Universidad de las Ciencias Informáticas

"Facultad 3"

Título: Solución informática para flexibilizar el sistema de intercambio de mensajes SWIFT.

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

Autor:

Emilio Ferrer Trujillo

Tutor:

Ing. Adolfo Miguel Iglesias Chaviano.

"La Habana. Junio de 2012"

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 junio del año 2012.

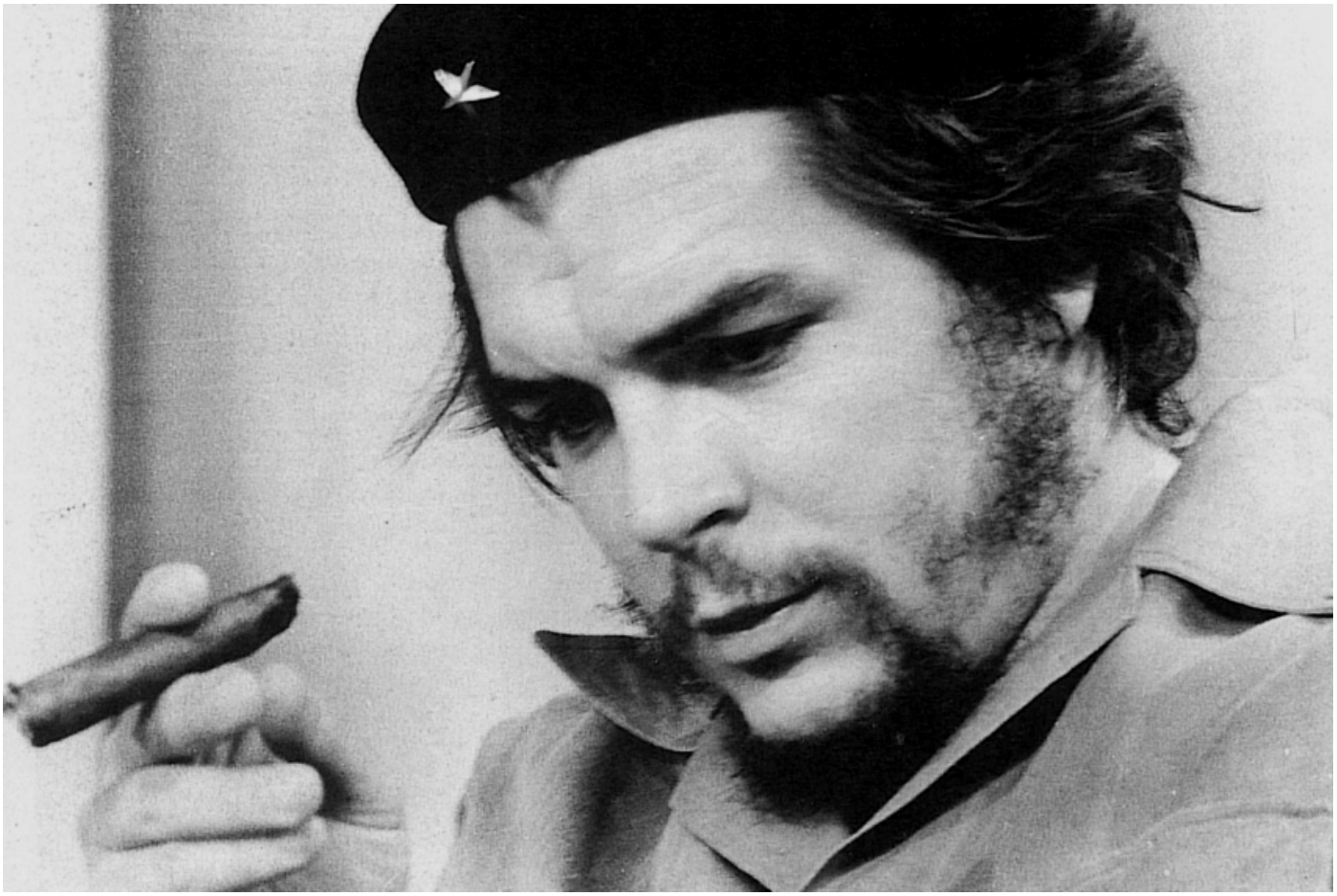
_____

Emilio Ferrer Trujillo

_____

Ing. Adolfo Miguel Iglesias Chaviano

FRASE



En la tierra hacen falta personas que trabajen más y critiquen menos, que prometan menos y resuelvan más, que esperen recibir menos y dar más, que digan mejor ahora que mañana.

Che

DEDICATORIA

Dedico este trabajo a mi madre, por ser lo más grande que tengo en la vida.

AGRADECIMIENTOS

A mi mamá por ser la mejor madre del mundo, por su ejemplo de sacrificio, valor y perseverancia. Por apoyarme siempre, por ser mi amiga, mi guía, por ser todo para mí. Por ti mami hoy soy ingeniero. Gracias una vez más por ser mi madre.

