Universidad de las Ciencias Informáticas

"Facultad 3"

Título: Diseño e implementación del módulo Captación de Mensajes en el sistema Quarxo

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

Autor:

Alain L. Piñero Añon

Tutor(es):

Ing. Yadira Calimano Meneses.

Ing. Adolfo Miguel Iglesias Chaviano.

"La Habana. ----, 2011"

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 firmamos la presente a los ____ días del mes de mayo del año 2010.

_____

Alain L. Piñero Añon

_____ _____

Ing. Yadira Calimano Meneses Ing. Adolfo Iglesias Chaviano

FRASE

"La imaginación es más importante que el conocimiento. El conocimiento es limitado, mientras que la imaginación no"

Albert Einstein

DEDICATORIA

A todos los que de una manera u otra se sientan con deseos de aprender y de ser buenos en lo que hacen. Que encuentren en este trabajo, parte de aquello que los haga buenos y mejores profesionales.

Dedico este trabajo que conforma el punto cumbre de una larga trayectoria como estudiante a todo aquel que de una forma u otra aporto un granito de arena, que brindó su apoyo, su confianza y su esfuerzo para que este sueño se hiciese realidad.

A la UCI y a la Revolución por darnos la oportunidad de formarnos como profesionales y por quedarnos aún en deuda.

AGRADECIMIENTOS

En lo familiar:
A mi mamá y a mi papá por todos estos años de esfuerzo, sacrificio y dedicación para que yo haya podido tener y ser todo lo que hasta hoy he logrado. A mi abuela por ser el ángel de mi guarda, por vivir para mí y brindarme su amor incondicional todos estos años que he compartido con ella. A mi hermana por marcar momentos felices en mi vida. A Pedro y a Marcia por estar ahí presentes cuando los necesitaba. A todos mis tíos que siempre me han demostrado mucho cariño. A toda mi familia, por ser parte importante en mi vida y sentirse siempre orgullosos de mí.

En lo amistoso:
A todos mis amigos, de la UCI y fuera de ella, a mis compañeros de grupo y otros que he adoptado a lo largo de estos 5 años. A los que siempre han tenido una mano para ayudar y un rato para compartir. Esos que sin dudas son los mejores que he tenido, los que más he querido y los que extrañaré muchísimo, a Leo un amigo para siempre, a Emilio y Annier por trabajar a mi lado día a día; a Manuel, Adrian, Ray y Ariel por compartir tantos ratos juntos; a Dayleni que siempre me ayudo cuando lo necesité, a todos los del proyecto (son muchos y cada uno especial).

En lo profesional:
A mis profesores de todas las etapas de vida escolar, todos muy buenos y que tanto me han enseñado. A esos que son la motivación a ser mejor cada día, y son el impulso para transmitir todo lo que de ellos he obtenido; a Adolfo, por demostrar ser un buen profesional y por confiar en mi voluntad. A todos por su consagración y cariño.

En la tesis:
A Yadira y a Adolfo, mis tutores, por toda la ayuda y soportarme todos estos años, a Yesenia que sin ser mi tutora siempre me brindó uno mano cuando me hizo falta. A ellos, por ser amigos y preocuparse por todos nosotros; por hacerme ingeniero.

En lo personal:
A Yuselis, por ser mi vida. Por ser mí amiga, mi compañera, mi confidente. La persona que me ha permitido quererle sin reservas y que me ha hecho completamente feliz. El impulso diario de todas mis acciones.

RESUMEN

Actualmente Cuba está llevando a cabo un programa de informatización nacional, proceso en el cual se aplican las nuevas tecnologías de la información y las comunicaciones en los diferentes sectores de la sociedad. El sistema bancario cubano es uno de los que se encuentra inmerso en dicho programa, por lo que a la Universidad de las Ciencias Informáticas (UCI) se le dio la tarea de desarrollar un sistema para la gestión bancaria que automatice los procesos del Banco Nacional de Cuba (BNC)

En Cuba se está desarrollando un programa de informatización nacional, permitiendo aplicar las nuevas tecnologías de la información y las comunicaciones en los procesos que se realizan en los diferentes sectores de la sociedad. El Sistema Bancario Cubano se encuentra inmerso en la modernización de sus sistemas con el objetivo de garantizar eficiencia y seguridad en sus procesos de negocios. Para ello se le solicitó a la Universidad de las Ciencias Informáticas (UCI) el desarrollar de una aplicación para la gestión bancaria que automatice los procesos del Banco Nacional de Cuba.

Este trabajo se basa en las actividades realizadas por el autor en los roles de diseñador e implementador como integrante del proyecto QUARXO. Se muestra el Diseño y la Implementación de la solución brindada para el módulo de Captación de Mensajes, a partir de los requisitos definidos A partir de la aplicación de una arquitectura por capas se realizó un diseño basado en patrones apoyándose en una metodología de desarrollo de software. Se utilizaron herramientas, tecnologías que permitieron la implementación de la solución que se propone. Para su implementación, se utilizaron herramientas y tecnologías de la plataforma JEE que exponen tendencias de la programación Web y que en su mayoría son libres y de código abierto.

Palabras claves

Programa de informatización nacional, sistema bancario cubano, sistema para la gestión bancaria, diseño basado en patrones, programación Web, código abierto.

Contenido

INTRODUCCIÓN

La sociedad está en continuo cambio y evolución, las nuevas tecnologías avanzan a pasos agigantados sin que seamos totalmente conscientes del proceso. Hoy en día se apuesta por el uso masivo de las nuevas tecnologías y por potenciar su desarrollo industrial.

Los procesos bancarios, como muchos otros, descansan desde hace mucho tiempo en la tecnología. Las innovaciones tecnológicas constituyen unas de las principales causas de cambio y adaptación del sistema financiero, al nuevo entorno competitivo. Por lo que constituyen un elemento de cambio estructural y un factor estratégico clave, obligando a los bancos a mejorar constantemente su infraestructura, para satisfacer a una clientela interna y externa cada vez más exigente, que demanda rapidez, seguridad, eficiencia, y sobretodo que el servicio sea continuo.

Hoy en día con el desarrollo de las Tecnologías de Información ha aumentado considerablemente el volumen de las Operaciones Financieras Interbancarias (Operaciones realizadas entre las Instituciones Financieras) tales como: Órdenes de pago, Transferencias, etc. Por lo que se hace necesario el establecimiento de mecanismos que permitan el acceso y el intercambio fluido de la información financiera relevante entre las distintas entidades bancarias, sucursales o entidades financieras, eliminando las barreras entre estas y permitiendo la transparencia de los sistemas bancarios.

Cuba lleva a cabo un Programa Nacional de Informatización de la Sociedad, proceso mediante el cual se aplican las Nuevas Tecnologías de la información y las Comunicaciones a las diferentes esferas y sectores de la sociedad para lograr la optimización de recursos y mayor productividad en dichas esferas.

En nuestro país contamos con un Sistema Bancario que se encuentra en vías de desarrollo, siguiendo objetivos estratégicos, tales como la reducción de costes y agilidad de funcionamiento con el fin de mejorar la eficacia y la eficiencia de los servicios. En función de esto a La Universidad de las Ciencias Informáticas (UCI) se le da la tarea de desarrollar un sistema para la gestión bancaria denominado Quarxo que automatice los procesos del Banco Nacional de Cuba (BNC).

Como cualquier institución financiera pone en práctica el intercambio de información, ya que está sujeto al movimiento de capitales o circulación de divisa tanto a nivel nacional como a nivel internacional.

Para que el intercambio y monitoreo de dicha información sea posible, el BNC cuenta con diferentes procesos, dentro de los cuales se encuentra el de captación de mensajes, este se encarga de

confeccionar, modificar o cancelar los distintos tipos de mensajes, así como controlar el proceso de firmas a los que estos deben someterse.

En la actualidad el BNC tiene en funcionamiento una versión del Sistema Automatizado para la Banca Internacional de Comercio (SABIC) con el cual gestiona las actividades bancarias, y un sistema de comunicación de mensajería SWIFT (SISCOM) para la comunicación financiera con otros bancos.

Dentro de las principales limitaciones que presentan estos sistemas se encuentra el no estar integrados, lo que implica que las operaciones que traigan consigo creación o recepción de mensajes se tornen muy engorrosas ya que los usuarios necesitan entrar los mismos datos para el negocio y también para la generación de los mensajes.

Por otra parte el SISCOM no soporta los cambios anuales que se realizan a los estándares de mensajes utilizados, por lo que existe la necesidad de detener la aplicación e introducir dichos cambios en la base de datos, lo cual trae consigo una inversión de tiempo considerable, así como la utilización de personal altamente calificado para dicha tarea.

La información que se maneja en los mensajes es altamente sensible, ya que esta corresponde a cuentas, montos de dinero, etc. y no debe ser consultada sin autorización. Los ficheros donde se almacena dicha información se encuentran en las estaciones de trabajo del banco nacional por lo que puede ser accedida sin ningún tipo de restricción.

Atendiendo a los problemas anteriormente expuestos se hace necesario el desarrollo de la investigación en cuestión y se define como **problema a resolver**:

¿Cómo lograr la creación y procesamiento automático de los mensajes SWIFT en el sistema Quarxo?

Partiendo del problema anteriormente planteado se ha determinado como **objeto de estudio** los Sistemas Informáticos para la Comunicación Financiera en Entidades Bancarias.

Objetivo General:

Diseñar e implementar las funcionalidades del módulo de Captación de Mensajes en el sistema Quarxo.

Campo de acción:

Proceso de Captación de Mensajes en el Banco Nacional de Cuba.

Idea a defender:

Si se diseña e implementa el Módulo Captación de Mensajes del sistema Quarxo, se facilitará la comunicación financiera del Banco Nacional de Cuba con otros bancos.

Posible resultado:

Obtener el Diseño y la Implementación del módulo de Captación de Mensajes del sistema Quarxo.

Tareas de la investigación:

1. Estudio detallado de la tecnología y la arquitectura definidas en el proyecto.
2. Estudio de los patrones de diseño para la búsqueda de soluciones a problemas comunes en el desarrollo.
3. Caracterización de los requisitos especificados para el desarrollo del módulo propuesto.
4. Diseño de la Arquitectura del Módulo.
5. Implementación de la solución.
6. Realización de pruebas.
7. Documentación de la solución.

Estructura del Documento:

Capítulo 1: Se expone el estado del arte, donde se realiza la fundamentación teórica del tema. Se describe el objeto de estudio, se explica el funcionamiento del módulo, los procesos fundamentales y otros detalles considerables. Se mencionan y describen algunos sistemas existentes que incluyan el objeto de estudio. También se describen las tecnologías a considerar, el lenguaje de programación que se utiliza para la solución, los sistemas distribuidos, las metodologías de desarrollo y las herramientas de desarrollo.

Capítulo 2: En este capítulo se describe la arquitectura del módulo, se explica el modelo del diseño, así como los patrones utilizados para dar respuesta a la solución planteada. Se realiza la estructuración del módulo, evidenciando los diagramas de clases del diseño, los de secuencia y la descripción de las clases y sus atributos.

Capítulo 3: Se enfoca en la construcción de la solución explicando los aspectos principales de la implementación. Se realiza una descripción de las clases y funcionalidades del Módulo Captación de Mensajes, quedando establecida la validación de la solución propuesta, la implementación del producto y además se realizarán las pruebas unitarias al producto.

CAPÍTULO 1 - FUNDAMENTACIÓN TEÓRICA.

Introducción

El presente capítulo se centra en el análisis de los conceptos fundamentales que facilitarán la comprensión de la gestión de los mensajes Swift. Se analizarán los sistemas que automatizan los procesos de mensajería SWIFT a nivel mundial y los que se utilizan a nivel nacional.

El principal objetivo de este capítulo es justificar teóricamente la realización de la presente investigación, siendo en este capítulo también donde se describen las tecnologías y metodologías que se utilizan durante el desarrollo del producto.

Conceptos fundamentales

Banco

Los Bancos son entidades que canalizan los recursos financieros en la economía a través de la captación de depósitos y el otorgamiento de préstamos, para ello utilizan como principal instrumento las tasas de interés, las cuales son un incentivo tanto para ahorrar dinero como para pedir prestado (ANEC).

Se puede señalar que el banco moderno tiene que cumplir tres grandes funciones:

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

Bancos Cubanos

La política económica cubana ha evolucionado sustancialmente hacia un proceso de ajuste conforme a las nuevas circunstancias en que se desarrollan sus relaciones y en cuyo contexto, la banca juega un nuevo e importante papel.

El BNC desempeña un papel fundamental para el estado cubano llevando a cabo funciones de obtener y otorgar créditos en moneda nacional y libremente convertible. Centraliza las relaciones con las entidades extranjeras de seguro de crédito oficial a la exportación según decida el Banco Central de Cuba (BCC). Constituye una entidad de seguro de crédito oficial a la exportación con arreglo a la legislación vigente en materia de seguros. Mantiene el registro, control, servicio y atención a la deuda externa que el Estado cubano y el BNC tienen contraída con acreedores extranjeros hasta la fecha de entrada en vigor del Decreto Ley No.172, de 1997, del BCC (CEPEC).

Transmisión de información

La transmisión de información consiste en el movimiento de información codificada, de un punto a uno o más puntos de llegada, mediante señales eléctricas, ópticas, electroópticas o electromagnéticas. (DefiniciónABC, 2008)

Estándar de comunicación

Los estándares de comunicación no son más que un conjunto de normas y recomendaciones técnicas usadas consistentemente como reglas, guías o definiciones, que regulan la transmisión en los sistemas de comunicaciones. (Eveliux, 2011)

Comunicación financiera

La comunicación financiera actúa en todo momento como la fuerza que activa y dinamiza cualquier mercado, es una de las fuerzas confluyentes, necesaria pero no suficiente, para que los mercados financieros alcancen la flexibilidad y transparencia necesaria, para la optimización de la función de financiación. La Comunicación Financiera se refiere al conjunto de estrategias de comunicación, tanto a través de los medios de comunicación como fuera de ellos, que un número creciente de organizaciones lleva a cabo para dirigirse a un público específico, denominado “Sector Financiero”.

SWIFT como empresa

Sociedad de Telecomunicaciones Financieras Interbancarias Mundiales (SWIFT, por sus siglas en inglés) es una organización que tiene a cargo una red internacional de comunicaciones financieras entre bancos y otras entidades financieras, se creó en Bruselas en 1973apoyado por 239 bancos en 15 países y hoy en día cuenta con 7600 miembros ubicados en 200 países, con un volumen de 9 millones de mensajes diarios.

SWIFT es una cooperativa propiedad de sus miembros, a través de la cual el sector financiero lleva a cabo sus operaciones comerciales de forma rápida, segura y fiable (SWIFT, 2009).

Principales estándares de comunicación a nivel mundial.

En un principio, no existía un formato común generalmente aceptado para los datos contenidos en los informes financieros. Por esta razón, los datos debían reintroducirse a menudo en las aplicaciones informáticas de los usuarios para su interpretación y tratamiento, o debían ser copiados y pegados de una aplicación a otra, lo que producía una ausencia de seguridad en la transmisión de información. Por este motivo surgen estándares de comunicación financiera con el objetivo de crear un medio de transmisión de mensajes seguros sobre transacciones financieras internacionales. El Extensible Business Reporting Language (XBRL), el Financial Information Exchange (FIX), la UnitedNations/Electronic Data Interchange

For Administration, Commerce and Transport (UN/EDIFACT) y la Society For World Wide Interbank Financial Telecommunication (SWIFT) son algunos de los estándares desarrollados para dicho propósito.

XBLR

XBRL (Extensible Business Reporting Language) 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, permite también el intercambio automático de información entre diversas aplicaciones de software,proporciona una estructura de trabajo basada en XML, esto lo convierte en un lenguaje interoperable y bastante flexible.

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.

El creador y promotor del XBRL fue Charles Hoffman, quien desde el 1998 decidió dedicarse por completo al diseño de un estándar para intercambiar información. En la actualidad el XBRL es también una organización internacional, pues cuenta con más de 400 miembros alrededor del mundo, dentro de los cuales se encuentran Microsoft, Oracle, IBM, NEC y otras muy reconocidas compañías.(Chaviano, 2010)

FIX

FIX (Financial Information Exchange) surgió en el año 1992, construido para establecer comunicación entre dos empresas, las empresas Fidelity Investments y Salomon Brothers. Hoy en día, FIX ha evolucionado mucho, y al igual que las tecnologías, se ha ido mejorando, evolucionando y ampliado, hasta llegar a tener una gran aceptación a nivel mundial ya que más de 100 compañías e instituciones la utilizan.

