compuesto se empleará notación CamelCasing

3.2.3.2. Nomenclatura según el tipo de clase

Las clases que se encuentran dentro del paquete controller, se nombran adicionándoles el nombre del controlador de Spring del cual heredan al final del nombre de la clase.

Ejemplo: CaptacionMultiActionController.

Las clases que se encuentran dentro del paquete flowHandler se les coloca la palabra FlowHandler después del nombre de la clase. El nombre debe indicar el objetivo del flujo que se quiere personalizar.

Ejemplo: ActualizarMensajeFlowHandler

Las clases que se encuentran dentro del paquete serviceFlow se les agrega la palabra ServiceFlow después del nombre de la clase.

Ejemplo: CaptacionServiceFlow.

Las clases que se encuentran dentro del paquete facade se les agrega después del nombre la palabra Facade. Para el subpaquete impl se les nombra igual que la interfaz que implementan y se le agrega la palabra Impl.
Ejemplo: CaptacionFacade, CaptacionFacadeImpl.

Las clases que se encuentran dentro del paquete manager se les agrega después del nombre la palabra Manager. Para el sub paquete impl se les nombra igual que la interfaz que implementan y se le agrega la palabra Impl.
Ejemplo: CaptacionManager, CaptacionManagerImpl.

3.2.4. Descripción de las principales clases.

A continuación se muestran las clases más utilizadas en la implementación de la funcionalidad Actualizar encabezado de un mensaje SWIFT. Para la funcionalidad Registrar mensaje SWIFT (Ver anexo 10).

Nombre: CaptacionMultiActionController	
Tipo de clase: Controladora	
Atributo	**Tipo**
captacionFacade	CaptacionFacade
buscarMultiAction	BuscarMultiActionController
Para cada responsabilidad:	
Nombre:	**Descripción**:
+buscarBanco (request, response) : void	Mediante este método se buscan los posibles bancos destinos a los que se puede enviar el mensaje.
+actualizarEncabezado (request, response) : void	Mediante este método se actualiza o modifica el destino al que se enviará el mensaje.
listarMensajesConformados (HttpServletRequest	Permite obtener un listado con todos los

request, HttpServletResponse response)	mensajes que se han confirmado como aceptados en el sistema, es decir, que no tienen errores.
cancelarMensajeConf(HttpServletRequest request, HttpServletResponse response)	Permite cancelar un mensaje elaborado previamente.
Nombre: CaptacionFacadeImpl	
Tipo de clase: Facade	
Atributo	**Tipo**
captacionNeManager	CaptacionNeManager
Para cada responsabilidad:	
Nombre:	**Descripción**:
obtenerSwiftMessageElement(**int** idMensaje)	Permite obtener un mensaje a partir del id del mismo.
listaMensajeSwiftMTConformados(**int** estado, String usuario)	Permite listar todos los mensajes conformados del sistema a partir del estado en que estos se encuentren.
actualizarMensajeConformado(SwiftMessageElement message, Usuario usuario)	Permite actualizar un mensaje elaborado previamente.
actualizarMensajePrimeraFirma(**int** idMensaje, Usuario user)	Permite el cambio de estado de los mensajes a primera firma.
cancelarMensajeConformado(**int** idMensaje, Usuario user)	Permite cancelar un mensaje anteriormente elaborado.

3.2.5. Aspectos de la implementación

3.2.5.1. Framework Spring WebFlow

Para el control de flujos de la aplicación se utilizó el framework Spring WebFlow en la funcionalidad Registrar mensaje SWIFT correspondiente al módulo Captación. A continuación se muestra una descripción del funcionamiento del flujo en base a los requerimientos de dicha funcionalidad.

Primeramente se declara la variable ***mrt*** que representa el objeto mensaje que será utilizado a lo largo de todo el flujo.

```
<var name="mrt" class="cu.uci.finixubnc.mensajeria.integracion.configuracion.flow.conformarMensaje"/>
```

La clase ***mensajeriaIntegrationServiceFlow*** se utiliza para evitar la interacción directa del flujo con el negocio y contiene las funcionalidades necesarias para el mismo haciendo función de fachada.

Las funcionalidades declaradas en el flujo no necesitan declararse con los parámetros debido a que reciben ***RequestContext*** como único parámetro y Spring WebFlow es capaz de reconocerlo.