A Krysita por ser mi mejor amiga, por su amor, su apoyo, confianza, por darme ánimos cada día de seguir adelante, por querer siempre lo mejor para mí.

A Krysia, Victoria, Ángel y Angelito por toda su ayuda y preocupación, por quererme como parte de su familia.

A mis tías Mirian y Mireya por todo su apoyo a lo largo de mi carrera, por la confianza que siempre depositaron en mí, por recordarme siempre que me quieren.

A mis primos y primas Katiuska, Yisel, Yunier y Robertico, por su ayuda y apoyo, por hacerme saber que puedo contar con ustedes; y a los más chiquiticos Eriel y Alex Alain por su cariño.

A Andrés por ser un tío más y tenerme siempre presente, por su cariño, respeto y admiración.

A mi tutor Adolfo por su ayuda durante toda mi carrera, por sus consejos y su apoyo en todo momento.

A todos los integrantes del proyecto SAGEB, QUARXO o FINIXU especialmente a Yesenia, Manuel, Alain, Yovani (Leo), Jorge, Leosvel y Ariel por su ayuda y amistad.

A todo el grupo 3509 por acogerme como uno más de ustedes, a Efraín, Wilfredo, el Bode, Danis, el Vlado, Adrian, Lucrecia, Dailiana, Lilian, Evelio, Puig gracias a todos por los momentos que compartimos juntos.

A mis mejores profesores en la universidad, Yaima García García y Alain Eduardo siempre recordaré sus clases, fue un placer haber sido su alumno.

A la Revolución, a Fidel, Raúl y a la UCI por haberme dado la oportunidad de formarme como ingeniero.

RESUMEN

Actualmente el Banco Nacional de Cuba (BNC) se encuentra inmerso en la modernización de su sistema informático con el objetivo de garantizar eficiencia en sus procesos de negocio. En el año 2007 se le solicitó a la Universidad de las Ciencias Informáticas (UCI) el desarrollo de dicho sistema.

Este nuevo sistema debía soportar tanto los procesos contables como el intercambio de información financiera que se llevan a cabo en el BNC. Para hacer frente a las modificaciones que ocurren anualmente en el estándar de mensajería utilizado (estándar SWIFT) era necesario que contara con funcionalidades para gestionar las reglas semánticas que se le aplican a los mensajes SWIFT. Además debía permitir la gestión de las operaciones de negocio y la asociación de estas a los mensajes SWIFT correspondientes.

La presente investigación se basa en el diseño e implementación de los módulos Configuración de las reglas semánticas y Administración de las asociaciones de las operaciones de negocio que conforman y reciben mensajes SWIFT. Se realizó el análisis, diseño e implementación de los módulos anteriormente expuestos cumpliendo con los requisitos solicitados por el cliente. Se utilizaron herramientas y tecnologías de la plataforma Java para la implementación, por tener como ventajas que son software libre y tienen un excelente potencial para el desarrollo de aplicaciones web.

Al concluir el desarrollo se comprobó el correcto funcionamiento de los módulos, realizando pruebas estructurales y funcionales a cada uno.

Palabras claves:
Intercambio de información financiera, estándar SWIFT, reglas semánticas, programación web, software libre, análisis, diseño, implementación.

ÍNDICE

INTRODUCCIÓN

Con la creación de la Universidad de las Ciencias Informáticas (UCI) en el año 2002, comenzó un largo camino en el proceso de informatización de la sociedad cubana. Con el objetivo de hacer eficientes los procesos de negocios, desde el año 2007 se inició el desarrollo de un sistema informático para el Banco Nacional Cuba (BNC).

Los bancos para su funcionamiento hacen uso de la tecnología y la información. Contar con aplicaciones desarrolladas en computadoras posibilita un acceso rápido y fácil a la información, permitiendo realizar una buena gestión de la misma. Con el actual desarrollo de las tecnologías y el creciente cúmulo de información manejada diariamente por entidades bancarias, se hace necesario contar con un sistema capaz de gestionarla de una manera eficiente y con la disminución del esfuerzo por parte de los usuarios. Cada banco o institución financiera depende en gran medida del intercambio de información, y esta a su vez debe estar regida por estándares para establecer una comunicación entendible por ambas partes, eliminando así las barreras del lenguaje.

Actualmente el BNC cuenta con el Sistema automatizado para la Banca Internacional de Comercio (SABIC) y el Sistema Nacional para la comunicación financiera (SISCOM) con los cuales realiza las operaciones que requieren tanto la gestión del negocio como el envío y recepción de mensajes SWIFT[1].