FIX es una especificación técnica para las comunicaciones basadas en el uso de mensajes, es decir, constituye una serie de especificaciones de mensajería desarrolladas en colaboraciones con bancos, corredores de bolsa, intercambiadores, utilidades industriales y asociaciones, investigadores y proveedores de tecnologías de la información de todo el mundo. (Bolsa de Valores, 2007)

FIX fue galardonado en el 2005 como Best Utility Technology del año por la publicación especializada “Operations Management”, es la tecnología que han elegido las bolsas más importantes del mundo – CME, NYSE, NASDAQ, LSE, ISE, Deutsche Borse, OMX y NYMEX entre otras - para un proyecto conjunto cuyo objetivo es definir las mejores prácticas de la industria y la armonización operativa de sus distintos sistemas de negociación electrónica, el paso previo para la integración global. (Primary, 2005)

El protocolo FIX es un protocolo abierto y gratuito, pero no es un software, se utiliza para que los desarrolladores de software puedan crear productos comerciales o productos open-source.

EDIFACT

EDIFACT (Electronic Data Interchange For Administration, Commerce and Transport) es un estándar internacional reconocido para una amplia gama de sectores comerciales y no comerciales. Fue desarrollado por la Electronic Data Interchange (EDI) y patrocinado por las Naciones Unidas.

EDIFACT no define el medio por el cual se envía el mensaje, o los protocolos que se utilizan en la comunicación, el solamente definen los mensajes y sus contenidos. EDIFACT fue adoptado con la ISO 9735.

EDIFACT es un estándar que proporciona una gran flexibilidad, por esta razón se acomoda fácilmente a los requisitos individuales de cada aplicación.

Este estándar brinda un protocolo de intercambio (I-EDI) de mensajes estándar que permite a muchos países y múltiples industrias intercambiar información. Propone una regla sintáctica para estructurar los datos que pueden ir en el mensaje. Es un estándar que no se especializa en mensajes financiero; aunque no se descarta su uso en el sector bancario. Presenta una sintaxis compleja y muy amplia. Su documentación es abundante y de fácil acceso. (Chaviano, 2010)

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 (Extensible Mark-up Language) que es una sintaxis para transmitir información de una manera estructurada y es paralelo al HTML (Hyper Text Mark-up Language).

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, 2009).

La configuración del mensaje SWIFT es la siguiente:

- ✓ Cabecera (Header): en la cabecera figuran: el banco ordenante, el banco receptor, el tipo de mensaje, la prioridad, la hora de envío y la hora de recepción.
- ✓ Texto: figuran los campos específicos de cada mensaje.
- ✓ Tráiler o parte final del mensaje: figura la información que tiene que ver con la seguridad del mensaje.

Los mensajes se agrupan en categorías de acuerdo con su primer dígito:

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.

SWIFT debido a sus características y su amplia gama de mensajes fue seleccionado por el BCC como el estándar para la comunicación financiera en nuestro país.

Principales sistema que se usan a nivel mundial para la comunicación financiera.

B2B Data Transformation

Utilizado por Natixis, una de las entidades financieras de más rápido crecimiento en Europa con sedes en Paris, Nueva York y otros países del mundo, 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 (Informatics).

B2B Data Transformation es un software de transformación de datos concebido para datos no estructurados y específicos de cada sector, permite realizar transformaciones automatizadas en datos de todo tipo, incluyendo datos estructurados y mensajes SWIFT de elevada complejidad, dentro de un solo entorno. Ofrece un desarrollo rápido por medio de una biblioteca de transformaciones predefinidas que cubren más de 100 tipos de mensajes SWIFT.

Permite aprovechar e integrar otros datos no relacionales de toda la empresa, incluyendo correos electrónicos, hojas de cálculo, presentaciones y documentos PDF o Word, entre otras posibilidades.

Provee una mejor visibilidad e integridad de los datos gracias a una arquitectura de integración de datos basada en metadatos.

Incentage Middleware Suite (IMS)

IMStiene 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.

IMS fue recientemente laureado con la certificación SWIFTReady Gold Financial EAI 2006 confirmando la calidad del software y la fidelidad con los más altos estándares de SWIFT. (Clavelink, 2006)
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 Data Dictionary 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.

IMS abarca los más de 250 tipos de mensajes definidos en el paquete SWIFT FIN e incluyó un grupo de herramientas a fin de asegurar que los estándares de mensajería utilizados estén alineados con los raleases de software de SWIFT.

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, reducir el tiempo y los gastos relacionados con la implementación de las soluciones empresariales a través de SWIFTNet. (SWIFT, 2009)

Dicha aplicación proporciona un marco para la transferencia de información entre aplicaciones empresariales, incluye un paquete de integración para facilitar la conexión con aplicaciones de negocio y el intercambio de información por FTP[1], JMS[2], SOAP[3], entre otros.

[1] **FTP**: Protocolo de Transferencia de Archivos.

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 a partir de reglas definidas por SWIFT para la conversión automática y la compatibilidad con varias versiones de los estándares de mensajes. (Chaviano, 2010)

En sentido general los sistemas antes expuestos, constituyen buenas soluciones informáticas para la comunicación financiera a nivel mundial, debido a la gran gama de funcionalidades que brindan y la calidad y la seguridad con que manejan la información, pero es necesario resaltar que dichos sistemas no pueden ser usados por el estado cubano debido a su condición de privativos, por lo que se necesita pagar una licencia para poder utilizarlos. No obstante el análisis de estos sistemas nos permite obtener una buena visión en cuanto a tecnologías y plataformas para nuestro futuro software.

[2] **JMS**: El Java Message Service es la solución creada por Sun Microsystems para el uso de colas de mensajes, permite a los componentes de aplicaciones basados en la plataforma Java2 crear, enviar, recibir y leer mensajes. También hace posible la comunicación confiable de manera síncrona y asíncrona.

[3] **SOAP**: El *Simple Object Access Protocol* es un protocolo estándar que define cómo dos objetos en diferentes procesos pueden comunicarse por medio de intercambio de datos XML.

Sistema Nacional para la comunicación financiera.

SISCOM

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

El sistema está integrado por cuatro módulos:

SACIM: Sistema de Archivo, Captación e Impresión de Mensajes.

En este módulo se confeccionan, archivan e imprimen los distintos tipos de mensajes que se intercambian entre las instituciones financieras a través de la red SWIFT y también se controla el proceso de firma en los mensajes.

Middleware: Intermediario entre SISCOM y Alliance Access (AA).

El Middleware lo conforman dos aplicaciones, SwMPLink y SwMPClient, cuya función es enrutar y distribuir en ambos sentidos la mensajería entre la estación de comunicaciones del SISCOM ubicada en cada banco y el servidor AA, ubicado este en el BCC. El módulo Middleware se utiliza como intermediario entre los bancos y el servidor AA. Es el que garantiza el flujo de mensajería en ambos sentidos. (Chaviano, 2010)

SwAdmin: Funciones administrativas.

Tiene como objetivo principal el control de toda la mensajería tramitada a través del SISCOM garantiza la configuración y administración del sistema así como la creación de una salva del tráfico diario de mensajes y la recuperación de los mismos, así como la generación de reportes por diferentes criterios.

SwMPServer

Es el encargado de interactuar directamente con el AA a través del adaptador que soporta la transmisión de mensajes en lotes.

El SISCOM funciona correctamente permitiendo la transmisión de los mensajes SWIFT, pero luego de un análisis se determinó:

En el diseño de su base de datos se percibe que:

- ✓ No se pueden guardar todos los formatos de los mensajes definidos en el estándar
- ✓ Existe redundancia en la información que guarda

El SISCOM al estar desarrollado en FoxPro mantiene sus bases de datos en las estaciones de trabajo lo que representa una vulnerabilidad del sistema. No brinda puntos de integración (permiten la comunicación entre los sistemas sin importar las herramientas utilizadas en su desarrollo, ni la forma en que gestiona su información), por lo que costaría mucho más esfuerzo lograr una integración con otras aplicaciones desarrolladas en un lenguaje diferente al Fox Pro.

Con respecto a la flexibilidad que debe tener un sistema de comunicación financiera atendiendo a las modificaciones que ocurren anualmente en la norma ISO 15022 del estándar SWIFT, se puede plantear que el SISCOM no es flexible. Los cambios en las reglas semánticas no son soportados desde la aplicación; ya que no existen ficheros que guarden la expresión lógica que representa cada regla, ni la asociación de éstas con los mensajes. La forma de modificar las reglas semánticas es cambiando el código del programa. Con relación a las variaciones en las estructuras de los mensajes SWIFT, se puede plantear que tampoco son tolerados ya que la aplicación no cuenta con las funcionalidades necesarias. La forma utilizada para realizar los cambios es a través de la interacción directa con la base de datos. (Chaviano, 2010)

Atendiendo a las observaciones realizadas al sistema SISCOM se evidencia la necesidad de obtener una aplicación informática que ofrezca una gestión integral del proceso de mensajería.

Metodologías de Desarrollo

Las metodologías se desarrollan con el objetivo de dar solución a los problemas existentes en la producción de software, que cada vez son más complejos. Estas engloban procedimientos, técnicas, documentación y herramientas que se utilizan en la creación de un producto de software.

Rational Unified Process (RUP)

RUP es un proceso pensado en dos dimensiones. El mismo se repite a lo largo de una serie de ciclos que constituyen la vida de un sistema. Cada ciclo concluye con una versión del producto para los clientes, cada ciclo consta de cuatro fases y cada fase termina con un hito.

El ciclo de vida de RUP se caracteriza por:

- ✓ Dirigido por casos de uso: Los casos de uso (CU) reflejan lo que los usuarios futuros necesitan y desean, lo cual se capta cuando se modela el negocio y se representa a través de los requerimientos. A partir de aquí los casos de uso guían el proceso de desarrollo ya que los modelos que se obtienen, como resultado de los diferentes flujos de trabajo, representan la realización de los casos de uso (cómo se llevan a cabo).
- ✓ Centrado en la arquitectura: La arquitectura muestra la visión común del sistema completo en la que el equipo de proyecto y los usuarios deben estar de acuerdo, por lo que describe los elementos del modelo que son más importantes para su construcción, los cimientos del sistema que son necesarios como base para comprenderlo, desarrollarlo y producirlo económicamente. RUP se desarrolla mediante iteraciones, comenzando por los CU relevantes desde el punto de vista de la arquitectura. El modelo de arquitectura se representa a través de vistas en las que se incluyen los diagramas de UML.
- ✓ Iterativo e Incremental: Una iteración involucra actividades de todos los flujos de trabajo, aunque desarrolla fundamentalmente algunos más que otros. Por ejemplo, una iteración de elaboración centra su atención en el análisis y diseño, aunque refina los requerimientos y obtiene un producto con un determinado nivel, pero que irá creciendo incrementalmente en cada iteración.

RUP propone flujos de trabajo en los que se definen las secuencias de actividades, quienes las deben desarrollar y los artefactos a generar. Entre estos flujos se encuentra el de modelación del negocio donde se describen los procesos de negocio, identificando quienes participan y que actividades requieren automatización y el flujo de trabajo de requisitos, que define qué es lo que el sistema debe hacer para lo cual se identifican las funcionalidades requeridas.

Tecnologías y Herramientas utilizadas en el Desarrollo

Debido a la importancia de conocer las herramientas y tecnologías definidas por el grupo de arquitectura del proyecto Quarxo, ya que estas influyen considerablemente en el tiempo de desarrollo y la calidad del producto, en el siguiente fragmento del capítulo se realiza una breve caracterización de las mismas, mostrando los beneficios que aporta cada una de ellas en el trabajo de diseño e implementación del módulo en cuestión.

Herramienta CASE

El acrónimo CASE en inglés significa Computer Aided Software Engineering y se traduce como Ingeniería de Software Asistida por Computadoras. Las herramientas CASE son utilizadas con el fin de automatizar los aspectos clave de todo el proceso de desarrollo de un sistema.

Visual Paradigm

Constituye una herramienta para facilitar el trabajo durante la confección de un software así como garantizar la calidad del producto final,fue creada para el ciclo completo de desarrollo del software que lo automatiza y acelera, permitiendo la captura de requisitos, análisis, diseño e implementación, también proporciona características tales como generación del código, ingeniería inversa y generación de informes.

Debido a las funcionalidades brindadas por el Visual Paradigm al permitir crear diagramas usando las notaciones UML y por ser una herramienta multiplataforma que se integra fácilmente con varios entornos de desarrollo integrado (IDEs), se decidió usarla como herramienta durante todo el proceso de modelado.

Lenguaje de modelado

Un lenguaje de modelado se puede definir como el conjunto estandarizado de símbolos y de modos de disponerlos para modelar (parte de) un diseño de software orientado a objetos.

Unified Modeling Language (UML)

UML es un lenguaje, que permite modelar analizar y diseñar sistemas orientados a objetos. Es el lenguaje de modelado de sistemas de software más conocido y utilizado en la actualidad (Jacobson, I, Booch, G y Rumbaugh, J. 2000.).

Ofrece un estándar para describir un panorama del sistema modelo, incluyendo aspectos concretos como expresiones de lenguajes de programación, esquemas de bases de datos y componentes de software reutilizables y aspectos conceptuales como los procesos de negocios y funciones del sistema.

UML es un lenguaje para visualizar, especificar, construir y documentar los artefactos de un sistema que involucra una gran cantidad de software. El UML está compuesto por diversos elementos gráficos que se combinan para conformar diagramas. La finalidad de los diagramas es presentar diversas perspectivas de un sistema, a las cuales se les conoce como modelo.

Debido a que el UML es un lenguaje, cuenta con reglas para combinar tales elementos. Es importante recalcar que UML no es una guía para realizar el análisis y diseño orientado a objetos, es decir, no es un proceso. UML es un lenguaje que permite la modelación de sistemas con tecnología orientada a objetos.

Patrones de Diseño

Los patrones de diseño de software son una solución probada para un problema general de diseño, en un contexto determinado. Encierran la experiencia de los programadores e ingenieros que han ido adquiriendo en las soluciones de problemas comunes.

Patrones de Asignación de Responsabilidades. (GRAPS)

Patrones Generales de Software para Asignación de Responsabilidades (General Responsibility Assignment Software Patterns). Describen los principios fundamentales de diseño de objetos para la asignación de responsabilidades. Entre las responsabilidades más importantes están:
Conocer:

- ✓ Conocer los datos privados encapsulados.
- ✓ Conocer los objetos relacionados.
- ✓ Conocer las cosas que puede derivar o calcular.

Hacer:

- ✓ Hacer algo él mismo, como crear un objeto o hacer un cálculo.
- ✓ Iniciar una acción en otros objetos.
- ✓ Controlar y coordinar actividades en otros objetos.

Patrón Bajo Acoplamiento

Su principal característica es mantener las clases más independientes entre sí y con la menor cantidad de relaciones; la cual posibilita que en caso de producirse una modificación en alguna de ellas, se tenga la mínima repercusión posible en el resto de las clases, potenciando la reutilización y disminuyendo la dependencia entre las clases, asignándoles una responsabilidad para mantener un bajo acoplamiento. (Patrones)
Ventajas:

- ✓ No se afectan por cambios de otros componentes.
- ✓ Fácil de entender por separado.

- ✓ Fácil de reutilizar.

Patrón Alta Cohesión

La principal característica de este patrón es asignar responsabilidades de modo que la cohesión siga siendo alta. La información que almacena una clase debe de ser coherente y debe estar en la medida de lo posible relacionada con la clase. (Patrones)

Ventajas:

- ✓ Mejoran la claridad y la facilidad del diseño.
- ✓ Simplifican el mantenimiento y las mejoras en funcionalidad.
- ✓ A menudo, se genera un bajo acoplamiento.
- ✓ Soporta una mayor capacidad de reutilización.

Patrón Creador

Permite 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. (Patrones)

Ventajas:

- ✓ Brinda soporte a un bajo acoplamiento, lo cual supone poca dependencia respecto al mantenimiento y mejores oportunidades de reutilización.

Patrón Experto

Este patrón permite asignar una responsabilidad al experto en información: la clase que cuenta con la información necesaria para cumplir la responsabilidad. La responsabilidad de la creación de un objeto o la implementación de un método, debe recaer sobre la clase que conoce toda la información necesaria para crearlo. De este modo se obtendrá un diseño con mayor cohesión y así la información se mantendrá encapsulada (disminución del acoplamiento). (Patrones)

Ventajas:

- ✓ Se conserva el encapsulamiento.

- ✓ Soporta un bajo acoplamiento, lo que favorece al desarrollo de sistemas más robustos y de fácil mantenimiento.
- ✓ El comportamiento se distribuye entre las clases que cuentan con la información requerida. Brinda soporte a una alta cohesión.

Patrón Controlador

Posibilita la asignación de responsabilidades para controlar el flujo de eventos del sistema, a clases específicas. Sirve como intermediario entre una determinada interfaz y el algoritmo que la implementa, de tal forma que es la que recibe los datos del usuario enviándolo a las distintas clases según el método llamado.

Ventajas:

- ✓ Mayor potencial de los componentes reutilizables.
- ✓ Analiza sobre el estado del caso de uso.

Patrones Estructurales

Los patrones estructurales están relacionados con cómo las clases y los objetos se combinan para dar lugar a estructuras más complejas y describen las formas de componer los objetos para conseguir una nueva funcionalidad.