El flujo comienza con la vista ***opcionesMensaje*** que contiene las dos posibles transiciones que se pueden generar a partir de la decisión que tome el usuario.

```
<view-state id="opcionesMensaje">
        <transition on="crearMensaje" to="mensagesPorOperacion"></transition>
        <transition on="cancelarCrearMensaje" to="endDecision">
                <evaluate expression="mensajeriaIntegrationServiceFlow.cancelarEnvioMensaje
                        (flowRequestContext)">
                </evaluate>
        </transition>
</view-state>
```

Si el usuario decide crear el mensaje, se direcciona a la vista ***mensajePorOperacion*** que contiene las transacciones a partir de los posibles eventos que se puedan generar por el usuario, constituyendo el estado más importante del flujo.

```
view-state id="mensagesPorOperacion">
        <on-render>
                <evaluate expression="mensajeriaIntegrationServiceFlow.getMessages
                        (idOperacion, flowRequestContext)" result="viewScope.listMensajes" result-
                        type="java.lang.String" />
                <evaluate expression="mensajeriaIntegrationServiceFlow.getPosiblesDestinos
                        (flowRequestContext)" />
```

```
        </on-render>
        <transition on="registrarE" to="introducirEncabezado">
                <evaluate expression="mensajeriaIntegrationServiceFlow.destinosDeUnMensaje
                        (flowRequestContext)">
                </evaluate>
        </transition>
        <transition on="eliminarE" to="eliminarEncabezado" />
        <transition on="submit" to="verificarProcesamiento">
                <evaluate expression="mensajeriaIntegrationServiceFlow.procesar
                                (flowRequestContext, idOperacion,datosNegocio)" />
        </transition>
        <transition on="atras" to="opcionesMensaje"></transition>
</view-state>
```

Una vez generando el evento ***submit*** se invoca el método: ***procesar (flowRequestContext, idOperacion, datosNegocio)*** el cual se encarga de crear el formato del mensaje que se desea enviar.

```
public void procesar(RequestContext context, int idOperacion,
                        Object datosNegocio) {

        ObjectMessageXmlEntidad messageXmlEntidad = new ObjectMessageXmlEntidad();
        String xmlTem = (String) context.getFlowScope().get("xmlEntidad");
        Reader xmlEntidad = new StringReader(xmlTem);
        messageXmlEntidad.setBusinessData(xmlEntidad);
        List<Encabezado> encabezados = (List<Encabezado>) context
                .getFlowScope().get("listaEncabezados");
        List<MessageXmlEntidadCommand> listaDeEncabezadosXML = new ArrayList
                <MessageXmlEntidadCommand>();

        for (Encabezado encabezado : encabezados) {
                MessageXmlEntidadCommand messageXml = new MessageXmlEntidadCommand();
                Reader readerHead = ConvertToXml.convertToXml(encabezado, "header");
                messageXml.setHeaderMsg(readerHead);
                messageXml.setNumber(encabezado.getNummensage());
                messageXml.setSufix(encabezado.getTag119());
                messageXml.setSend(encabezado.isSend());
                messageXml.setRepeticionTotal(encabezado.getRepeticionTotal());
                listaDeEncabezadosXML.add(messageXml);
        }
        context.getFlowScope().put("erroresMensajes","");
        messageXmlEntidad.setListaDeEncabezados(listaDeEncabezadosXML);
        Usuario usuario = UserHandler.getUser();
        HashMap mapMessage;
        try {
                mapMessage = mensajeriaIntegrationFacade.procesar(datosNegocio,
                        idOperacion, messageXmlEntidad, usuario);

                if (mapMessage == null) {
```

```
                context.getFlowScope().put("mrt", null);
                context.getFlowScope().put("exception", null);
            } else {
                MessageResultTransformation messageResultTransformationList =
                (MessageResultTransformation)mapMessage.get(idOperacion);

                if (messageResultTransformationList.isErrorResult()) {
                    context.getFlowScope().put("errorResult", true);
                } else {
                    context.getFlowScope().put("errorResult", false);
                }
                context.getFlowScope().put("mrt", messageResultTransformationList);
            }
            } catch (BusinessException e){
                tratarException(context, e);
            } catch (SwiftRuntimeException e){
                e.printStackTrace();
            }
        }
```