El SABIC a pesar del papel tan fundamental que ha jugado en la automatización de los procesos del BNC, se ha quedado atrás ante los nuevos cambios en el negocio y la tecnología, dificultando el trabajo de los funcionarios dentro de la entidad; dicho software se ejecuta sobre el sistema operativo MS-DOS el cual no es compatible con SISCOM, además este último tiene como principal desventaja que es monotarea, lo que entorpece el trabajo del que opera con dicha aplicación. Con el SABIC y el SISCOM hoy en día no se satisfacen todos los requisitos que necesita el BNC, y surge la necesidad de implementar un nuevo sistema que contemple todas las funcionalidades necesarias. Con este objetivo surgió Quarxo, una aplicación Web desarrollada en la UCI, implementada sobre el núcleo del SABIC y que incluye la gestión de todos los procesos de negocios llevados a cabo por el BNC.

[1] **SWIFT:** ***Society for Worldwide Interbank Financial Telecommunication*** *(Sociedad para las Telecomunicaciones Financieras Interbancarias Mundiales)*

Quarxo cuenta con el subsistema de mensajería mediante el cual se logra integrar las operaciones contables realizadas con el envío y recepción de mensajes SWIFT, por lo que ya no es necesario introducir los mismos datos en el sistema SABIC y luego en el SISCOM, reduciendo así el riesgo de que el usuario se equivoque al realizar estas operaciones. Cuando el mensaje resultante de una operación contable está conformado se le aplican una serie de validaciones llamadas reglas semánticas, las que a su vez están regidas por la norma ISO 15022 del estándar SWIFT. (1)

El subsistema de mensajería no cuenta con una funcionalidad para gestionar desde la aplicación las modificaciones que sufren anualmente las reglas semánticas que se usan para validar los mensajes SWIFT. Además cuando surge la necesidad de añadir una nueva operación de negocio que requiera el envío o recepción de mensajes SWIFT no existe un mecanismo para gestionarlo, en cambio actualmente este proceso se realiza de forma manual en la base de datos, lo que trae consigo una inversión de tiempo considerable y las personas encargadas de realizarlas deben tener bastos conocimiento de las tablas y los datos que se necesitan modificar.

Debido a los problemas anteriormente planteados, se puede identificar como **problema a resolver**:

¿Cómo flexibilizar el sistema de intercambio de mensaje SWIFT ante los cambios que ocurren en las reglas semánticas de la norma ISO 15022 y en la asociación de las operaciones de negocio con mensajes SWIFT?

A partir del problema anteriormente expuesto se ha determinado como **objeto de estudio** el intercambio de información financiera.

Objetivo General:

Desarrollar módulos de configuración para las reglas semánticas y la asociación entre las operaciones de negocio y los mensajes SWIFT, que faciliten la actualización de dichas reglas y la creación de los mensajes.

Campo de acción:

Proceso de intercambio de mensajes SWIFT en el Banco Nacional de Cuba.

Idea a defender:

Con el desarrollo de los módulos Configuración de las reglas semánticas y Administración de las asociaciones de las operaciones de negocio que conforman y reciben mensajes SWIFT, del subsistema mensajería del Quarxo, se podrán administrar los cambios que ocurren en las reglas semánticas de la norma ISO 15022 y administrar las asociaciones de las operaciones de negocio con mensajes SWIFT.

Posible resultado:

La obtención de un componente informático que permita la creación, actualización, consulta y eliminación de las reglas semánticas, así como la asociación de dichas reglas a los mensajes SWIFT correspondientes. También se desarrollará un componente informático que permita la gestión de las asociaciones de las operaciones de negocio con los mensajes SWIFT.

Tareas de la investigación:

- Caracterización de la norma ISO 15022 del estándar SWIFT.
- Análisis de los sistemas informáticos compatibles con la norma ISO 15022.
- Descripción de los artefactos necesarios según la metodología de desarrollo de software utilizada para construir los módulos.
- Realización de las pruebas de sistema a las funcionalidades implementadas.

Estructura del Documento:

El presente trabajo está estructurado por capítulos. En el **Capítulo 1** se expone el estado del arte, donde se realiza la fundamentación teórica del tema. Se mencionan y describen algunos sistemas existentes que gestionan el intercambio de mensajes SWIFT. Además se describen las tecnologías, metodologías y herramientas de desarrollo. En el **Capítulo 2** se describe el análisis y el diseño de los módulos, así como los patrones utilizados para dar respuesta a la solución planteada. Se realiza la estructuración de los módulos, evidenciando los diagramas de clases del diseño, los de secuencia, paquetes y la descripción de las principales clases. Por último el **Capítulo 3** se enfoca en la construcción de la solución explicando los aspectos principales de la implementación. Se realiza la descripción de las clases y funcionalidades de los módulos Configuración de las reglas semánticas y Administración de las asociaciones de las operaciones de negocio que conforman y reciben mensajes SWIFT, quedando establecida la validación de la solución propuesta y la implementación del producto.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

1.1 Introducción

En el presente capítulo se describen los fundamentos teóricos que sustentan la investigación en cuestión. Se analizan los sistemas existentes tanto nacionales como internacionales que gestionan los procesos de intercambio de mensajes SWIFT, así como se muestra la descripción de las tecnologías y metodologías que se utilizan durante el desarrollo del producto.