Facade (Fachada)

Proporciona una interfaz unificada a un conjunto de interfaces de un sistema. Facade define una interfaz de alto nivel, posibilitando que el sistema sea más fácil de usar. (Patrones)

Ventajas:

- ✓ Facilita el uso del subsistema.
- ✓ Se promueve un acoplamiento débil entre el subsistema y sus clientes, al ser eliminadas o reducidas las dependencias.
- ✓ No existen obstáculos para que las aplicaciones usen las clases del subsistema que necesiten. De esta forma, se puede elegir entre facilidad de uso y generalidad.

Patrones de Comportamiento

Los patrones de comportamiento estudian las relaciones de llamadas entre los diferentes objetos, proporcionan estrategias comprobadas para modelar la manera en que los objetos colaboran entre sí en un sistema.

Patrón de Acceso a Datos (DAO)

Centraliza todo el acceso a datos en una capa independiente, aislándolo del resto de la aplicación. Desacoplando la lógica de negocio de la lógica de acceso a datos, de manera que se pueda cambiar la fuente de datos fácilmente. Sus principales beneficios son que reduce la complejidad de los objetos de negocio al abstraerlos de la implementación real de la comunicación con la fuente de datos. Cada DAO tiene que implementar los métodos declarados en la interfaz DAO correspondiente. (Barroso y Bello, 2010)

Mediator (Mediador)

Define un objeto que coordine la comunicación entre objetos de distintas clases, pero que funcionan como un conjunto. Evita las referencias explícitas entre los objetos y permite, por tanto, que su interacción se modifique de forma independiente. (Patrones)
Ventajas:

- ✓ Al reducir el acoplamiento entre los objetos que interactúan es más fácil variar o reutilizar dichos objetos.
- ✓ Al hacer de la mediación un concepto independiente, encapsulado en un objeto a parte, se favorece la concentración de la atención en la interacción entre objetos, al margen del comportamiento individual de cada uno.

Strategy (Estrategia)

Dispone de varios métodos para resolver un problema y elige cuál utilizar en tiempo de ejecución. Este permite a los algoritmos variar con independencia de los clientes que los usan. (Patrones)

Ventajas:

- ✓ Posibilita ofrecer diferentes implementaciones del mismo comportamiento, en función de restricciones como el espacio en memoria o el tiempo de respuesta.

Se concluye con que los patrones de diseño:

- ✓ Proponen una forma de reutilizar la experiencia de los desarrolladores, clasificando y describiendo formas de solucionar problemas que ocurren en el desarrollo de software.
- ✓ Están basados en la recopilación del conocimiento de los expertos en desarrollo de software.

Patrón de Presentación

Composite View (Vistas Compuestas)

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.* (Barroso y Bello, 2010)

MVC (Modelo Vista Controlador)

Este patrón propone dividir la aplicación en tres capas el Modelo, la Vista y el Controlador, el modelo es la representación del dominio o datos del sistema, la vista se encarga de presentar la interfaz al usuario, en sistemas web, esto es típicamente HTML, aunque pueden existir otro tipo de formatos. En la vista sólo se deben de hacer operaciones simples y el controlador es el encargado de escuchar los cambios en la vista y enviarlos al modelo, el cual le regresa los datos a la vista como un ciclo. Cuando se aplica este patrón los accesos a la base de datos se hacen en el modelo, la vista y el controlador no deben de saber si se usa o no una base de datos. El controlador es el que decide que vista se debe de imprimir y que información es la que se envía. (Barroso y Bello, 2010)

Ambiente de Desarrollo

Plataforma JEE

En el desarrollo de software cuando se desea seleccionar un lenguaje de programación se debe tener en cuenta elementos como son: el entorno de ejecución, el rendimiento, la escalabilidad, la portabilidad y la seguridad, una vez analizados estos puntos de forma detallada se puede seleccionar el lenguaje que más se ajuste según lo que se desea.

La plataforma Java Enterprise Edition (JEE) creada por Sun Microsystems, define un conjunto de estándares y especificaciones para el desarrollo de aplicaciones empresariales basado en la tecnología Java. JEE está capacitada para ejecutar aplicaciones creadas usando el Lenguaje de programación Java u otros lenguajes que compilen a bytecode[4]. La plataforma como tal es una máquina virtual encargada de la ejecución de aplicaciones, y un conjunto de librerías estándar que ofrecen funcionalidad común.

Lenguaje Java

Java es un lenguaje de programación orientado a objetos, robusto y fácil de aprender, permite la codificación de la propuesta de solución de una manera rápida y flexible, al integrarse con los 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.

Es un lenguaje multiplataforma de código abierto, distribuido, que proporciona 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, Java fue diseñado para crear software altamente fiable (Barroso y Bello, 2010).

[4]**Bytecode**: código intermedio entre el código fuente y el código máquina.

Contenedor Web (Tomcat)

El Tomcat es un contenedor de servlets que implementa las especificaciones de Java Server Page (JSP). Es desarrollado en un entorno abierto. Tomcat puede funcionar como servidor web por sí mismo. Es usado como servidor web autónomo en entornos con alto nivel de tráfico y alta disponibilidad.

Control de Versiones (SubVersion).

Subversion es un software de sistema de control de versiones diseñado específicamente para reemplazar al popular CVS, el cual posee varias deficiencias. Es un software libre bajo la licencia Apache y se le conoce también como SVN por ser ese el nombre de la herramienta de línea de comandos. Una característica importante de Subversion es que, a diferencia de CVS, los archivos versionados no tienen cada uno un número de revisión independiente; en cambio, todo el repositorio tiene un único número de versión que identifica un estado común de todos los archivos del repositorio en cierto punto del tiempo. (Barroso y Bello, 2010)

Existen dos razones fundamentales para el uso de esta herramienta:

- ✓ Gestiona las modificaciones durante el desarrollo.
- ✓ Permite que varias personas trabajen sobre los mismos ficheros.

Base de Datos

SQL Server 2005 provee herramientas sólidas, reduciendo la complejidad de la creación, despliegue, administración y uso de aplicaciones analíticas y de datos empresariales en plataformas que van desde los dispositivos móviles hasta los sistemas de datos empresariales. 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. A través de un conjunto global de características, la interoperabilidad con sistemas existentes y la automatización de tareas rutinarias, SQL Server 2005 ofrece una solución completa de datos para empresas de todos los tamaños. (IGP SQLServer, 2007)

Eclipse IDE

Es un IDE de programación gratuito asociado comúnmente al lenguaje de programación Java. Constituye un armazón sobre el que se pueden montar herramientas de desarrollo para cualquier lenguaje, mediante la instalación de los plugins adecuados. La arquitectura de plugins[5] de Eclipse permite, además de integrar diversos lenguajes sobre un mismo IDE, introducir otras aplicaciones accesorias que pueden resultar útiles durante el proceso de desarrollo como: herramientas UML, editores visuales de interfaces y ayuda en línea para librerías.

Framework

Los frameworks orientados al objeto son la piedra angular de la moderna ingeniería del software. El desarrollo de frameworks está ganando rápidamente la aceptación debido a su capacidad para promover la reutilización del código del diseño y el código fuente (source code). Los frameworks son los Generadores de Aplicación que se relacionan directamente con un dominio específico, es decir, con una familia de problemas relacionados. Un framework no tiene funcionalidades de una aplicación específica, sino que las aplicaciones se construyen sobre ellos.

Spring Framework

El Spring Framework (también conocido como Spring) es un framework de código abierto de desarrollo de aplicaciones para la plataforma Java. A pesar de que Spring Framework no obliga a usar un modelo de programación en particular, se ha popularizado en la comunidad de programadores en Java al considerársele una alternativa y sustituta del modelo de Enterprise JavaBean. Por su diseño el framework ofrece mucha libertad a los desarrolladores en Java y soluciones muy bien documentadas y fáciles de usar, prácticas comunes en la industria.

[5]Un **plugin** es un módulo de hardware o software que añade una característica o un servicio específico a un sistema más grande.

Spring Framework 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 de forma que podría ser desvinculada de él sin demasiada dificultad.

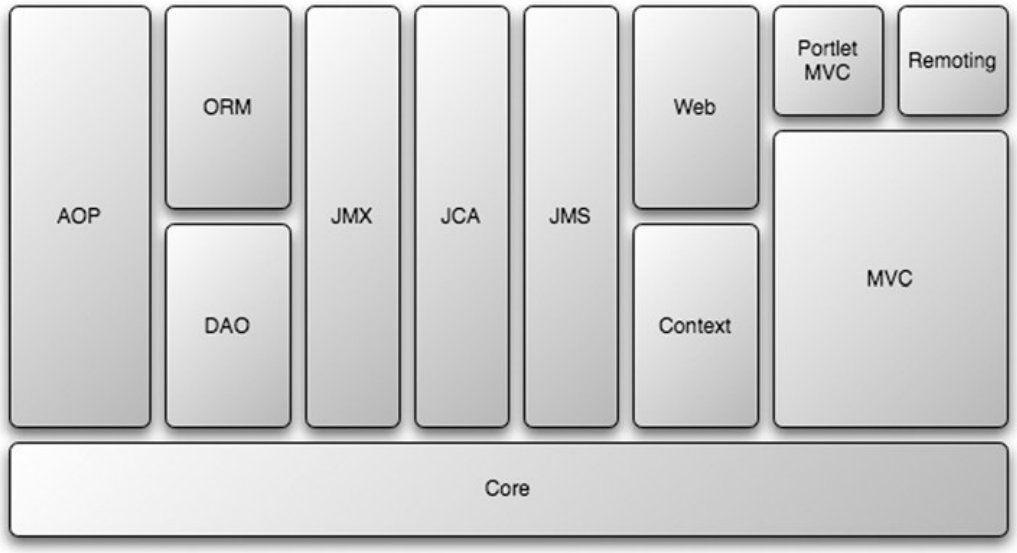


Ilustración 1: Módulos de Spring Framework.

Spring WebFlow

Spring WebFlow es un framework que permite operar la navegación de la aplicación Web, proporciona un motor capaz de capturar los flujos de páginas de una aplicación e integrarlo con otros frameworks de presentación. Los flujos Web en este framework son diseñados para ser auto-controlados, dando la posibilidad de definir reglas de navegación múltiples y complejas. Permite la creación de flujos reutilizables en toda la aplicación (Barroso y Bello, 2010).

Hibernate

El framework Hibernate tiene como objetivo facilitar la persistencia de objetos Java en bases de datos relacionales y la consulta de estas bases de datos para obtener objetos.

Hibernate integra la estructura relacional y la objetual permitiendo que el programador trabaje desde Java de forma fácil y correcta, usando el modelo objetual, pero además se integra en cualquier tipo de aplicación por encima del contenedor de datos. Ofrece un lenguaje de consulta de datos llamado HQL (Hibernate Query Language). Entre los beneficios que aporta está la independencia de la base de datos, el bajo acoplamiento entre negocio y persistencia, además proporciona un desarrollo rápido, cubriendo de manera sencilla y rápida la persistencia de una aplicación.

DojoToolkit

Esta herramienta es un framework que contiene APIs[6] y controles para ayudar al desarrollo de aplicaciones web. Incluye un sistema de empaquetado, los efectos visuales de la interfaz de usuario, abstracción de eventos, almacenamiento de APIs en el cliente, e interacción de APIs con AJAX.
Dojo proporciona variadas opciones en una sola biblioteca haciendo un mejor trabajo que sustenta los nuevos y viejos Browsers, resolviendo los problemas de compatibilidad entre los navegadores. Tiene múltiples puntos de entrada, es independiente del intérprete y unifica estándares de codificación. (Barroso y Bello, 2010)

[6] **API**: API es la abreviatura de **A**plication **P**rogramming **I**nterface. Un API no es más que una serie de servicios o funciones que se le ofrece al programador.

Concluciones Parciales

Al resultar poco factible costear uno de los sistemas ya existentes en el mundo y además no existir ninguno que se encargue de la comunicación financiera elaborado bajo la filosofía de software libre, se hace necesario el desarrollo de uno. Por lo que se realiza el diseño e implementación del modulo en cuestión, a partir de la definición de requisitos existentes.

La metodología seleccionada permitirá el seguimiento de un proceso de desarrollo y la obtención de un producto con las características esperadas.

El ambiente de desarrollo integrado por la plataforma J2EE, el IDE Eclipce, el gestor de base de datos SQL server, los frameworks Dojo, Spring e Hibernate y la herramienta de modelado Visual Paradigm, así como el uso de un estilo arquitectónico por capas, permitirá el desarrollo claro y fluido de un sistema construido sobre bases sólidas y un entorno de desarrollo bien definido.

A partir de este análisis se dará paso a los temas específicos del Módulo de Captación de Mensajes.

CAPÍTULO 2–ARQUITECTURA Y DISEÑO DEL MODULO CAPTACIÓN DE MENSAJES.

Introducción

El objetivo que se persigue con el desarrollo de este capítulo es llevar a cabo la descripción de los elementos fundamentales a tener en cuenta en el desarrollo del producto, así como la descripción del diseño de la solución.

EL diseño permite ser una guía para que los desarrolladores puedan leer y entender el producto que se desea construir. En él se definen las clases y asociaciones que se utilizarán en la implementación, así como las interfaces y los algoritmos de los métodos utilizados para implementar las operaciones.

Los objetivos del diseño son:

- ✓ Desarrollar un diseño para el sistema.
- ✓ Adaptar el diseño a las particularidades del sistema para que sea consistente con el entorno de implementación.

Además en este capítulo se explicará la arquitectura del sistema Quarxo, la cual responde a las necesidades planteadas en el proyecto.

Arquitectura del sistema Quarxo

La arquitectura que se propuso en el proyecto y sobre la que se realiza el desarrollo del software está compuesta por una arquitectura por capas, con tres capas principales:
Capa de Presentación.
Capa de Negocio.
Capa de Acceso a Datos.

Y por una capa transversal a las otras capas con los objetos del dominio. Esta es para aquellos objetos del dominio que están presente en el resto de las capas, como lo son el módulo de seguridad, el módulo de auditoría, el manejo de transacciones.

Dicha arquitectura fue 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: Conjunto de módulos relacionados con los procesos que ejecutan.
Módulo: Conjunto de Casos de Uso relacionados con uno o más procesos bancarios.
Componentes: Conjunto de funcionalidades comunes que son reutilizados por el resto de los módulos del sistema.

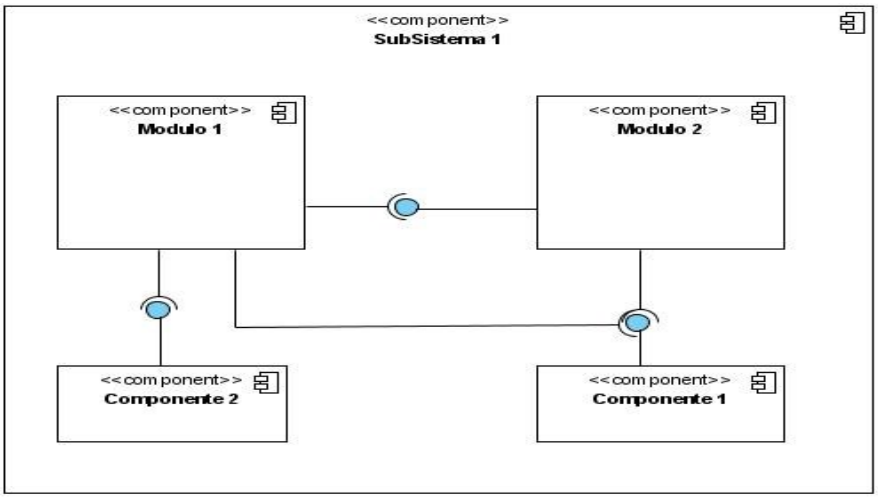


Ilustración 2: Arquitectura de un subsistema del Sistema QUARXO.

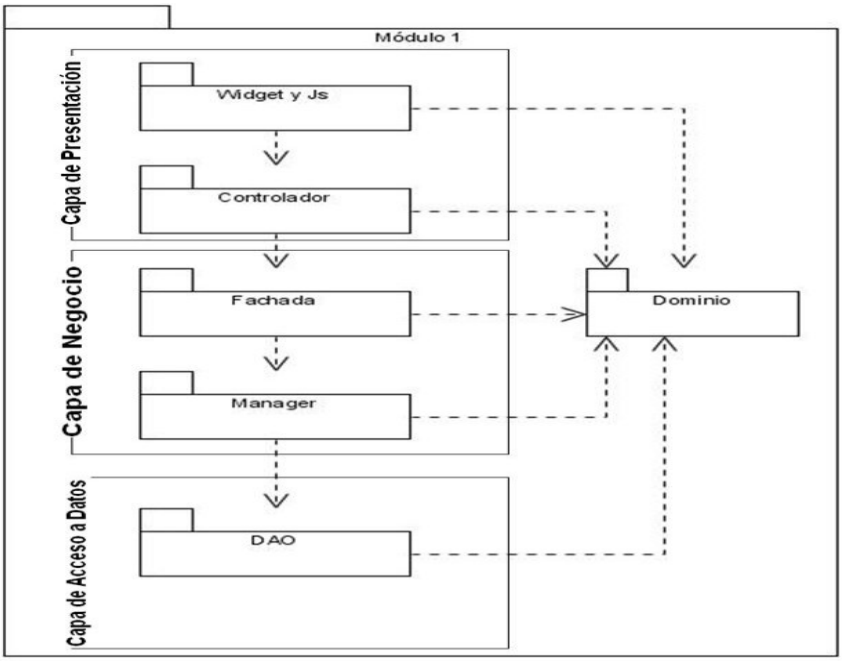


Ilustración 3: Arquitectura de un módulo del Sistema QUARXO.