Luego de crearse el formato del mensaje se pasa al siguiente estado de decisión:

```
<decision-state id="verificarProcesamiento">
        <if test="flowScope.errorResult == true" then="mostrarErrorValidacion"
            else="comprobarMRT" />
</decision-state>
```

En el cual, de haberse encontrado errores a la hora de procesar el formato del mensaje, se direcciona el flujo hacia la vista ***mostrarErrorValidacion*** notificando el error, de lo contrario se pasaría al estado de decisión:

```
<decision-state id="comprobarMRT">
        <if test="flowScope.mrt != null" then="registroDeMensajes" else="endDecision" />
</decision-state>
```

En este estado de decisión se comprueba que el formato del mensaje no esté nulo, en el caso de que sea así se iría al estado ***endDecision*** para terminar el flujo, de lo contrario el flujo direccionaría a la vista ***registroDeMensajes*** en la cual se llena el mensaje con los valores que el usuario desea introducir porque no se obtuvieron de la operación contable, una vez introducido todos los valores el usuario generaría el evento ***submitAllMessage***

```
<transition on="submitAllMessage" to="comprobarMRT">
        <evaluate expression="mensajerialntegrationServiceFlow.procesarMessage
(flowRequestContext,datosNegocio)">          </evaluate>
</transition>
```

Una vez disparado este evento se comprueba nuevamente que el mensaje no tenga valor nulo y se procesa el mensaje. El estado final del flujo devuelve el control al proceso contable de negocio direccionando hacia la página principal.

```
<end-state id="endMVC" view="externalRedirect:servletRelative:../../common/home.htm" />
```

3.3. Prueba

En el proceso de desarrollo de un software normalmente hay grandes posibilidades de que puedan existir posibles fallos de implementación, calidad, o usabilidad de un software. Por lo que se hace necesario dentro de este proceso incluir alguna actividad que permita verificar y revelar la calidad del producto que se está creando. Las pruebas permiten comprobar el grado de cumplimiento de las especificaciones que se implementan en el sistema. Para demostrar el funcionamiento eficiente de los módulos implementados se realizaron pruebas unitarias, con las cuales se probó el correcto funcionamiento del código (2011).

3.3.1. Pruebas de caja blanca

Las pruebas de caja blanca son las que se llevan a cabo en la parte interna del software, específicamente sobre el código fuente. Se basan en el examen minucioso de los detalles procedimentales. Se comprueban los caminos lógicos del sistema generando casos de prueba que ejerciten las estructuras condicionales y los bucles Se identifican dos formas de realizar las pruebas de caja blanca (Tahchiev, 2010):

- **Pruebas estáticas de caja blanca**: es el proceso que cuidadosamente y metódicamente revisa el diseño del software, la arquitectura o el código para encontrar defectos sin necesidad de ejecutar el código. Esto algunas veces se refiere a un análisis estructural.
- **Pruebas dinámicas de caja blanca**: en estas pruebas se revisa dentro de la “caja”, se examina el código y se observa este mientras se ejecuta. Utiliza la información que se obtiene al observar qué hace el código, cómo trabaja, para así determinar qué probar, qué no probar y cómo aproximarse a las pruebas.

Para realizar pruebas de caja blanca en el sistema se seleccionaron las pruebas dinámicas con la utilización del framework JUnit, este contiene un conjunto de librerías (Ver anexo 11) que permiten evaluar si el funcionamiento de cada uno de los métodos de la clase es el esperado. En la realización de estas pruebas se escogen distintos valores de entrada para examinar cada uno de los posibles flujos de

ejecución del programa y cerciorarse de que se devuelven los valores de salida adecuados. Si la clase cumple la especificación, entonces JUnit devolverá que el método de la clase pasó exitosamente la prueba. En caso de que el valor esperado sea diferente al retornado por el método durante la ejecución, JUnit devolverá un fallo en el método correspondiente.