1.2 SWIFT

SWIFT es una organización especializada para facilitar el intercambio de información financiera entre bancos y entidades financieras. Se creó en Bruselas en 1973 y su desarrollo en el ámbito del intercambio de información financiero ha sido vertiginoso. Este actúa únicamente como 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. (2)

1.3 Norma ISO 15022

En los años 80 la empresa SWIFT creó la primera norma ISO 7775 basada en los mensajes SWIFT creados con anterioridad. Luego creó la norma ISO 15022 como una evolución de la anterior y la última es la ISO 20022, la cual se está adoptando internacionalmente. La estructura de los mensajes de esta última norma está sobre el formato XML, no así las dos normas anteriores que están en texto plano. Específicamente el BNC se rige para realizar el intercambio de mensajes SWIFT por la norma ISO 15022. (2)

La innovación fundamental de la ISO 15022 fue el enfoque orientado a negocio que le proporcionaron a los mensajes. Sus principales características son:

- Las piezas de información de negocio están definida sin ambigüedades, utilizando un diccionario de datos. Los elementos del negocio son reutilizados por diferentes mensajes de negocio.

- Los mensajes de negocio son definidos en un catálogo de mensajes. Esto agiliza el trabajo de mantenerlos estandarizados con la evolución del negocio.

Los mensajes bajo la norma ISO 15022, se identifican por la palabra MT más un número de tres dígitos. El primer dígito responde a la categoría del mensaje, el segundo al grupo dentro de la categoría y el tercero al número del mensaje dentro del grupo. Existen un total de 10 categorías, desde la categoría 1 hasta la categoría 9 y una categoría n. Cada una de las categorías responde a operaciones financieras menos la última que es de propósito general. (2)

Los mensajes MT tienen una estructura sintáctica y opcionalmente reglas semánticas. La estructura sintáctica del mensaje define los componentes que debe tener y su orden. Las reglas son las condiciones que deben cumplir los datos contenidos en los mensajes. Los mensajes están compuestos internamente por bloques de contenido. Los tres primeros bloques contienen información de referencia para el mensaje, el cuarto bloque representa el texto del mensaje y el quinto se utiliza para guardar información de seguridad. (2)

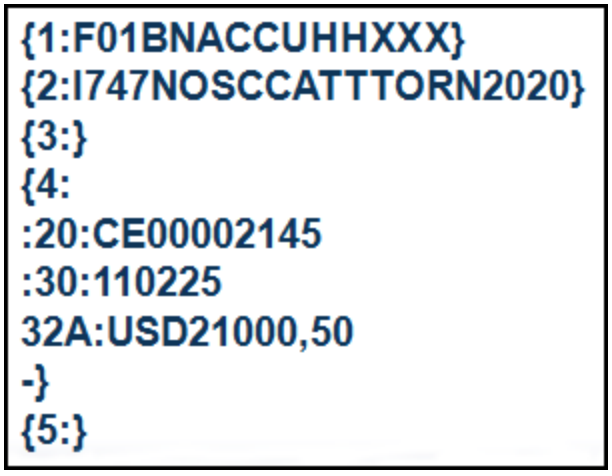


Fig. 1.1 Ejemplo de un mensaje SWIFT de la norma ISO 15022

1.4 Reglas semánticas

Las reglas semánticas son condiciones que debe cumplir el contenido de los mensajes. Un mensaje puede tener asociado más de una regla semántica y una misma regla puede ser aplicada a varios mensajes. Para representar las reglas, SWIFT definió una notación, la cual plantea que una regla está

constituida por uno o varios “Construct”. Existen dos tipos de “Construct”, el primero se asemeja a las tradicionales bifurcaciones de la mayoría de los lenguajes de programación. Su estructura es la siguiente:

IF <statement>

THEN <procedure>|<construct>

ELSE <procedure>|<construct>

ENDIF

El segundo Construct tiene la forma de un ciclo repetitivo como la mayoría de los lenguajes de programación:

FOR <statement>

<procedure>|<construct>

ENDFOR

Un “statement” es una condición que debe evaluarse. Existen varias formas de establecer una condición, está la que verifica si existe o no un componente dentro del mensaje, está la que evalúa si el valor de un componente del mensaje es igual, mayor o menor que un valor de otro componente, y así otras tantas. Por su parte un “procedure” puede ser de dos formas diferentes. Uno representa la ejecución satisfactoria de la validación semántica y la otra, una ejecución no satisfactoria. (2)

1.5 Ficheros de transformación

Un mismo mensaje puede ser enviado en más de una operación contable, por lo que se hace necesario contar con un mecanismo que se encargue de la transformación de los datos introducidos en la operación realizada al formato del mensaje definido por SWIFT.