Capa de Presentación:

En esta capa se desarrollará la lógica de presentación, se utilizará en el lado del servidor el framework Spring MVC para recibir, controlar y enviar respuestas a las peticiones realizadas desde el cliente. En el lado del cliente se utilizará la librería DojoToolkit para generar las interfaces que interactuarán con el usuario y el framework Spring WebFlow para representar y controlar los flujos complejos y reutilizables de la aplicación. Esta capa estará relacionada con la Capa de Negocios y la Capa de Dominio.

Capa de Negocio:

La Capa de negocio estará divida en dos subcapas, Facade y Manager. La capa Facade será el punto de intercambio entre la Capa de presentación y la Capa de negocio en esta se agrupan las funcionalidades para que pueda ser invocada desde la Capa de presentación. Por otro lado, la subcapa Manager tendrá la jerarquía de clases suficiente para implementar el negocio de la aplicación, esta subcapa utilizará la Capa

de acceso a datos para obtener los datos persistidos y la Capa de dominio para generar las entidades del negocio.

Capa de Acceso a Datos:

Esta capa se encargará de la conexión con el gestor de base de datos, en ella se realizarán todas las operaciones que permitan el acceso a los datos de la aplicación. Para el desarrollo de esta capa se utilizará el patrón DAO (Data Acces Object en inglés), en español Objetos de Acceso a Datos y la interacción con la capa de negocio será a través de su interfaces.

Diseño del Módulo

Tomando como artefactos de entrada los requerimientos definidos por los analistas del QUARXO, las funcionalidades a diseñar son:

- ✓ Actualizar Mensaje Conformado SWIFT.
- ✓ Consultar Mensaje Conformado SWIFT.
- ✓ Conformar Mensaje a partir de la aplicación de negocio.
- ✓ Buscar Mensaje Conformado SWIFT.
- ✓ Cancelar un Mensaje Conformado SWIFT.
- ✓ Cambiar Estado de un Mensaje.
- ✓ Eliminar Encabezado de un Mensaje.
- ✓ Realizar maqueta.
- ✓ Actualizar maqueta.
- ✓ Eliminar maqueta.

Modelo de Datos

Los modelos de datos aportan la base conceptual para diseñar aplicaciones que hacen un uso intensivo de datos. Un modelo de datos es usado para describir la representación lógica y física de la información persistente[7] manejada por el sistema. Con el objetivo de garantizar el almacenamiento de los datos persistentes utilizados por la aplicación en el módulo Captación de Mensajes se diseñó un modelo de datos que cumplen con las restricciones identificadas.

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* constituyen las encargadas de responder por el almacenamiento del formato de los mensajes SWIFT.

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.

Las tablas *o_firma_mensaje_usuario, o_usuario_firma*y*o_tipo_firma* identifican los mensajes que han sido firmados, firma que responde a una previa revisión del mensaje.

La tabla *o_numero_mensaje_operacion*se encarga de relacionar cada mensaje con la operación a la que pertenece.

Por último, las tablas *o_maqueta_SWIFT*, *o_swift_mensaje_maqueta* y *o_maqueta_negocio_swift* almacenan las maquetas de cada mensaje, que no son más que mensajes predefinidos que se utilizan como plantillas a la hora de confeccionar nuevos mensajes.

El modelo de datos se presenta a continuación:

[7]La persistencia es la capacidad de un objeto de mantener su valor en el espacio y en el tiempo.

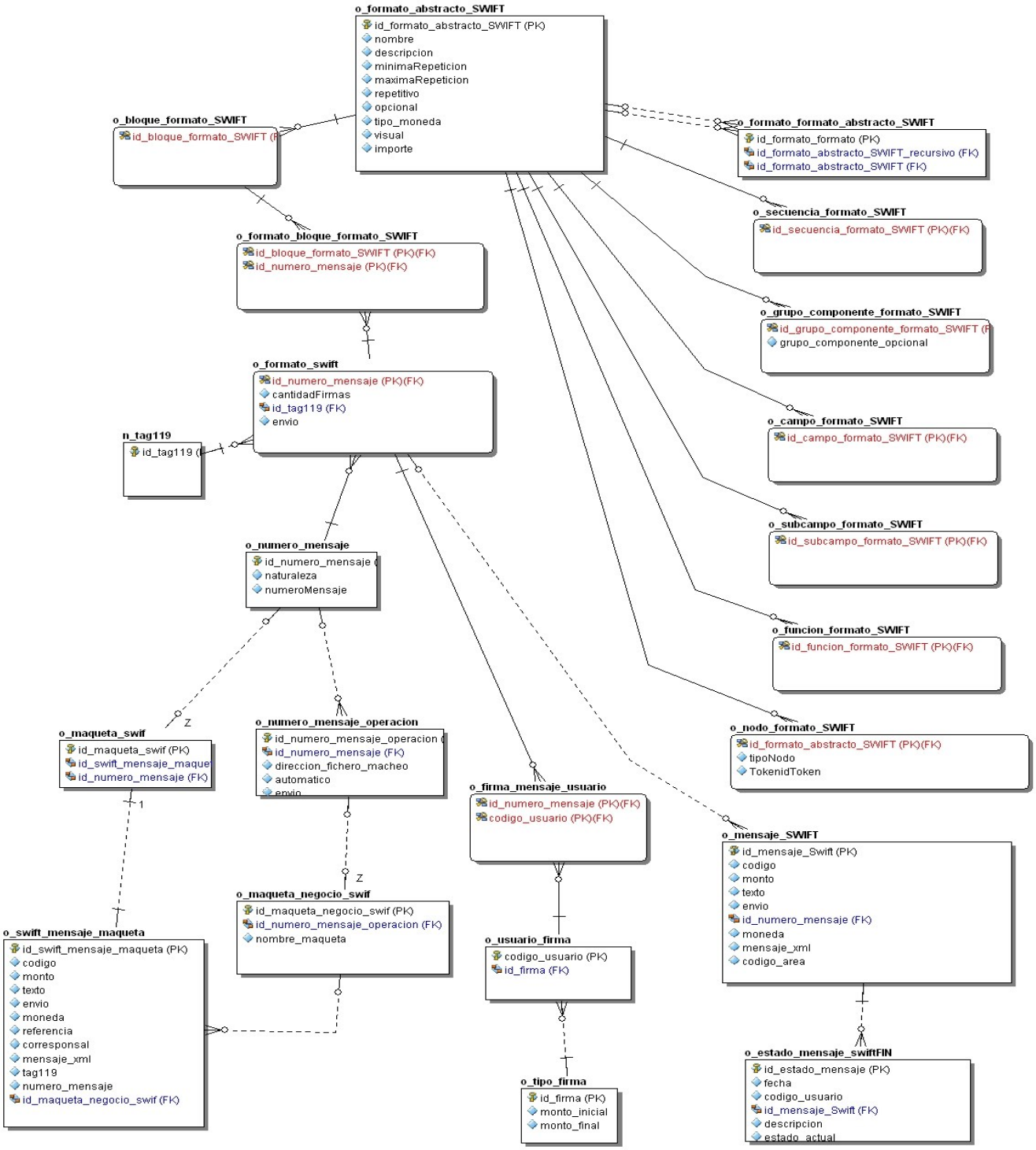


Ilustración 4: Modelo de Datos del Módulo Captación de Mensajes.

Modelo de Paquetes

Los paquetes reflejan la arquitectura de alto nivel de un sistema: su descomposición en subsistemas y sus dependencias. Una dependencia entre paquetes resume las dependencias entre los contenidos del paquete. Los mismos están organizados de manera funcional, siguiendo un cierto principio racional, tal como: funcionalidad común, implementación estrechamente relacionada, y un punto de vista común. Los paquetes pueden contener otros paquetes. Hay un paquete raíz, que contiene indirectamente el modelo completo de un sistema.(Rumbaugh)

A continuación se muestra el diagrama de paquetes para el Módulo Captación de mensajes:

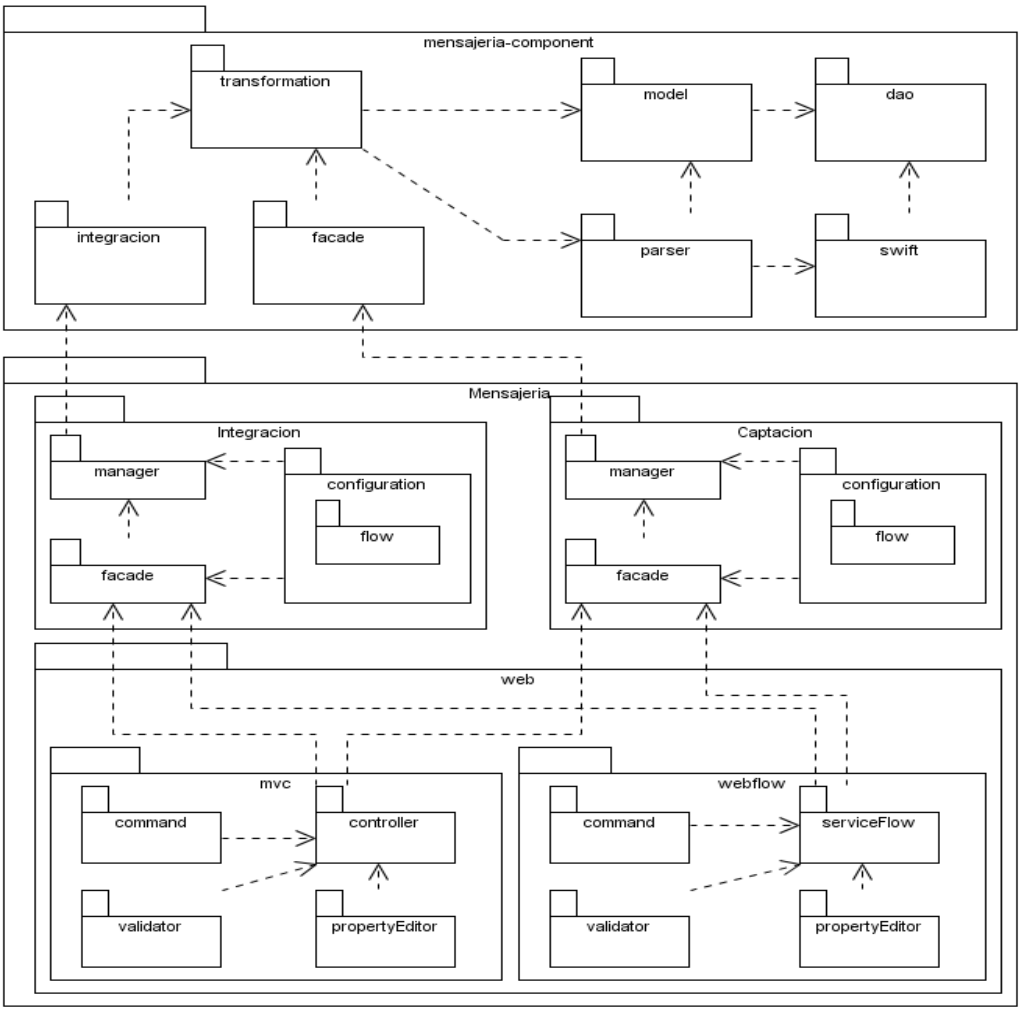


Ilustración 5: Modelo de Paquetes del Módulo Captación de Mensajes.

En el caso del Módulo Captación se decidió elaborar un paquete Integración que fuera el encargado de las funcionalidades que permiten la conformación de los mensajes a partir de la aplicación de negocio y un paquete Captación para solucionar las demás funcionalidades que debe permitir el Módulo.
El paquete Mensajería-Component es el encargado de resolver todo el negocio del Subsistema Mensajería (transformación y parseo de los mensajes, validación semántica y sintáctica de los mensajes, contiene las clases que constituyen el modelo de datos y las clases DAO, presenta un número de clases encargadas de proporcionar puntos de entrada a las funcionalidades que brinda dicho componente las cuales se encuentra en los paquetes Integración y Facade.

Organizar por criterios, en paquetes, las clases y ficheros de un módulo ayuda a la fácil comprensión de la aplicación, en el caso de Captación las mismas se agruparon 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 SpringMVC.

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.

Diseño de la Capa de Presentación

En la capa de presentación del Módulo Captación se hizo uso de los frameworks Spring MVC y Spring Web Flow en dependencia de la complejidad de las funcionalidades.
Donde Spring Web Flow se utilizó para darle solución a la siguiente funcionalidad que constituye un proceso con un flujo complejo:

- ✓ Actualizar Mensaje Conformado SWIFT.
- ✓ Conformar Mensaje a partir de la aplicación de negocio.
- ✓ Eliminar Encabezado de un Mensaje.

Por otra parte Spring MVC fue utilizado para darle solución al resto de los casos de usos:

- ✓ Consultar Mensaje Conformado SWIFT.
- ✓ Cambiar Estado de un Mensaje.
- ✓ Cancelar un Mensaje Conformado SWIFT.
- ✓ Realizar maqueta.
- ✓ Actualizar maqueta.

- ✓ Eliminar maqueta.

Diagrama 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 el diseño.

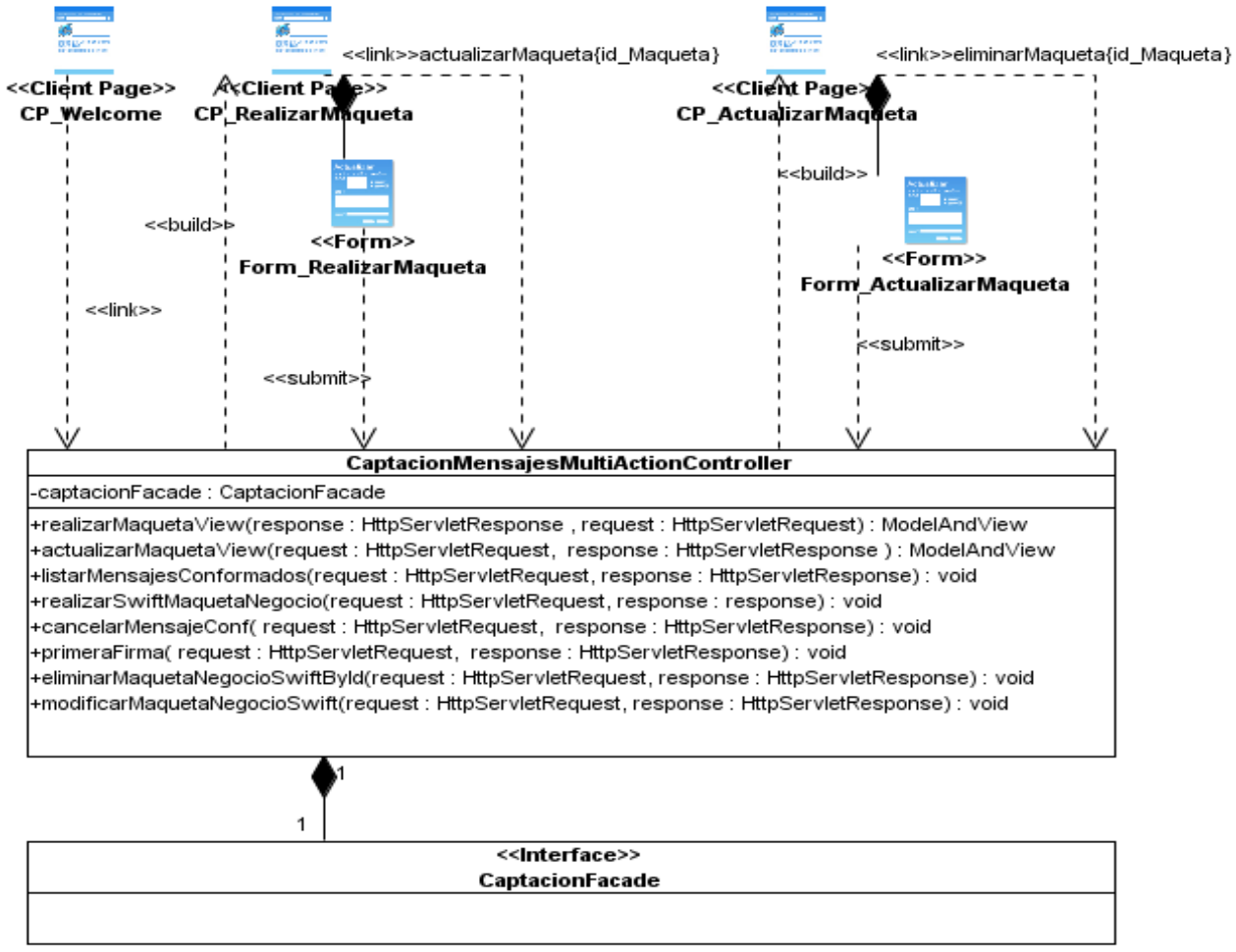


Ilustración 6: Diagrama de clases del diseño (Spring MVC).