A continuación se muestra el análisis realizado a la clase, controladora **CaptacionMultiactionController**, especificando los elementos usados para la gestión de la prueba, comenzando por captación-contexto.xml (Ver anexo 12), el cual mapea todas las referencias necesarias para ejecutar los métodos a probar, como es el caso de **ActualizarEncabezado**, dentro del subsistema Mensajería, este devuelve un model and view vacio.

```
public ModelAndView ActualizarEncabezado(HttpServletRequest request,
        HttpServletResponse response) {

    int referencia = Integer.parseInt(request.getParameter("ref"));

    if (referencia != 0) {

        MensajeriaIdentificadorPc datos = mensajeriaFacade
                .findByID(referencia);

        String nombre = datos.getNombrePc();
        String codigo = datos.getCodigoPc();

        Map model = new HashMap();
        model.put("id", referencia);
        model.put("nombre", nombre);
        model.put("codigo", codigo);

        return new ModelAndView("ActualizarEncabezado", model);
        }
        return new ModelAndView("ActualizarEncabezado");

    }
```

Ilustración 27: Método Actualizar encabezado del controlador

Primeramente se creó un objeto de la clase donde se encuentra el método a probar (controlador) y luego se crearon los mocks correspondientes para simular los parámetros de entrada (mockhttpServletRequest y mockhttpServletResponse), debido a que recibe como parámetros dos objetos de tipo HttpServlet (request y response).

```
public class conformarMultiActionController_Junit_Test extends TestCase{
    private CaptacionMultiActionController controlador;

    private MockControl<HttpServletRequest> controlHttpServlet;
    private HttpServletRequest mockhttpServletRequest;
    private MockControl<HttpServletResponse> controlHttpResponse;
    private HttpServletResponse mockhttpServletResponse;
    private MockControl<HttpSession> controlHttpSession;
    private HttpSession mockHttpSession;


    @Before
    public void setUp() throws Exception {
        ApplicationContext context = new ClassPathXmlApplicationContext(
        "classpath:cu/uci/finixubnc/junit/captacion-context.xml");

        controlador = (CaptacionMultiActionController) context.getBean("CaptacionMultiActionController");
```

Ilustración 28: Declaración de las variables de la prueba.

Durante la realización de la prueba al método, se preparan los parámetros de pruebas necesarios al método **ActualizarEncabezado**. Dichos parámetros serán pasados por el mockHttpServletRequest. Posteriormente se realiza la condición de prueba a través del método assertEquals, este evalúa si existen datos o no en el "map" enviado a la vista, comprobando en caso satisfactorio de que no existan dichos datos puesto que para el valor de prueba obtenido mediante el atributo "ref" el cual es cambiado por el valor numérico "0" y mostrando en la paleta (color rojo en vez de verde) en caso contrario un error al ejecutar el método.

```
public void testActualizarEncabezado() {
    mockhttpServletRequest.getParameter("ref");
    controlHttpServlet.setReturnValue("0");
    controlHttpServlet.replay();
    ModelAndView expeResult = controlador.ActualizarEncabezado
    (mockhttpServletRequest, mockhttpServletResponse);
    Map map = expeResult.getModel();
    assertEquals(new HashMap(),map);
}
```

Ilustración 29: Parámetros del método a probar.

Al ejecutar el framework JUnit se obtuvo el resultado de la prueba realizada a todos los métodos de la clase **CaptacionMultiactionController**, el cual fue satisfactorio como se muestra a continuación:

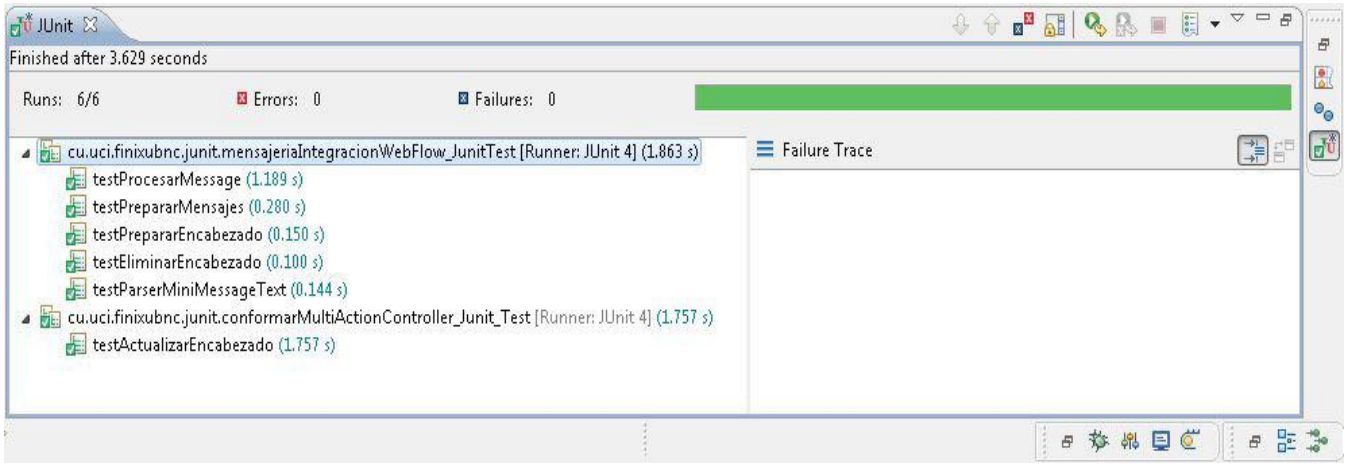


Ilustración 30: Resultados de las pruebas aplicadas a los métodos.

En la imagen se evidencia la factibilidad del método ya que se obtuvieron un total de 0 errores en la ejecución de la prueba. Además la paleta derecha representada de color verde indica que el proceso llevado a cabo a la clase cumplió con el caso definido, de lo contrario la paleta hubiese sido de color rojo.

3.3.2. Pruebas de caja negra

Las pruebas de caja negra son las que se llevan a cabo sobre la interfaz del software, o sea, los casos de prueba pretenden demostrar que las funciones del software son operativas, que la entrada se acepta de forma adecuada, obteniéndose un resultado correcto y la integridad de la información externa se mantiene. (2011)

Estas pruebas permiten encontrar:

- Funciones incorrecta o ausente.
- Errores de interfaz.
- Errores en estructuras de datos o en accesos a las bases de datos externas.
- Errores de rendimiento.
- Errores de inicialización y terminación.

Entre las técnicas para desarrollar la prueba de caja negra se encuentran:

- **Técnica de la partición de equivalencia**: divide el campo de entrada en clases de datos que tienden a ejercitar determinadas funciones del software.

- **Técnica del análisis de valores límites**: prueba la habilidad del programa para manejar datos que se encuentran en los límites aceptables.
- **Técnica de grafos de causa-efecto**: permite al encargado de la prueba validar complejos conjuntos de acciones y condiciones.

La técnica utilizada en la realización de las pruebas de caja negra fue partición de equivalencia ya que esta permite la reducción del número total de casos de prueba que hay que desarrollar. Con el diseño de casos de prueba de caja negra se obtienen un conjunto de pruebas, cuya misión es encontrar defectos y errores en el sistema. Cada caso de prueba brinda un número de valores de entradas en las pruebas, condiciones de ejecución y resultados esperados de las mismas para verificar una determinada funcionalidad del sistema. Se diseñaron casos de prueba por cada una de las funcionalidades del sistema (Ver anexo 4).

3.3.3. Resultados de las pruebas realizadas al sistema

Luego de poner en práctica los métodos de prueba, como parte de las pruebas internas realizadas a los módulos Captación, Integración y Administración se obtuvieron resultados satisfactorios desde el punto de vista interno y funcional, atendiendo al correcto comportamiento del mismo ante diferentes situaciones.

Se detectaron mediante las pruebas ejecutadas una serie de no conformidades en el subsistema (Ver anexo 5), estas fueron revisadas y corregidas. Es importante destacar que los módulos implementados fueron probados y liberados por el departamento de calidad del centro (Ver anexo 11), donde para la revisión se emplearon un total de tres iteraciones para lograr el resultado de cero no conformidad y tres tipo esenciales de pruebas: pruebas exploratorias, pruebas funcionales y pruebas de regresión.

3.4. Validación de la variable de investigación

La investigación desarrollada plantea como idea a defender que: “Si se desarrolla el subsistema Mensajería SWIFT versión 2 se facilitará la creación y procesamiento de los mensajes en el BNC simplificando el proceso en una sola aplicación.”, por lo que con la integración de las nuevas funcionalidades detectadas, el sistema permite crear todos los mensajes para la gestión financiera del banco desde la aplicación Quarxo sin necesidad de utilizar la aplicación SISCOM para los mensajes SWIFT que son enviados. La utilización de los dos sistemas traía como consecuencia el aumento del

tiempo empleado para el envío de mensajes SWIFT ya que los empleados del banco tardaban alrededor veinticinco minutos en crear y enviar un mensaje. Al Quarxo brindar la posibilidad de enviar todos los mensajes SWIFT desde la aplicación se reduce el tiempo empleado para este proceso a unos ocho minutos, haciendo muchísimo más interactivo y fácil el trabajo dentro de la gerencia del BNC. A continuación se grafica la comparación de tiempos para un antes y un después de la implementación realizada:

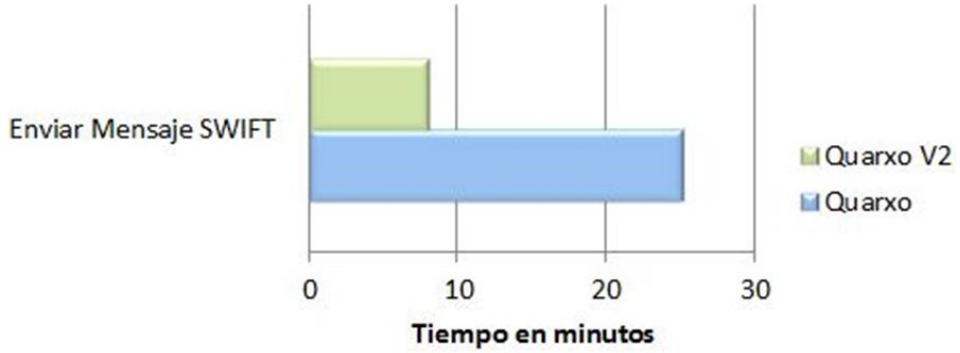


Ilustración 31: Tabla comparativa para tiempos de ejecución.

3.5. Conclusiones parciales

En este capítulo se presentaron los estándares de codificación establecidos para Quarxo, permitiendo la obtención del código del subsistema de manera organizada, siendo este práctico y entendible para cualquier programador. Con la descripción de las principales clases y de sus correspondientes métodos se logró la comprensión del flujo entre ellas para la implementación de los módulos de Captación, Administración e Integración, desarrollados con tecnologías libres. Los módulos fueron probados mediante pruebas de caja blanca y caja negra, quedando validados mediante pruebas de aceptación y demostrando que cumplen con los requerimientos necesarios para satisfacer las necesidades del cliente. Quedó demostrado que se reduce el tiempo necesario para el envío de todos los mensajes SWIFT necesarios para las gerencias del BNC.

CONCLUSIONES

Con la culminación del presente trabajo se arriban a las siguientes conclusiones:

- Se dio solución al problema planteado al inicio de la investigación, eliminado de esta manera las deficiencias que presentaba el subsistema Mensajería.
- A través del análisis, diseño, implementación y prueba de los requisitos detectados para dar solución a las deficiencias del sistema en explotación, se logró desarrollar el subsistema Mensajería SWIFT versión 2 soportando el envío de todos los mensajes SWIFT utilizados en el BNC, dando solución así al objetivo general propuesto.
- Con el desarrollo de los módulos Captación, Administración e Integración se contribuyó a la eficiencia del sistema Quarxo, mejorando considerablemente el trabajo con los mensajes SWIFT en el banco y minimizando el tiempo utilizado para esta gestión.

RECOMENDACIONES

- Continuar los estudios del tema con el objetivo de encontrar nuevas funcionalidades para futuras mejoras de la aplicación.
- Adaptar el subsistema obtenido para que pueda ser utilizado posteriormente por otras entidades bancarias que incluyan el trabajo con mensajes SWIFT entre sus funciones.

BIBLIOGRAFÍA

Agile software development & services. 2010. Fixprotocol. *Fixprotocol.* [Online] 2010. [Cited: 12 15, 2012.] http://www.fixprotocol.org/.

Areba, Jesús Barranco de. 2001. *Metodología Del Análisis Estructurado de Sistemas.* 2001.

CEIGE. 2012. *MODELO DE DESARROLLO DE SOFTWARE.* 2012.

Channon, Dereck F. 2001. *Estrategia Global Bancaria.* 2001.

Chaviano, Adolfo Miguel Iglesias. 2011. *Análisis de las tecnologías compatibles con la norma ISO 15022 de SWIFT.* La Habana : s.n., 2011.

CiberAula. 2010. Ciberaula. *Ciberaula.* [Online] 2010. [Cited: 1 23, 2013.] http://java.ciberaula.com/.