Los datos introducidos en el sistema son recogidos en un XML y mediante ficheros de transformación son llevados a la estructura de los mensajes que se quieren enviar. Esta transformación se lleva a cabo gracias a las facilidades que brindan los XSLT (E**x**tensible **S**tylesheet **L**anguage **T**ransformation).

XSLT es un subconjunto de otro lenguaje basado en XML: XSL (Extensible Stylesheet Language). XSL se utiliza para definir el formato de documentos XML, y XSLT contiene plantillas y comandos para seleccionar y manipular la estructura de los datos. (3)

Para que el subsistema de mensajería sea actualizable y adaptable a las necesidades del BNC, debe contar con la funcionalidad de añadir nuevas operaciones contables y poder asociarlas con los mensajes que se necesiten enviar o recibir según sea el caso. Para esto los ficheros de transformación son escritos en lenguaje XSL, y este lenguaje provee un mecanismo para extraer los datos del negocio y conformar los mensajes deseados.

1.6 Sistema nacional para la comunicación financiera (SISCOM)

El SISCOM es un producto desarrollado en Cuba para el procesamiento y enrutamiento de mensajería SWIFT a través de la red SWIFTNet FIN, utiliza como interfaz los servicios de un servidor SWIFT Alliance Access (SAA) y sus módulos están implementados en FoxPro y C. (4)

Los módulos del sistema son:

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

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

SwAdmin: Funciones administrativas.

SwMPServer: Módulo que interactúa directamente con el AA.

Específicamente el módulo SACIM es el encargado de aplicar las validaciones semánticas definidas por SWIFT. Para hacer frente a las modificaciones que ocurren anualmente en la norma ISO 15022 del estándar SWIFT, su implementación debe ser modificada, debido que no cuenta con ficheros que guarden las expresiones lógicas que representan a cada regla semántica, ni la asociación de éstas con los mensajes. Se puede llegar a la conclusión que SISCOM no es un sistema flexible ya que los cambios que se necesiten realizar a las reglas semánticas no son soportados desde la aplicación.

Es justo señalar que la aplicación funciona correctamente y permite la transmisión de los mensajes SWIFT definidos en su base de datos. Sin embargo, analizando el diseño de su base de datos se percibe que no se pueden guardar todos los formatos de los mensajes definidos en el estándar y en la información que

guarda existe redundancia. Además es importante señalar que tampoco cuenta con ninguna herramienta que permita la asociación de una operación del sistema de gestión contable con uno o varios mensajes. Esto es un punto en contra ya que SWIFT agrupa sus mensajes de acuerdo a su propósito y todos excepto los informativos pueden indicar la ejecución de una operación financiera. (1)

1.7 Sistemas internacionales para la comunicación financiera basados en el estándar SWIFT

Muchas empresas productoras de software a nivel mundial se han introducido en el desarrollo de sistemas para el intercambio de información financiera, entre ellas se encuentran empresas como Oracle, SAP y SWIFT; siendo esta última la más destacada en el desarrollo de sistemas basados en el estándar SWIFT.

1.7.1 B2B Data Transformation

B2B Data Transformation es 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 automatizada de datos SWIFT. Ofrece una transformación integral de datos estructurados y no estructurados. Ofrece un desarrollo rápido por medio de una biblioteca de transformaciones predefinidas que cubren más de 100 tipos de mensajes SWIFT. (5)

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. (6)

1.7.2 Incentage Middleware Suite (IMS)

Incentage Middleware Suite es un producto desarrollado en Java, gracias a esto se puede ejecutar en Windows, Solaris, HP-UX, AIX y Linux. 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. (7)

A IMS le fue reconocido con la certificación SWIFTReady Gold Financial EAI[2] 2006 confirmando la correspondencia del producto con los más altos estándares de SWIFT en la línea de productos EAI. (7)

IMS abarca los más de 250 tipos de mensajes definidos en el paquete SWIFT FIN e incluyó un grupo de herramientas en el IMS a fin de asegurar que los estándares de mensajería utilizados estén alineados con las versiones de software liberados de SWIFT. (7)

1.7.3 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[3], además de reducir el tiempo y los gastos relacionados con la implementación de las soluciones empresariales a través de SWIFTNet. (8)

Alliance Integrator incluye una completa biblioteca de adaptadores técnicos que facilitan la conexión con las aplicaciones de su empresa. Acepta casi cualquier tipo de registro o formato de archivo de las aplicaciones empresariales, por ejemplo, valores separados por comas (CSV), cuadernos COBOL, mensajes XML o mensajes SWIFT. (8)