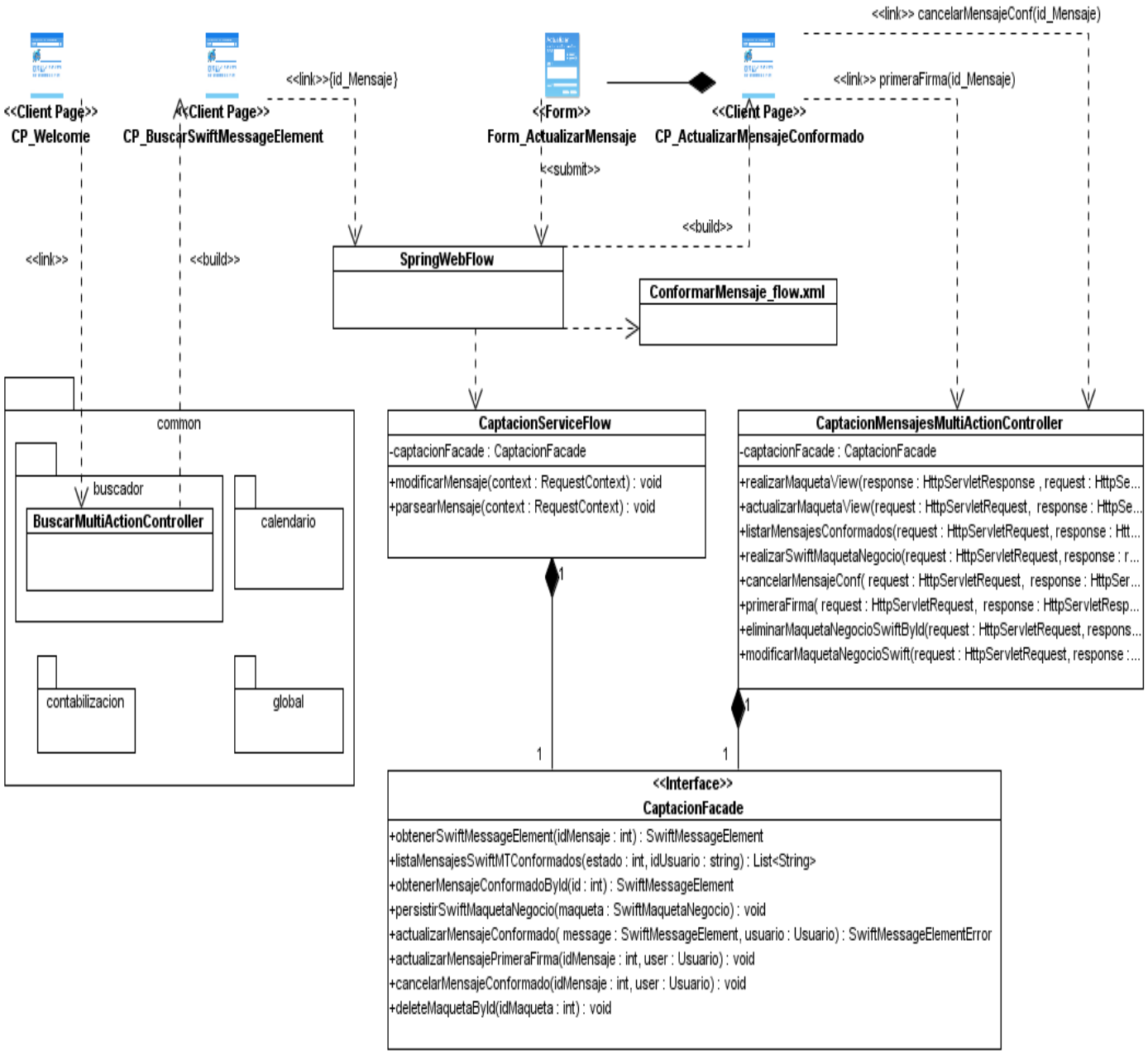


Ilustración 3: Diagrama de clases del diseño (Spring WebFlow y Spring MVC).

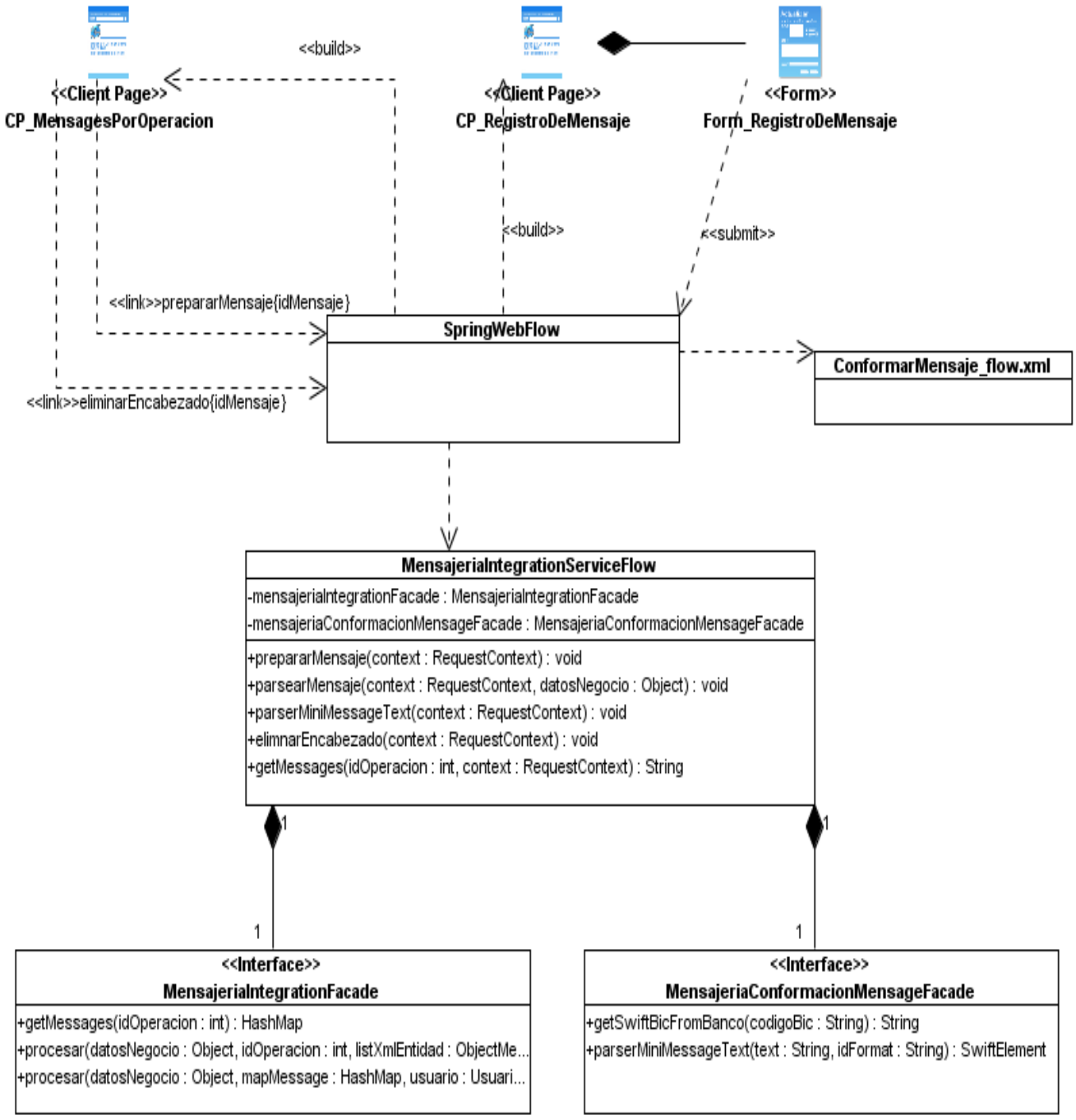


Ilustración 4: Diagrama de clases del diseño (Spring WebFlow).

Diseño de la Capa de Negocio

Las clases que componen la capa de negocio se encuentran declaradas en el Manager y sus funcionalidades para solucionar el negocio están implementadas en el ManagerImpl.

Cada módulo tiene una Fachada. La Fachada es la que brinda las funcionalidades a la capa de presentación, así la capa de presentación no se entera de la verdadera implementación de los métodos. Una Fachada puede interactuar con el resto de los módulos o subsistemas a través de las Fachadas de dichos módulos y subsistemas.

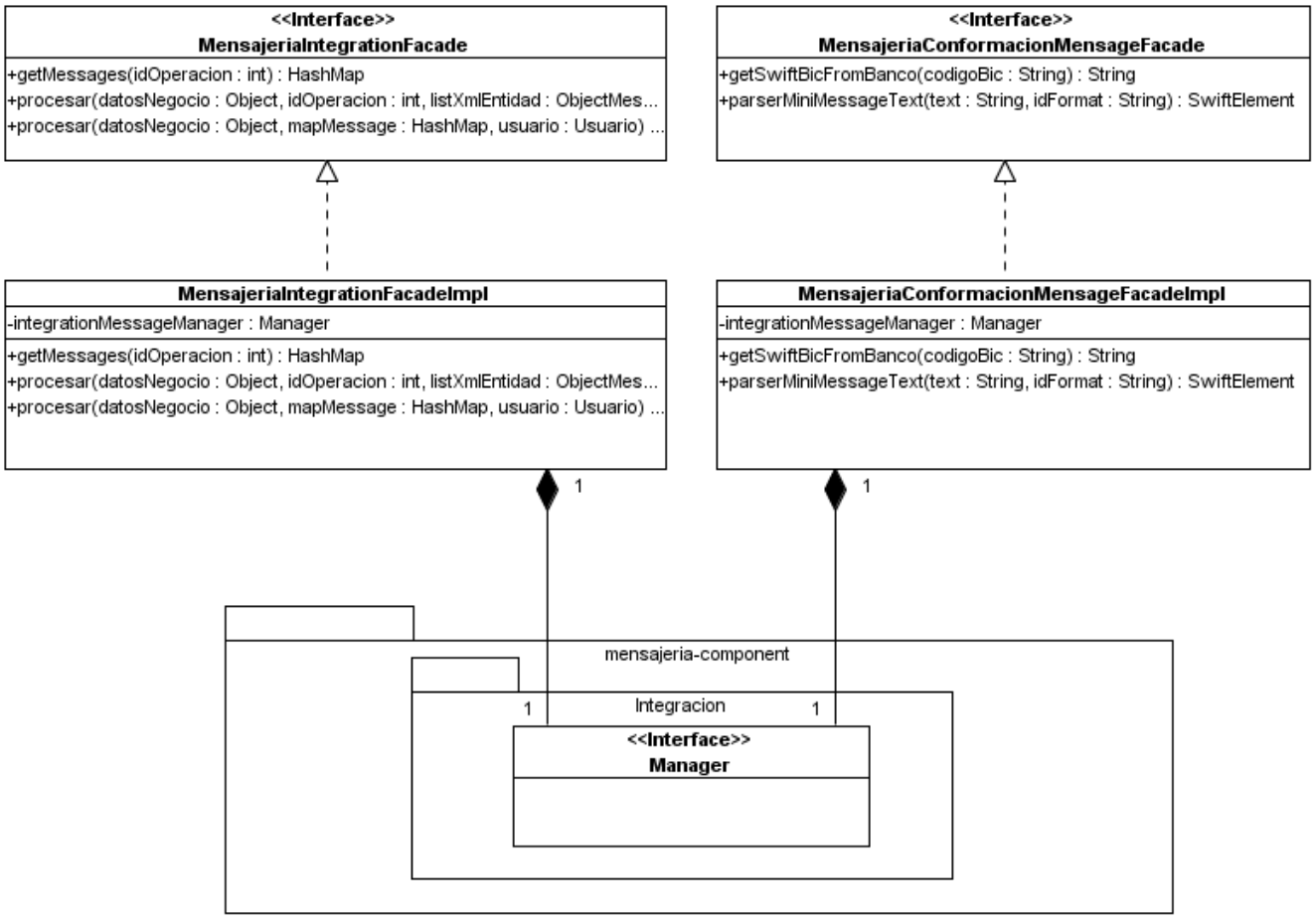


Ilustración 9: Capa de Negocio del Módulo Captación (Integración).

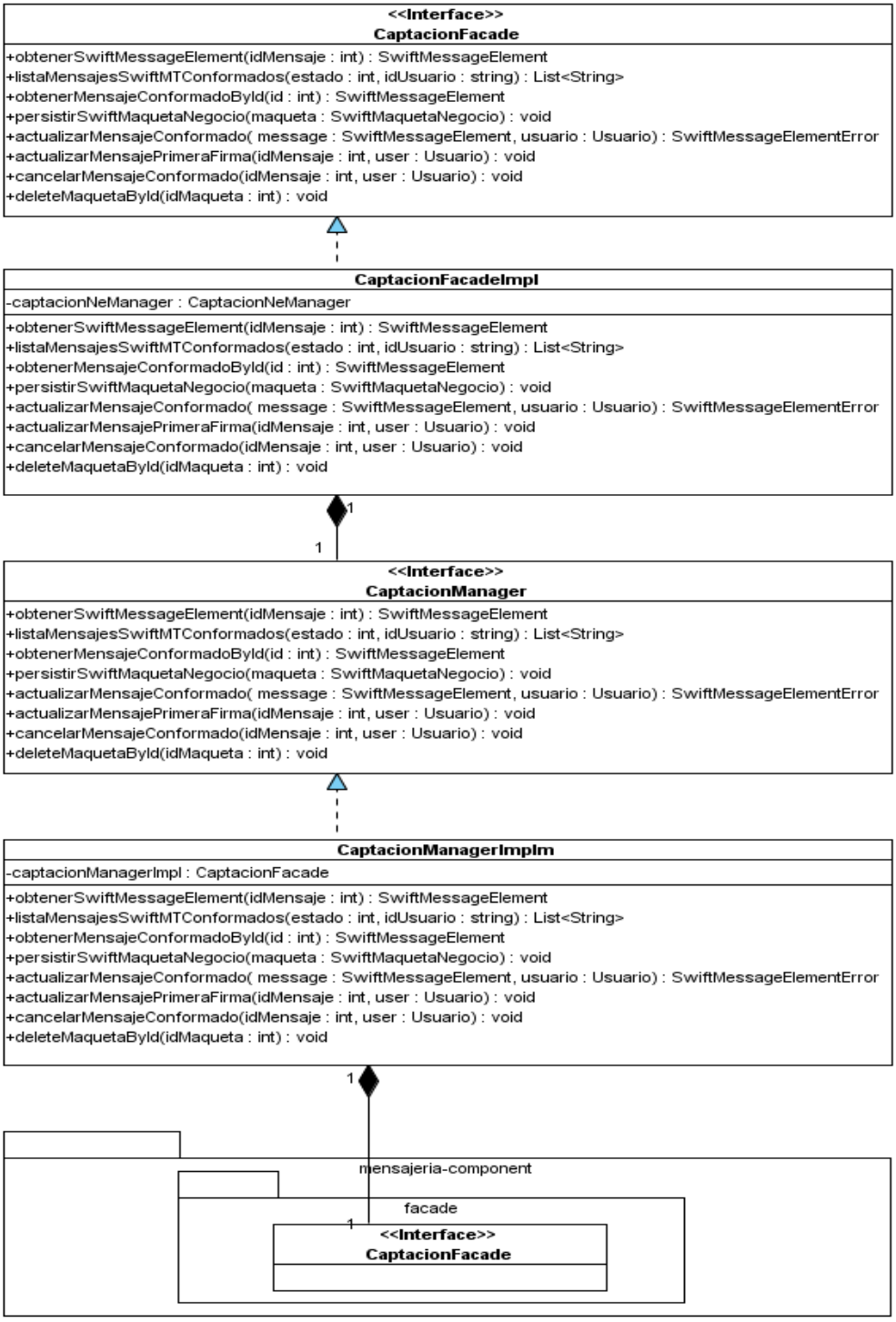


Ilustración 5: Capa de Negocio del Módulo Captación.

La clase *CaptacionManagerImplm* utiliza las funcionalidades que brinda el paquete *mensajeria-component* mediante la clase interface *CaptacionFacade* del paquete *facade*, esta clase se encargada de brindar un punto de entrada a las funcionalidades de dicho componente, que es el encargado de resolver toda la lógica de negocio del Subsistema de Mensajería.

Las clases M*ensajeriaIntegrationFacadeImpl* y *MensajeriaConformacionMensageFacadeImpl* utilizan las funcionalidades que brinda el paquete *mensajeria-component* mediante la clase interface *Manager* del paquete *integracion*, esta clase también está encargada de brindar un punto de entrada a las funcionalidades de dicho componente.