Dülmen, Richard Van. 2002. *Los Inicios de la Europa moderna, 1550-1648.* Buenos Aires : Argentina Editores S.A, 2002.

EDIFACTMX. ASI, Developer Center. *ASI, Developer Center.* [Online] [Cited: 1 15, 2013.] http://www.asiware.com/edifact.php.

Fonseca, Adrian Veranes. 2011. *Diseño e implementación de los módulos Transacciones generales y Plan de cuentas del sistema Quarxo.* La Habana : s.n., 2011.

Iglesias, Adolfo Miguel. 2009. *Documento de Arquitectura de Software. Modernización Sistema del Banco Nacional de Cuba.* 2009.

—. 2010. *Tesina para optar por el Diplomado de Formación de Investigadores: Subsistema de mensajería financiera para el Banco Nacional de Cuba.* Ciudad Habana : s.n., 2010.

2011. *Ingeniería de Software I Conferencia # 7 Flujo de Trabajo Prueba.* 2011.

JAMES C AUTOR VAN HORNE, JOHN M AUTOR WACHOWICS. *Fundamentos de administración financiera.*

Jetin, Bruno. 2005. *La tasa Tobin: un arma para detener la expoliación financiera.* s.l. : Fondo Editorial Question, 2005.

López, Ernest Teniente. 2003. *Diseño de sistemas software en UML.* 2003.

Marcuse, Robert. *Diccionario de términos Financieros & Bancarios.*

Mercado, Carlos Ortega. 2012. *¿Todos los caminos llevan a SWIFT? Conectividad y estandarización total.* Santiago de Chile : s.n., 2012.

Miguel Iglesias, Adolfo. 2012. *SOLUCIÓN INFORMÁTICA COMPATIBLE CON LA NORMA ISO 15022.* La Habana : s.n., 2012.

Nogueras Creuets., Jorge Carlos . 2009. *Propuesta de herramientas para la integración.* La Habana : s.n., 2009.

OPENTIA. 2007. *Estudio sobre Estándares Informáticos tipos y caracterizaciones.* España : s.n., 2007.

Pauli, Gunter. *El alto crecimiento: cómo lo consiguen las empresas.*

Piñero Añon, Alain Lázaro. 2011. *Diseño e implementación del módulo Captación de Mensajes en el sistema Quarxo.* La Habana : s.n., 2011.

Ponce, Jorge and Tubio, Magdalena. 2010. *Estabilidad financiera: conceptos básicos.* Montevideo, Uruguay : s.n., 2010. 004.

Pressman, Roger S. 2002. *Ingenirría del Software. Un enfoque práctico.* España : McGraw-Hi, 2002.

Reyes León , Yelani and Rivero Alonso, Yosbel . 2008. *DEFINICIÓN DE LOS REQUERIMIENTOS FUNCIONALES DEL MÓDULO NEGOCIACIONES Y PAGO DEL PROYECTO BANCO NACIONAL.* La Habana : s.n., 2008.

Rivero Alonso, Yosbel and Reyes León, Yelanis. 2008. *DEFINICIÓN DE LOS REQUERIMIENTOS FUNCIONALES DEL MÓDULO NEGOCIACIONES Y PAGO DEL PROYECTO BANCO NACIONAL.* 2008.

SIBANC. 2006. *Manual de Usuario del SISCOM.* [Documento Word] Habana : s.n., 2006.

Software, Unidad Docente de Ingeniería del. *Patrones del "Gang of Four".* España : s.n.

Sommerville, Ian. 2005. *Ingeniería del software. Séptima edición.* Madrid. : PEARSON EDUCACIÓN. S.A., 2005. ISBN: 84-7829-074-5.

SWIFT. 2013. SWIFT El proveedor global de servicios seguros de mensajería financiera. *SWIFT El proveedor global de servicios seguros de mensajería financiera.* [Online] 2013. [Cited: 1 21, 2013.] http://www.swift.com/.

SWIFTSibos. **SWIFTSibos. 2009.** 4, Hong Kong : s.n., 2009.

Tahchiev, P. y otros. 2010. *Junit in action. .* s.l. : Manning Publications Co., 2010.

Tejada, David Hernández. 2002. *Guia de Patrones de Diseño.* 2002.

trusted, Oxford Dictionaries. The world´s most. 2012. [Online] 2012. http://oxforddictionaries.com/definition/library.