Desde el punto de vista de los formatos de datos estándar, Integrator incluye una biblioteca con todos los esquemas estándar de MT, MX y FpML, además de que también puede incorporar fácilmente otras definiciones de mensajes XML. (8)

Los sistemas antes expuestos constituyen buenas soluciones informáticas para la comunicación financiera a nivel mundial basada en el estándar SWIFT. Los mismos cuentan con gran cantidad de funcionalidades, gestionan la información con calidad y tienen bien presente la seguridad de la misma. Pero, es necesario aclarar que estos sistemas son privativos, no existe documentación de cómo manejan la gestión de las reglas semánticas y no se tiene acceso a su código fuente por lo que es imposible determinar si se puede integrar alguno con el subsistema de mensajería de Quarxo.

[2] ***E**nterprise **A**pplication **I**ntegration (Integración de aplicaciones Empresariales)*

[3] **Back-office:** *Tareas destinadas a gestionar la propia empresa*

1.8 Metodología, lenguaje de modelado y herramientas utilizadas

1.8.1 Metodologías de desarrollo de software

Las metodologías de desarrollo de software se dividen en dos grandes grupos: metodologías tradicionales o pesadas, las cuales se centran en el control del proceso, estableciendo rigurosamente las actividades involucradas, los artefactos que se deben producir y las herramientas y notaciones que se usarán; y las metodologías ágiles o ligeras, las cuales son efectivas en proyectos con requisitos muy cambiantes y en donde se exige reducir drásticamente los tiempos de desarrollo manteniendo una alta calidad de los mismos. Algunas de las metodologías pesadas más empleadas en el mundo de la producción de software son: **R**ational **U**nified **P**rocess (**RUP**, por sus siglas en Inglés) y Microsoft Solution Framework (MSF). Por su parte la metodología ágil más utilizada es: eXtreme Programming (XP). (9)

Teniendo en cuenta que el sistema Quarxo tiene una complejidad alta, que el equipo de trabajo supera a las 60 personas y que no cuenta con una amplia experiencia, es conveniente garantizar la calidad del proceso desde el principio y ganar en organización, por lo que se decidió por parte de los arquitectos adoptar una metodología tradicional.

Entre las metodologías pesadas se escogió RUP, la cual es una de las más utilizadas actualmente en el mundo para el desarrollo de software, la misma define claramente quién, cómo, cuándo y qué debe hacerse en el proyecto, con 3 características esenciales: centrado en la arquitectura, iterativo e incremental y dirigido por casos de uso. RUP toma en cuenta las mejores prácticas en el modelo de desarrollo de software. En particular, desarrollo de software en forma iterativa, manejo de requerimientos, utiliza arquitectura basada en componentes, modela el software visualmente (modela con UML[4]), verifica la calidad del software y controla los cambios. (9)

Por otra parte MSF es otra metodología tradicional pero está basado en desarrollo con tecnología Microsoft lo que limita las opciones del cliente en lo que se refiere a herramientas de desarrollo. Esta metodología hace un análisis de riesgo demasiado exhaustivo que puede frenar el avance del proyecto y además solicita demasiada documentación en todas las fases resultando muy engorroso el trabajo y por consiguiente afecta el tiempo de entrega del producto. (10)

[4] ***U**nified **M**odeling **L**anguage (Lenguaje Unificado de Modelado)*

1.8.2 Lenguaje y herramientas para el modelado

El lenguaje propuesto por la metodología de desarrollo RUP para modelar elementos de software es UML, el cual es un sistema de notación que se ha convertido en estándar en el mundo del desarrollo de sistemas. El mismo es un lenguaje de modelado visual que se usa para especificar, visualizar, construir y documentar artefactos de un sistema de software. Pretende unificar la experiencia pasada sobre técnicas de modelado e incorporar las mejores prácticas actuales en un acercamiento estándar. Incluye conceptos semánticos, notación, y principios generales. Tiene partes estáticas, dinámicas, de entorno y organizativas.

Existen varios software para el modelado UML, son las denominadas herramientas CASE[5]. La herramienta CASE es la aplicación de tecnología informática a las actividades, las técnicas y las metodologías propias de desarrollo, su objetivo es acelerar el proceso para el que han sido diseñadas, automatizar o apoyar una o más fases del ciclo de vida del desarrollo de software. La principal ventaja de la utilización de estas herramientas es la mejora de la calidad de los desarrollos realizados y el aumento de la productividad. Entre las herramientas CASE para el modelado están Enterprise Architect, Visual Paradigm y Rational Rose.