Diagramas de Secuencia del Módulo Captación de Mensajes

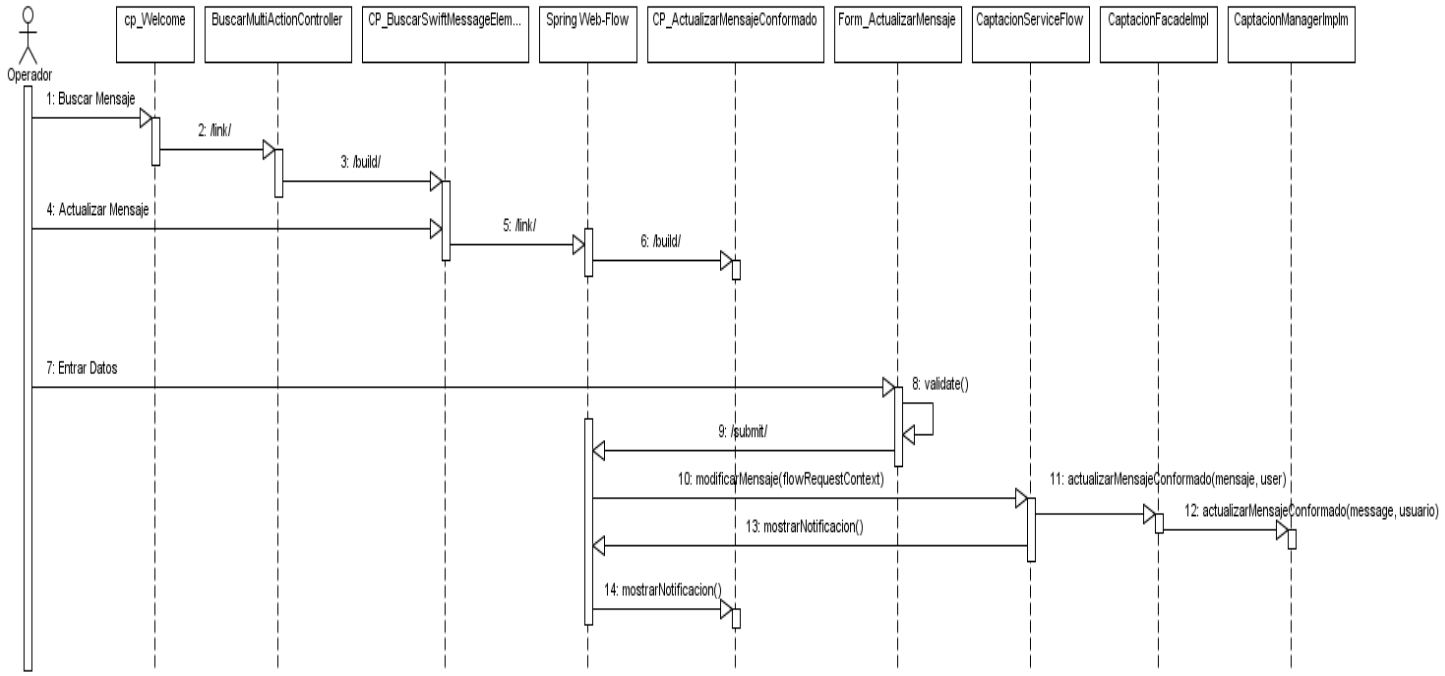


Ilustración 11: Diagrama de secuencia: Actualizar Mensaje.

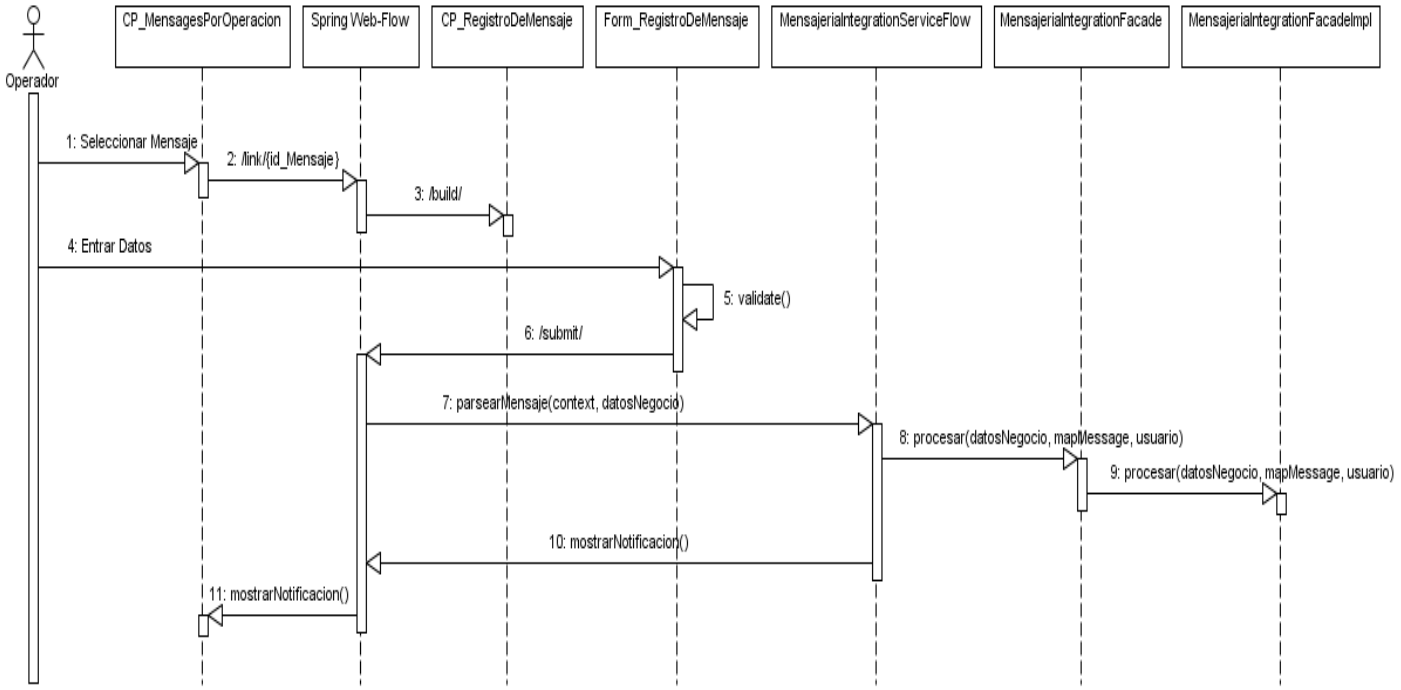


Ilustración 12: Diagrama de secuencia: Conformar Mensaje a partir de la aplicación de negocio.

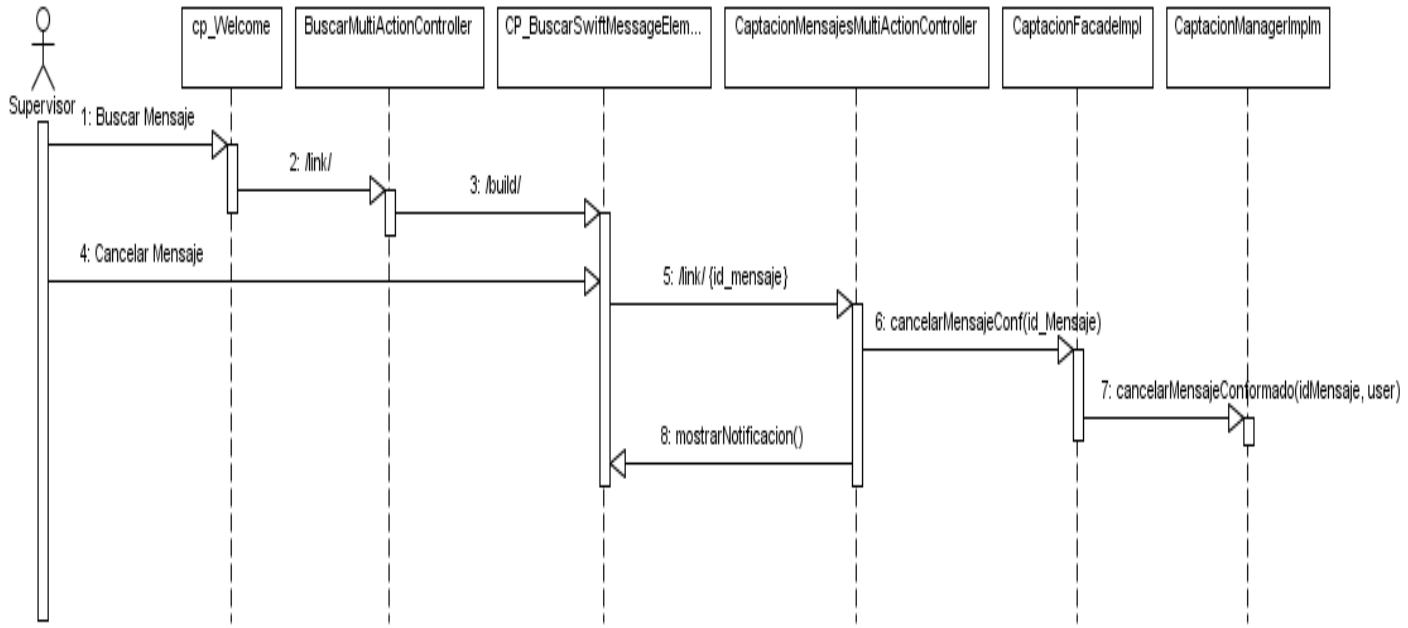


Ilustración 13: Diagrama de secuencia: Cancelar Mensaje.

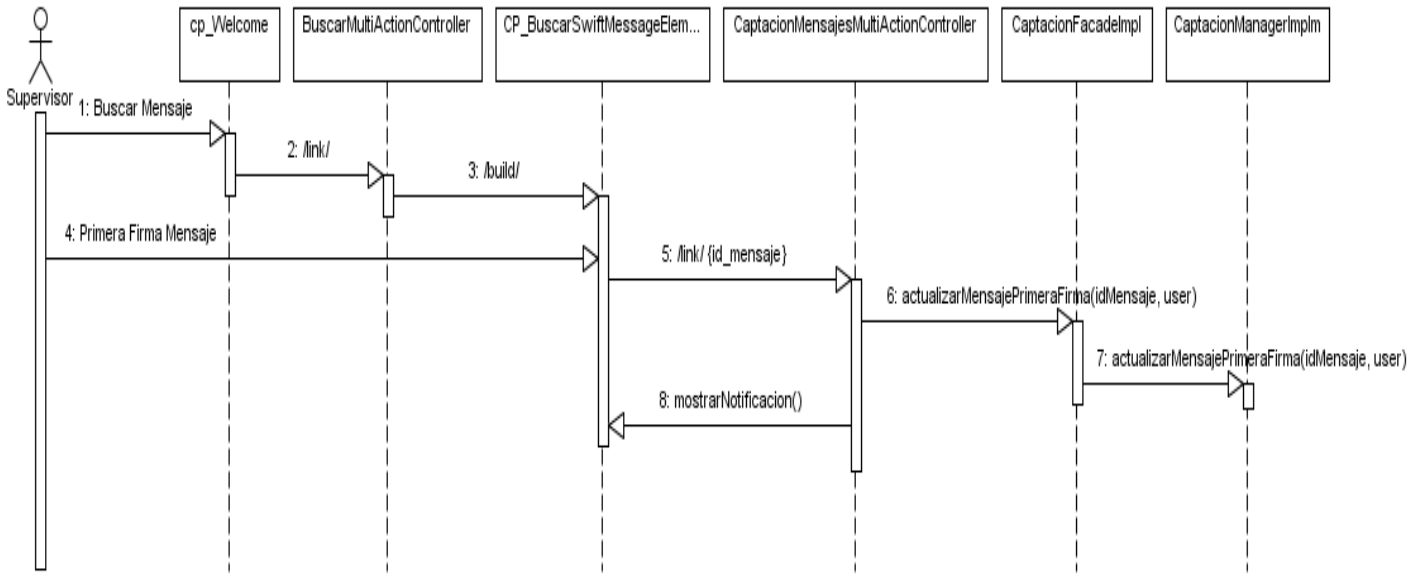


Ilustración 14: Diagrama de secuencia: Cambiar Estado de un Mensaje.

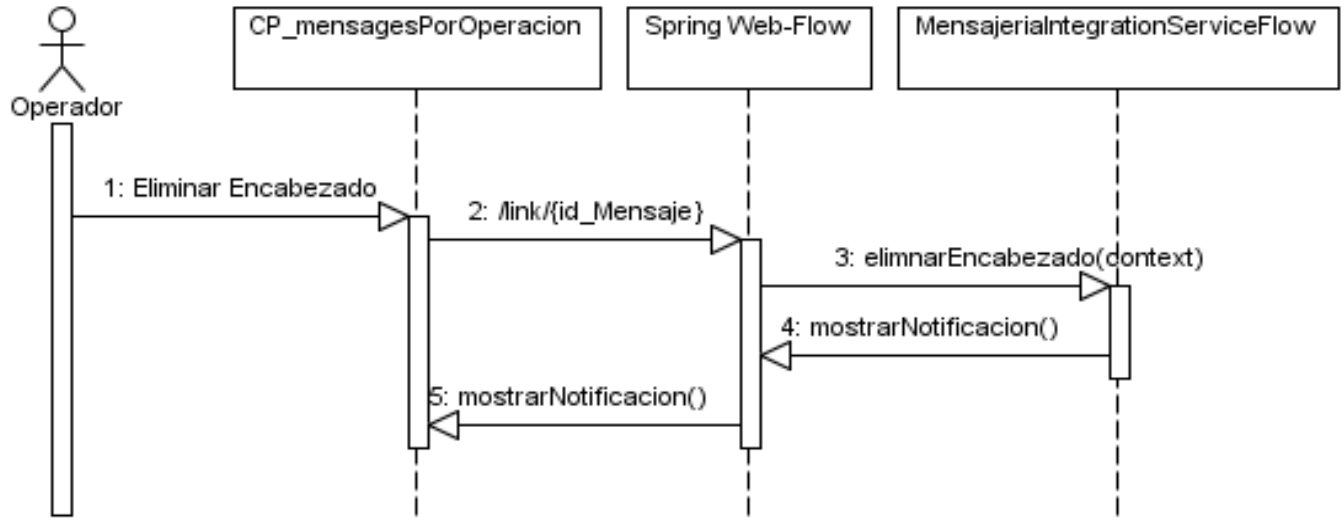


Ilustración 15: Diagrama de secuencia: Eliminar Encabezado de un Mensaje.

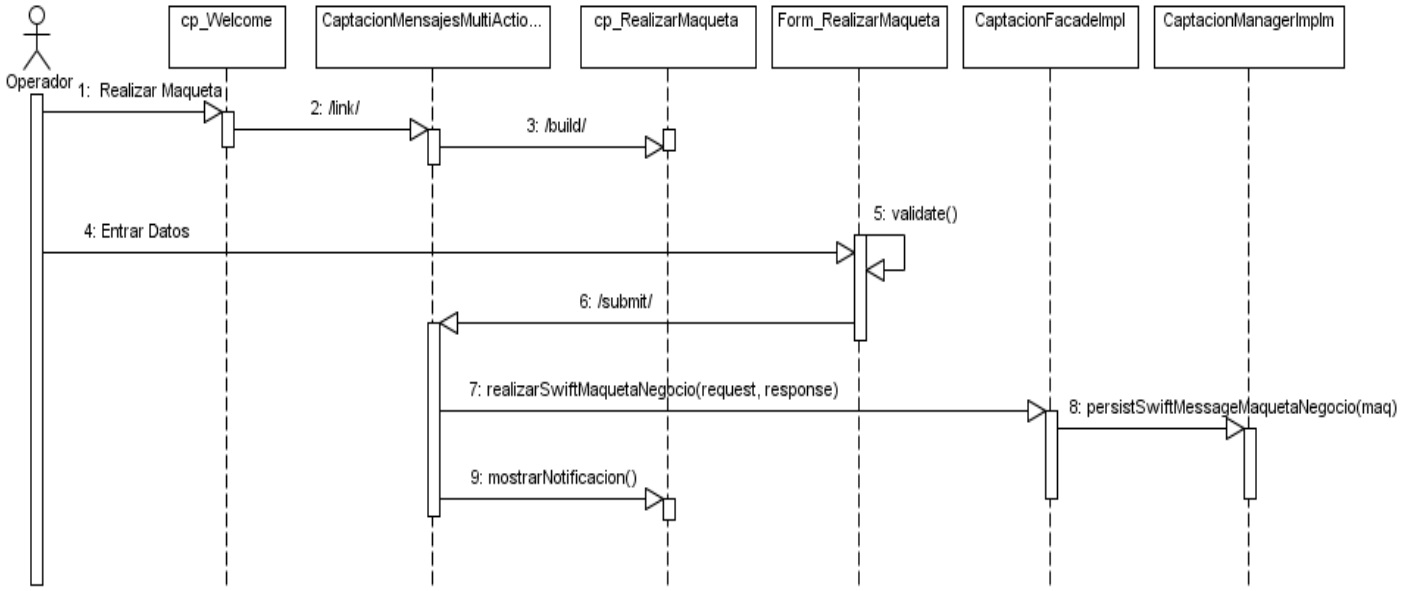


Ilustración 16: Diagrama de secuencia: Realizar Maqueta.

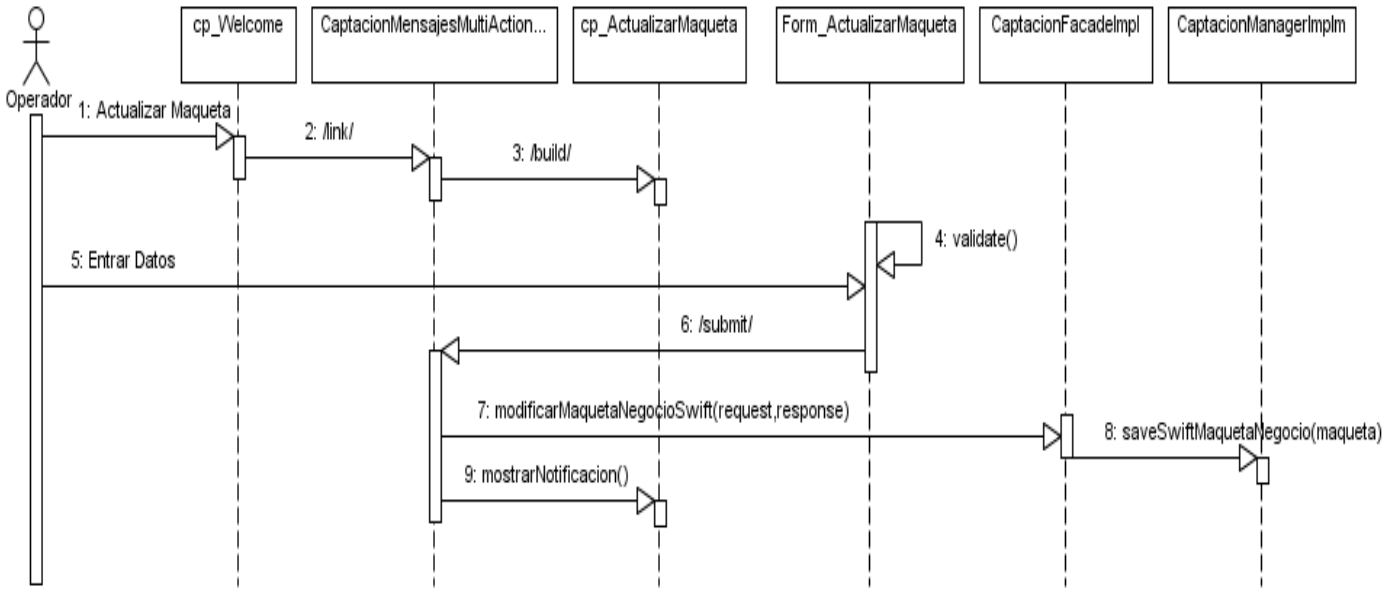


Ilustración 17: Diagrama de secuencia: Actualizar Maqueta.

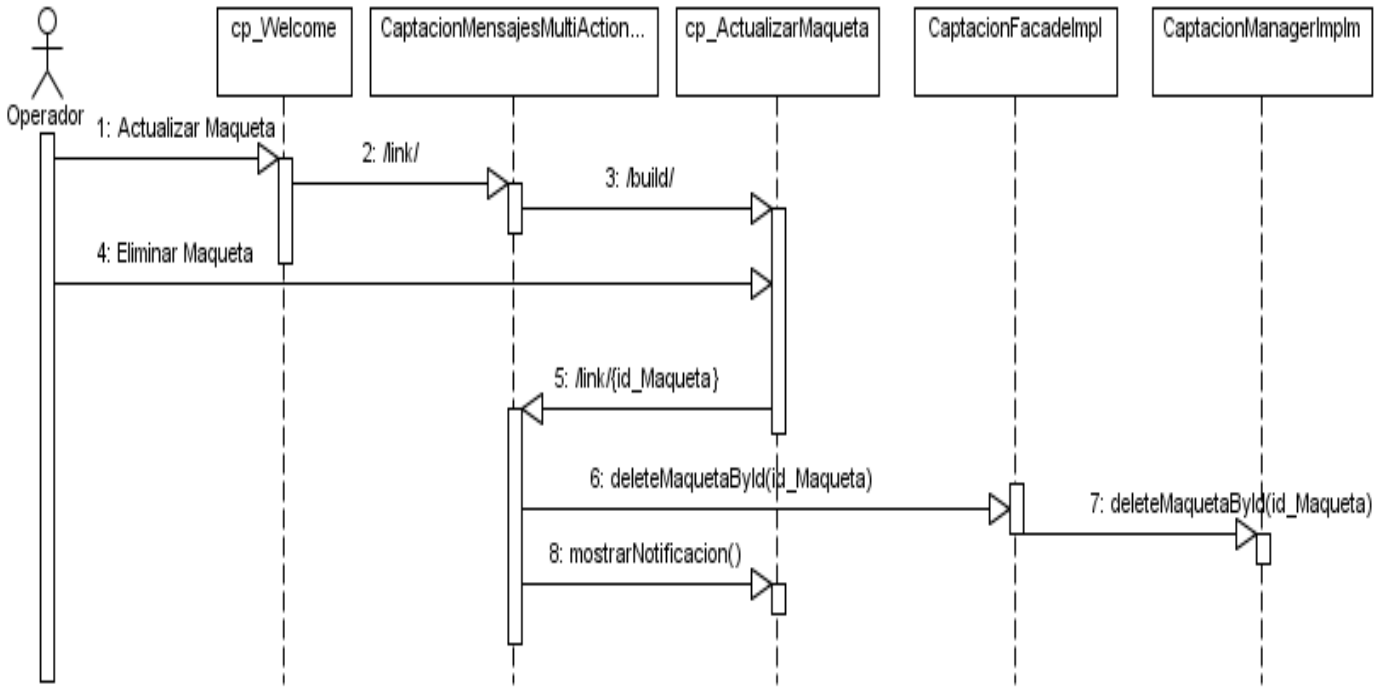


Ilustración 18: Diagrama de secuencia: Eliminar Maqueta.

Conclusiones Parciales

El análisis de la arquitectura del proyecto QUARXO y el diseño del Módulo Captación de Mensajes realizado en este capítulo, facilitó la comprensión de la estructura de paquetes existentes en el módulo, así como un mejor entendimiento del diseño del mismo.

Todo lo realizado en este capítulo incide de manera notable en el siguiente, ya que se logra una uniformidad y organización a la hora de implementar la solución propuesta.

CAPÍTULO 3–IMPLEMENTACIÓN Y PRUEBA

Introducción

En el presente capítulo serán abordados todos los temas referentes a la implementación y pruebas del sistema. Se muestra el diagrama de componentes y los estándares de codificación utilizados. Además, se especifican las pruebas que acreditan el correcto funcionamiento del producto.

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. 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.

En este flujo de trabajo se define la organización del código en términos de los subsistemas de implementación organizados en capas. Se implementan los elementos del diseño en términos de implementación, obteniéndose los componentes necesarios para el funcionamiento de la aplicación así como el diagrama de componentes, que conforma lo que se conoce como un modelo de implementación al describir los componentes a construir, su organización y dependencia entre nodos físicos en los que funcionará la aplicación.

Diagrama de componentes

El diagrama de componentes muestra las relaciones entre las partes físicas y reemplazables del sistema, por tanto, expresa las dependencias existentes entre estas estructuras denominadas componentes.

Se utilizan para modelar la vista estática de un sistema. Muestran la organización y las dependencias lógicas entre un conjunto de componentes de software, sean estos componentes de código fuente, librerías, binarios o ejecutables.

Los diagramas de componentes muestran el conjunto de componentes y sus relaciones. Un componente se corresponde con una o más clases, interfaces o colaboraciones. El siguiente diagrama muestra los componentes del módulo siguiendo la arquitectura Modelo Vista Controlador.

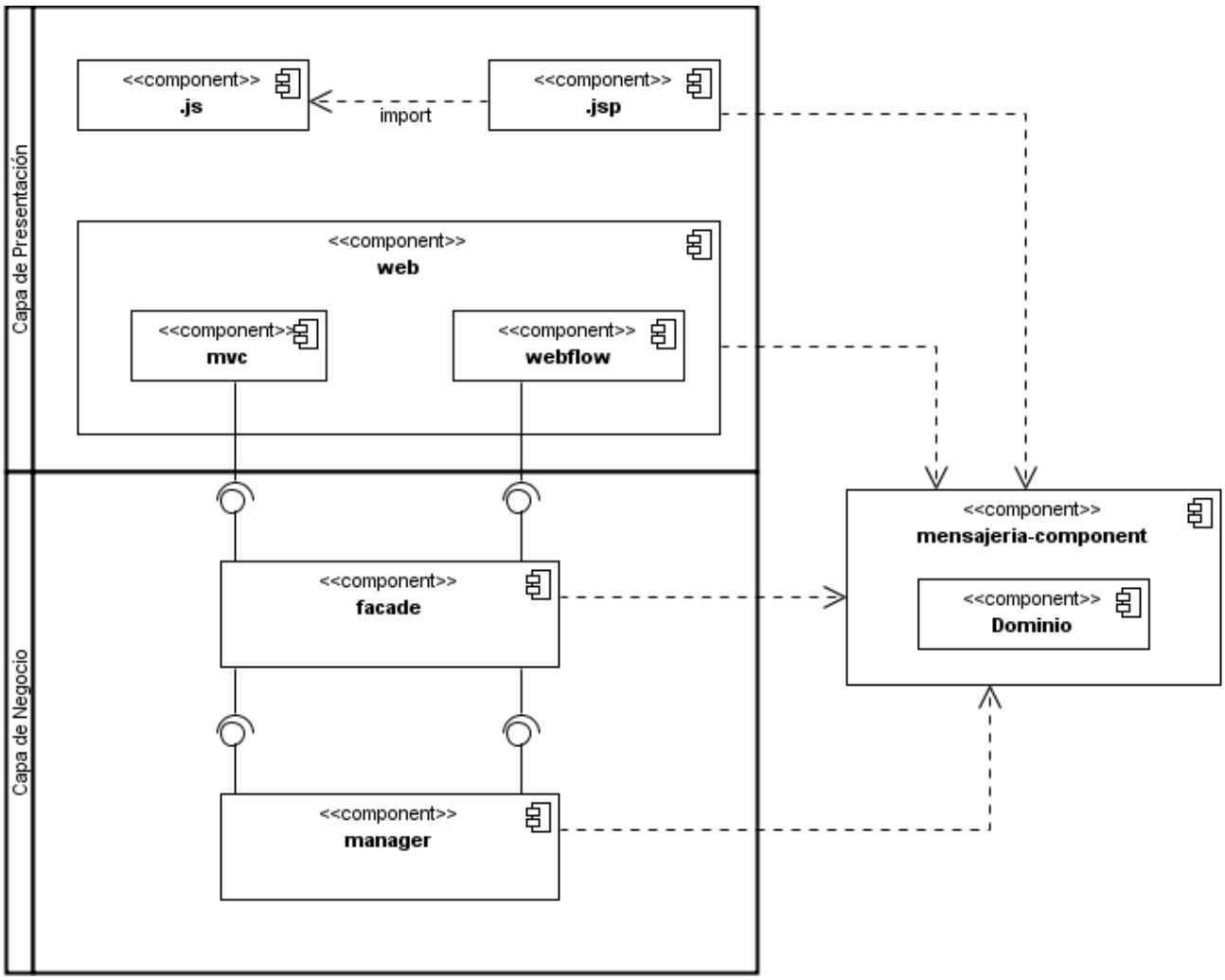


Ilustración 6: Diagrama de Componentes.

Estándares de Codificación

Los estándares de codificación son pautas de programación que no están enfocadas a la lógica del programa, sino a su estructura y apariencia física para facilitar la lectura, comprensión y mantenimiento del código. Un estándar de programación define la nomenclatura de las variables, objetos, métodos y funciones, teniendo que ver además, con el orden y legibilidad del código escrito.

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[8].

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[9].

Nomenclatura según el tipo de clases

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.

[8] **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.

[9] **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.

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.

Descripción de las principales clases a utilizar

A continuación se describirán las clases más importantes desde el punto de vista funcional para el módulo Captación de mensajes.

Nombre: CaptacionMensajesMultiActionController
Tipo de clase: Controladora

Atributo	**Tipo**
captacionFacade	CaptacionFacade
buscarMultiAction	BuscarMultiActionController
Para cada responsabilidad:	
Nombre:	**Descripción**:
realizarMaquetaView(HttpServletRequest request, HttpServletResponse response)	Permite redireccionar a la vista en la cual se crearan las maquetas que se guardaran en el sistema.
actualizarMaquetaView(HttpServletRequest request, HttpServletResponse response)	Permite redireccionar a la vista en la cual se actualizarán las maquetas existentes.
listarMensajesConformados (HttpServletRequest request, HttpServletResponse response)	Permite obtener un listado con todos los mensajes que se han confirmado como aceptados en el sistema, es decir, que no tienen errores.
realizarSwiftMaquetaNegocio(HttpServletRequest request, HttpServletResponse response)	Método encargado de procesar las maquetas que se realizan a partir de los mensajes confeccionados.
cancelarMensajeConf(HttpServletRequest request, HttpServletResponse response)	Permite cancelar un mensaje elaborado previamente.
primeraFirma(HttpServletRequest request, HttpServletResponse response)	Permite el cambio de estado de los mensajes a primera firma.
eliminarMaquetaNegocioSwiftById(HttpServletRequest request, HttpServletResponse response)	Permite eliminar las maquetas anteriormente elaboradas.
modificarMaquetaNegocioSwift (HttpServletRequest	Permite modificar las maquetas

request, HttpServletResponse response)	anteriormente elaboradas.

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.
persistirSwiftMaquetaNegocio(SwiftMaquetaNegocio maqueta)	Método que se encarga de guardar las maquetas que se elaboran en la aplicación.
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.
deleteMaquetaById(HttpServletRequest request, HttpServletResponse response)	Permite eliminar una maqueta anteriormente elaborada a partir del id de la misma.

Aspectos Principales de la Implementación

Utilización del Framework Spring WebFlow

El Framework Spring WebFlow permite operar la navegación web y el control de los flujos complejos de la aplicación. En el desarrollo del módulo Captación se hizo uso del mismo para darle solución a las funcionalidades conformar mensaje y actualizar mensaje lo que facilito la gestión de los mensajes manteniendo un flujo común que nos permitiera dirigirnos a diferentes estados de la vista trabajando siempre con el mismo objeto mensaje. A continuación haremos una pequeña descripción del funcionamiento de los flujos tomando como ejemplo el que responde a la funcionalidad conformar mensaje.

Descripción del flujo de WebFlow

El primer paso es la declaración de la variable ***messageResultTransformation*** que representa el objeto mensaje que se estará utilizando a lo largo de todo el flujo.

```
<var     name=" messageResultTransformation "
         class="cu.uci.finixu.tranformation.model.MessageResultTransformation" />
```

Todas las funcionalidades presentes en el flujo son implementadas en la clase ***mensajeriaIntegrationServiceFlow***, la misma se utiliza para evitar la interacción directa desde el flujo con el negocio, haciendo función de fachada. Nótese que estas funcionalidades declaradas en el flujo no necesitan declararse con los parámetros correspondientes, ya que, si reciben el ***RequestContext*** como único parámetro y Spring WebFlow es capaz de reconocerlo.

El flujo comienza en la vista ***opcionesMensaje*** conteniendo las dos transiciones que se pueden generar a partir de lo que decida 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>
```

En caso de que el usuario decida crear el mensaje, se redireccionaría a la vista ***mensajePorOperacion*** la cual constituye el estado más importante del flujo conteniendo las diferentes transiciones a partir de los posibles eventos que se puedan generar por el usuario.

```
<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 perteneciente a la operación especificada y con los datos de la operación contable que se realizó.