Visual Paradigm es una herramienta de diseño que soporta todos los diagramas UML y diagrama entidad-relación. Visual Paradigm para UML ofrece amplias características de modelado, diagrama de casos de uso, editor de flujos de evento, caso de uso/red de actor y la generación de diagramas de actividades. Los analistas del sistema pueden estimar las consecuencias de los cambios en los diagramas de análisis de impacto, tales como la matriz y el diagrama de análisis. Esta herramienta permite además la ingeniería inversa de código Java, .NET, PHP5, Python, C. Características:

- Modelado UML
- Modelado de Casos de Uso
- Generación de reportes
- Análisis de Impacto

[5] ***C**omputer **A**ided **S**oftware **E**ngineering (Ingeniería de Software Asistida por Ordenador)*

Rational Rose es la herramienta CASE que comercializan los desarrolladores de UML (Booch, Rumbaugh y Jacobson) a través de IBM y soporta de forma completa la especificación del UML. Es una herramienta que propone la utilización de cuatro tipos de modelos para realizar un diseño del sistema, utilizando una vista estática y otra dinámica de los modelos del sistema, uno lógico y otro físico. Permite crear y refinar vistas creando un modelo completo que representa el dominio del problema y el sistema de software. Dentro de sus características está el desarrollo iterativo pues utiliza un proceso de desarrollo iterativo controlado, donde el desarrollo se lleva a cabo en una secuencia de iteraciones. Genera código en distintos lenguajes de programación a partir de un diseño en UML y proporciona mecanismos para realizar ingeniería inversa.

Se escoge como herramienta de modelado el Visual Paradigm para UML 6.4 debido fundamentalmente a que es multiplataforma (incluyendo Linux) y soporta modelado tanto de todas las fases como del modelo de datos. Soporta además ingeniería directa e inversa y todos los diagramas UML, entidad-relación y el ciclo de vida del desarrollo de software completo: análisis, diseño, implementación, pruebas y despliegue.

1.8.3 IDE[6] de desarrollo (Eclipse Galileo)

Entre los principales IDE's de desarrollo para el lenguaje Java se encuentran Eclipse, Netbeans, e IntelliJ Idea. (11)

Eclipse es un IDE gratuito que soporta varios lenguajes de programación adicionados a él mediante plugins[7] entre los que se encuentra Java. Es considerado por muchos como la mejor herramienta disponible de desarrollo con Java. Ofrece una excelente edición Java con validación, compilación incremental, referencias cruzadas, asistente de código, un editor de XML y mucho más. (11)

IntelliJ IDEA es un IDE centrado en la productividad del código. El editor entiende profundamente el código, hace sugerencias muy adecuadas y ayuda a dar forma a su código. Soporta Java, Groovy, XML con asistencia de código y Subversion. Entre sus principales características se encuentra la asistencia inteligente de codificación, la generación de código, el estilo de código, la documentación que posee, el

[6] ***IDE: I**ntegrated **D**evelopment **E**nvironment (Entorno de desarrollo Integrado)*

[7] ***Plugin:** módulo de hardware o software que añade una característica o un servicio específico a un sistema más grande.*

análisis de código sobre la marcha lo cual usa un método intuitivo para ayudar a escribir el código. Posee una amplia gama de lenguajes de programación permitidos. Trae integrado varias tecnologías y framework como: Java Server Pages (JSP), Java Server Faces (JSF), Asynchronous JavaScript and XML (AJAX), Spring, Hibernate, Servicios WEB, entre otras. Una característica de importancia que se debe conocer, es que es una aplicación propietaria y por tanto se debe pagar por su licencia o uso. (11)

Netbeans es un IDE de código abierto cuya principal misión es hacer fácil el desarrollo de todo tipo de aplicaciones en la plataforma JAVA EE. Con Netbeans es posible desarrollar tanto aplicaciones de escritorio como aplicaciones empresariales en varias capas. Cuenta además con una arquitectura modular que permite que con posterioridad se le añadan complementos (plugins).

Todas estas herramientas son muy potentes y facilitan en gran medida el desarrollo con el lenguaje Java. La utilización de IntelliJ Idea fue descartada desde el inicio por su condición de software privativo. Entre el Netbeans y el Eclipse se analizó la experiencia de los desarrolladores con los mismos, comprobándose que en su mayoría usaban Eclipse debido a las prestaciones de las estaciones de trabajo y al bajo consumo de recursos que evidencia este frente al Netbeans.

1.8.4 Contenedor Web (Apache Tomcat 6.0.26)

Como soporte para el funcionamiento de las aplicaciones desarrolladas en Java existen varios entornos de ejecución como Jboss Server, GlassFish y Apache Tomcat, este último no es considerado como un servidor de aplicaciones, sino como un contenedor de servlets, pero es capaz de soportar la mayoría de las aplicaciones desarrolladas en dicha tecnología. (12)

JBoss Application Server (JBoss AS) es el servidor de aplicaciones de código abierto más ampliamente desarrollado del mercado actualmente. Combinando una arquitectura orientada a servicios revolucionaria con una licencia de código abierto, JBoss AS puede ser descargado, utilizado, incrustado, y distribuido sin restricciones por la licencia. (13)

GlassFish es un servidor de aplicaciones que implementa la plataforma JavaEE, soporta las últimas versiones de tecnologías como: JSP, JSF, servlets, arquitectura Java para Enlaces XML y muchas otras tecnologías. (14)

El Apache Tomcat es un software de código abierto implementado para las tecnologías Java Servlet y JSP. Es desarrollado en un entorno abierto, participativo y publicado bajo la licencia del software de Apache.