Método encargado de realizar dicha funcionalidad:

```
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 pasamos 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 redireccionaría 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 redireccionaría a la vista ***registroDeMensajes*** en la cual se llena el mensaje con los valores que 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="mensajeriaIntegrationServiceFlow.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 redireccionando hacia la página principal.

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

Prueba

Las pruebas del software son el proceso que permite verificar y mostrar la calidad de un producto, a través de resultados registrables que proporcionan una evaluación.

Dentro de los diversos métodos de pruebas se encuentran:

Las pruebas de caja negra, las cuales verifican las especificaciones funcionales sin tener en cuenta la estructura interna del programa y son realizadas sin el conocimiento interno del producto. El objetivo es demostrar que las funciones del software son operativas, que las entradas se aceptan de forma adecuada y se produce un resultado correcto, y que la integridad de la información externa se mantiene, saber qué es lo que hace el software pero sin entrar en detalles de código, es decir, que es lo que hace, y no cómo lo hace. Por ello se realizan sobre la interfaz del sistema controlando los datos de entrada y de salida.

Las pruebas de caja blanca, denominadas pruebas de caja de cristal es un método de diseño de casos de prueba que usa la estructura de control del diseño procedimental para obtener los casos de prueba. (Pressman, 2002)

Se comprueban los caminos lógicos del software. Examina el programa en varios puntos para determinar si el estado real coincide con el esperado (sobre el código). Así como las pruebas de caja negra ejercitan los requisitos funcionales desde el exterior del módulo, las de caja blanca están dirigidas a las funciones

internas. Las pruebas de caja blanca se llevan a cabo en primer lugar, sobre un módulo concreto, para luego realizar las de caja negra sobre varios subsistemas.

Este método de prueba requiere el conocimiento de la estructura interna del programa y se debe garantizar como mínimo que:(Pressman, 2002)

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

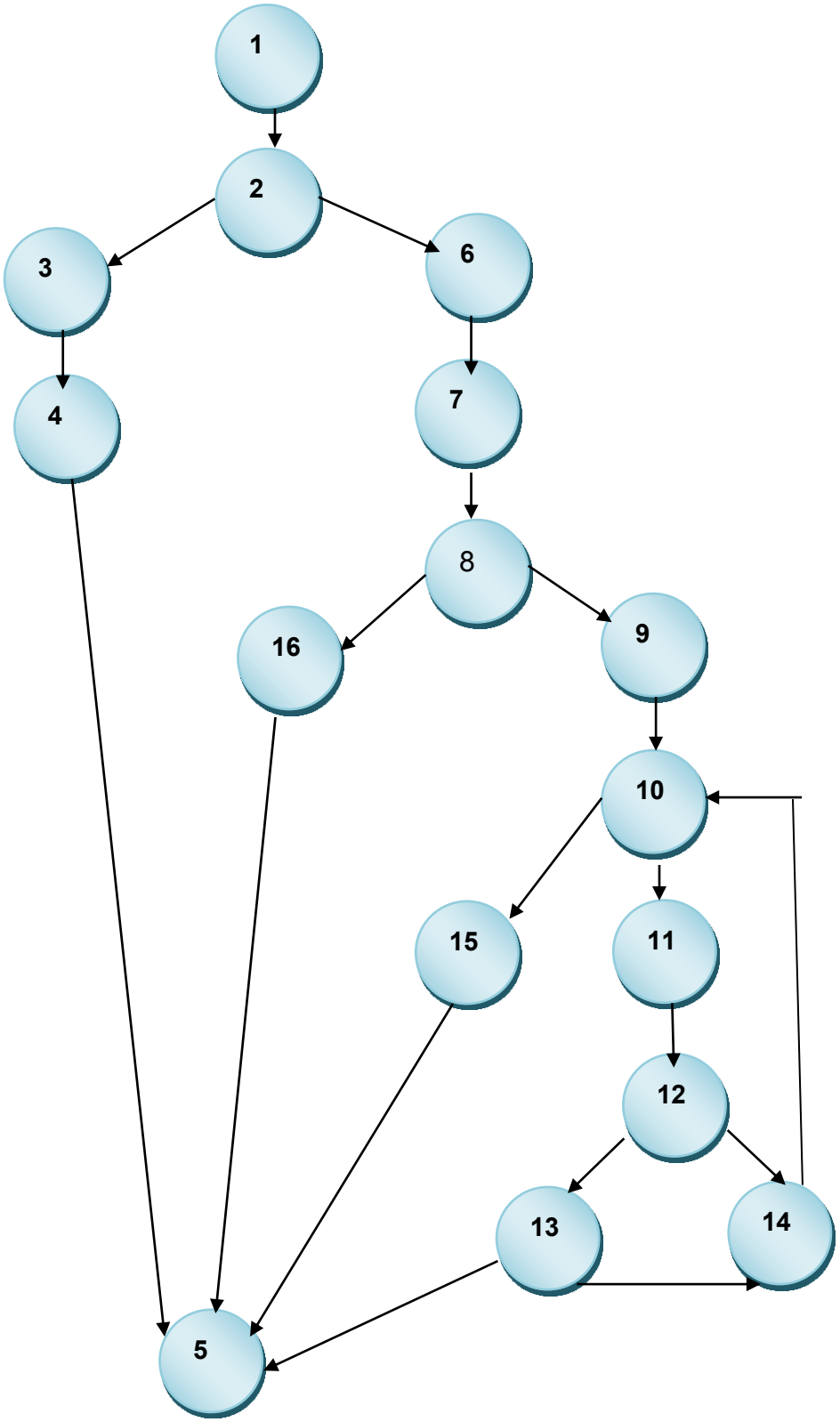
Con la realización de las pruebas se pretende demostrar que hay errores, en ningún momento es posible asegurar lo contrario. El principal objetivo al diseñar las pruebas es tratar de ejecutar todos los flujos posibles, para descubrir la mayor cantidad de errores, haciéndolo con la menor cantidad de tiempo y esfuerzo.

Aplicación de la Prueba de Caja Blanca

A continuación se analizará el método buildChildrenTree contenido en la clase ConformarMensajeMultiaccionController, este recibe un elemento SWIFT como parámetro y se encarga de determinar todos los elementos SWIFT hijos. Esta funcionalidad es invocada como parte del método que construye el árbol del mensaje SWIFT que se desea generar.

```
public JSONObject buildChildrenTree(SwiftElement element) {

        JSONObject child = new JSONObject(); //1
        if (element instanceof SwiftComponent) { //2
                if (element.isVisual()) { //3
                        child.put("label", element.getName() + ":" + element.getValue()); //4

                        child.put("idFormat", element.getIdFormat()); //4
                        return child; //5
                } else {
                        return null; //5
                        }
        } else {
                if (element.isVisual()) { //6
                        child.put("label", element.getName() + ":" + element.getValue()); //7

                        child.put("idFormat", element.getIdFormat()); //7

                        if (hasVisualChildren(element)) { //8
                                JSONArray array = new JSONArray(); //9

                                for (SwiftElement childElement
                                        : element.getListSwiftElement()) { //10

                                    JSONObject subChild = buildChildrenTree(childElement); //11

                                    if (subChild != null) { //12
                                        array.add(subChild); //13
                                    }
                                } //14
                                child.put("children", array); //15
                                return child;//5
                        } else {
                                String value = element.getValue(); //16
                                child.put("label", child.get("label")
                                + ": " + value); //16
                                return child; //5
                                }
                        } else {
                                return null; //5
                        }
                }
        }
```

Grafo de flujo asociado al código:



Complejidad Ciclomática:

La complejidad ciclomática es una métrica de software útil, pues proporciona una medición cuantitativa de la complejidad lógica de un programa. El valor calculado como complejidad ciclomática define el número de caminos independientes del conjunto básico de un programa y nos da un límite superior para el número de pruebas que se deben realizar para asegurar que se ejecute cada sentencia al menos una vez. Un camino independiente es cualquier camino del programa que introduce por lo menos un nuevo conjunto de sentencias de procesamiento o una nueva condición. El mismo, debe moverse por lo menos por una arista no recorrida anteriormente.

El cálculo de complejidad ciclomática se efectúa mediante tres vías o fórmulas de manera que se justifique resultado, siendo el mismo en cada uno de los casos:

Caso1:

V (G) = (A – N) + 2
Siendo “A” la cantidad total de aristas y “N” la cantidad total de nodos.
V (G) = (20 - 16) + 2
V (G) = 6.

Caso2:

V (G) = P + 1
Siendo “P” la cantidad total de nodos predicados[10].
V (G) = 5 + 1
V (G) = 6.

[10] ***Nodos Predicados:*** son los nodos de los cuales parten dos o más aristas.

Caso3:

V (G) = R
Siendo “R” la cantidad total de regiones, se incluye el área exterior del grafo, contando como una región más.
V (G) = 6

El cálculo efectuado mediante las tres fórmulas ha dado el mismo valor, por lo que se puede decir que la complejidad ciclomática del código es 6, lo que significa que existen seis posibles vías por donde el flujo puede circular. El valor representa el límite mínimo del número total de casos de pruebas para el procedimiento tratado.

Seguidamente es necesario representar los caminos básicos por los que puede recorrer el flujo:

Camino básico #1: 1 – 2 – 3 – 5
Camino básico #2: 1 – 2 – 3 – 4 – 5
Camino básico #3: 1 – 2 – 6 – 7 – 8 – 16 – 5
Camino básico #4: 1 – 2 – 6 – 7 – 8 – 9 – 10 – 15 – 5
Camino básico #5: 1 – 2 – 6 – 7 – 8 – 9 – 10 – 11 – 12 – 13 – 14 – 10 – 15 – 5
Camino básico #6: 1 – 2 – 6 – 7 – 8 – 9 – 10 – 11 – 12 – 14 – 10 – 15 – 5

Caso de prueba para el camino básico #1:

Descripción: Se debe generar un JSON que contenga el nombre y el valor del elemento que se introduce como parámetro del método.

Condición de ejecución: Es necesario que el elemento pasado por parámetro sea de tipo SwiftComponent y que no tenga propiedades visuales.

Entrada:
element = componenteLlaveCerrada

Resultados esperados: Teniendo en cuenta el dato pasado por parámetro se espera que el elemento que se retorne sea un JSON nulo. El resultado obtenido fue correcto.

Caso de prueba para el camino básico #2:

Condición de ejecución: Es necesario que el elemento pasado por parámetro sea de tipo SwiftComponent y que tenga propiedades visuales.

Entrada:
element = componente24x

Resultados esperados: Teniendo en cuenta el dato pasado por parámetro se espera que el elemento que se retorne sea un JSON que tenga los valores (componente24x: 24x). El resultado obtenido fue correcto.

Caso de prueba para el camino básico #3:

Condición de ejecución: Es necesario que el elemento pasado por parámetro no sea de tipo SwiftComponent, que tenga propiedades visuales y que no tenga hijos con propiedades visuales.

Entrada:
element = subCampoDebitoCredito

Resultados esperados: Teniendo en cuenta el dato pasado por parámetro se espera que el elemento que se retorne sea un JSON que tenga los valores (subCampoCodigo: D). El resultado obtenido fue correcto.

Caso de prueba para el camino básico #4:

Condición de ejecución: Es necesario que el elemento pasado por parámetro no sea de tipo SwiftComponent, que tenga propiedades visuales y que tenga un hijo con propiedades visuales.

Entrada:
element = Campo22AObligatorio

Resultados esperados: Teniendo en cuenta el dato pasado por parámetro se espera que el elemento que se retorne sea un JSON que contenga los valores del campo 22 y de su hijo. El resultado obtenido fue correcto.

Caso de prueba para el camino básico #5:

Condición de ejecución: Es necesario que el elemento pasado por parámetro no sea de tipo SwiftComponent, que tenga propiedades visuales y que tenga una lista de hijos con propiedades visuales.

Entrada:
element = Campo53AObligatorio

Resultados esperados: Teniendo en cuenta el dato pasado por parámetro se espera que el elemento que se retorne sea un JSON que contenga los valores del campo 53A y de la lista de hijos. El resultado obtenido fue correcto.

Caso de prueba para el camino básico #6:

Condición de ejecución: Es necesario que el elemento pasado por parámetro no sea de tipo SwiftComponent, que tenga propiedades visuales y que tenga una lista de hijos con propiedades visuales a los cuales no se les haya introducido valor, por lo que esta lista sería nula.

Entrada:
element = grupoComponente53AorDorJ

Resultados esperados: Teniendo en cuenta el dato pasado por parámetro se espera que el elemento que se retorne sea un JSON que contenga los valores del grupoComponente53AorDorJ solamente, ya que a sus hijos no se les introdujo ningún valor. El resultado obtenido fue correcto.

Aplicación de la Prueba de Caja Negra o Funcional.

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.
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.

Algunas técnicas utilizadas en la prueba de caja negra son:

- ✓ **Partición de Equivalencia:** Técnica que divide el campo de entrada de un programa en clases de datos de los que se pueden derivar casos de prueba.
- ✓ **Análisis de Valores Límite*:*** La experiencia muestra que los casos de prueba que exploran las condiciones límite producen mejor resultado que aquellos que no lo hacen. Las condiciones límite son aquellas que se hallan en los márgenes de la clase de equivalencia, tanto de entrada como de salida.
- ✓ **Grafos de causa-efecto:** Es una técnica que 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.

Resultado de las pruebas

Luego de la realización de los métodos de prueba, como parte de las pruebas internas realizadas al Módulo Captación se obtuvieron resultados satisfactorios desde el punto de vista interno y funcional, atendiendo al correcto comportamiento del mismo ante diferentes situaciones.

Es importante destacar que el Módulo Captación fue probado y liberado por el departamento de calidad de la universidad (CALISOFT), posteriormente paso a la fase de pruebas de aceptación realizadas en el Banco Nacional de Cuba donde fue aprobado por lo que se concluye que el Módulo Captación se encuentran en completa disposición para su utilización en el Banco nacional de Cuba.

Conclusiones Parciales

Se puede concluir que, luego de este capítulo se ha logra un mejor entendimiento del sistema desde el punto de vista de implementación lo que posibilitó el cumplimiento de los requisitos funcionales. Además, los resultados obtenidos en las pruebas realizadas, ejemplifican la calidad del trabajo efectuado en la implementación del Módulo Captación de Mensajes, también muestran pleno cumplimiento de los requisitos funcionales y no funcionales.

CONCLUSIONES GENERALES

Durante el desarrollo del presente trabajo se analizaron diferentes cuestiones que aportaron conocimientos sobre los Sistemas de Gestión Bancaria, centrándonos en los procesos de comunicación financieros a los que estos deben someterse. El estudio de sistemas similares aportó ideas en cuanto a plataforma y herramientas a utilizar en la implementación del software.

Se describieron las herramientas, lenguajes y metodología empleados en la solución, los cuales poseen características que agilizan y facilitan el proceso de desarrollo de software. Las herramientas ayudaron a realizar los modelos y diagramas útiles para la implementación de la propuesta de solución, el lenguaje UML facilitó la realización de los modelos, el lenguaje de programación proporciono la codificación y la metodología registró y documentó todo lo relacionado con la solución.

El diseño del Módulo Captación dotó al desarrollador de una vista lógica del sistema y los elementos que lo componen, lo que posibilitó la implementación del mismo respetando las pautas definidas, lo cual contribuyó a la obtención de un código limpio y flexible a futuros cambios.

Una vez integrado el Módulo Captación al sistema QUARXO, luego de las pruebas de calidad aplicadas para la detección y corrección de errores existentes, (tanto por expertos en calidad como por los clientes), se demostró su validez, el cumplimiento de las características requeridas y de los objetivos propuestos.

La utilización de este módulo contribuye al mejor desarrollo del proceso de mensajería del Banco Nacional de Cuba facilitando la elaboración de los mensajes de manera rápida y mediante un proceso automatizado que disminuye el gasto de tiempo y de trabajo innecesario.

RECOMENDACIONES

Se recomienda que se le incorpore al sistema QUARXO como una funcionalidad más, la realización del envió de los mensajes en todo su proceso, desde su inicio, confección, aprobación y envío sin tener que interactuar con las bases del SISCOM, que es como se realiza en estos momentos.

BIBLIOGRAFÍA

- **Chaviano, Adolfo Miguel Iglesias. 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.

- ANEC. El economista de Cuba. [En línea]
http://www.eleconomista.cubaweb.cu/.

- CEPEC. Centro para la promoción del Comercio Exterior de Cuba. [En línea]
http://www.cepec.cu/economia2.php.

- Definición ABC. [En línea]
http://*www.**definicionabc**.com/*

- Eveliux, *2011. [En línea]*
http://www.exeliux.com/

- SWIFT.*2009.* *El proveedor global de servicios seguros de mensajería financiera. 2009. [En línea]*
http://www.swift.com/

- Bolsa de Valores. *2007 [En línea]*
http://bolsamexicanadevalores.com.mx/tag/fix/

- Primary. *Services, Solutions and Technology 2005 [En línea]*
*http://www.**primary**.com.ar/main.php?sec=4&n=55*

- Informatics. *The Data Integration Company. [En línea]*
http://www.informatica.com/products_services/b2b_data_transformation/Pages/index.aspx

- Clavelink. *2006. [En línea]*
*http://www.**clavelink**.com/**incentage**.php?idioma=es*

- SIBANC. 2006. *Manual de Usuario del SISCOM.* 2006.

- Barroso y Bello, 2010. Diseño e Implementación del Subsistema Cartas de Créditos Del Proyecto SAGEB.

- Jacobson, I, Booch, G y Rumbaugh, J. 2000. *El Proceso Unificado de Desarrollo de Software.* Madrid : s.n., 2000.

- Patrones. [En línea] [Citado el: 2 de febrero de 2010.] http://www.inf.utfsm.cl/~visconti/ili236/Documentos/08-Patrones.pdf.

- IGP SQLServer. Información General del Producto SQLServer 2005 [En línea]
http://www.microsoft.com/spain/sql/productinfo/overview/default.mspx

- Rumbaugh, James, Jacobson, Ivar and Booch, Grandy. *El lenguaje unificado de modelado. Manual de referencia. s.l. : Addison Wesley.*