El mismo se ejecuta en varios sistemas operativos. Es un servidor configurable de diseño modular, con diversidad que permiten garantizar una elevada seguridad y buenas prestaciones. Además, brinda soporte para la plataforma de desarrollo JEE. Existe abundante documentación asociada al mismo, cuenta con una gran comunidad de desarrollo y es uno de los más usados internacionalmente. (15)

Teniendo en cuenta las características expuestas, se selecciona Apache Tomcat como servidor de aplicaciones. Debido a que todos los servidores analizados están muy parejos en lo referente a sus funcionalidades, se utilizó la experiencia del equipo de desarrollo y los bajos recursos que este demanda como factores fundamentales en la toma de esta decisión.

1.8.5 Control de Versiones (SubVersion 1.6.6)

Para el control de versiones existen distintos software que se encargan de realizar estas actividades, entre los más usados internacionalmente se encuentran: Subversion, CVS[8] y Git.

Subversion: es un sistema de control de versiones libre, de código fuente abierto. Maneja ficheros y directorios a través del tiempo. El repositorio es como un servidor de ficheros ordinario, excepto que recuerda todos los cambios hechos a sus ficheros y directorios. Esto le permite recuperar versiones antiguas de sus datos, o examinar la historia de cómo han cambiado. La característica de Subversion de que puede acceder a su repositorio a través de redes, permite que varias personas puedan modificar y administrar el mismo sistema de datos desde sus respectivas ubicaciones.

CVS: es un sistema para el control de versiones que se ha hecho muy popular en el mundo del software libre pues es muy difundido por sus desarrolladores bajo la licencia GPL. Es empleado en proyectos comerciales y prácticamente en todos los proyectos de código abierto.

Es un software de código abierto que permite gestionar los cambios realizados sobre códigos fuentes de cualquier archivo en varias plataformas (C, C++, Java, etc.). Pero tiene como desventajas que no ofrece

[8] **CVS: *C**oncurrent **V**ersions **S**ystem (Sistema de versiones concurrentes)*

facilidades para hacer referencia a cambios en múltiples ficheros y no permite renombrar o copiar ficheros dentro del sistema de control por lo que puede ser muy doloroso reorganizar el código después de iniciar el proyecto. (16)

Git: es un software para el control de versiones que se caracteriza por ser un sistema rápido y eficaz, fue desarrollado por el creador del núcleo de Linux, Linus Torvalds. Git es un tipo muy diferente de control de versiones ya que se trata de un sistema de control de versiones distribuidas, es decir no hay una base de código centralizada para extraer el código. Diferentes ramas ocupan diferentes partes del código. Esto lo hace diferente de otros sistemas de control de versiones, como CVS y SVN que usan un control centralizado de versiones, lo que significa que sólo muestran una copia del software que se utiliza. (17)

Según las características particulares de cada herramienta se seleccionó Subversion para el control de versiones debido a que era la herramienta más sólida que existía en el momento de la selección, además se contaba con el apoyo de personal con conocimiento y experiencia en la administración de dicha herramienta.

1.8.6 Gestor de Bases de Datos (SQL Server 2005)

SQL Server 2005 provee herramientas sólidas que reducen la complejidad de creación, despliegue, administración y uso de aplicaciones analíticas en plataformas que van desde los dispositivos móviles hasta los sistemas de datos empresariales. Permite trabajar en modo cliente-servidor, donde la información y los 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. (18) Es importante aclarar que aunque este gestor de Base de datos es un software privativo su selección fue impuesta por el cliente, debido a que para acelerar el desarrollo se utilizará el núcleo del SABIC el cual incluye un grupo de tablas y procedimientos almacenados que ayudarán en la implementación del sistema.

1.9 Ambiente de Desarrollo

1.9.1 Java Platform, Enterprise Edition

Java Platform, Enterprise Edition o Java EE es una plataforma de programación para desarrollar y ejecutar software de aplicaciones en el lenguaje de programación Java, se basa en componentes de software modulares ejecutándose sobre un servidor de aplicaciones. Java EE incluye varias especificaciones de API, tales como JDBC, JAXB, Servicios Web, XML, etc. Entre los beneficios que aporta el uso de esta plataforma se encuentran que el servidor de aplicaciones puede manejar transacciones, la seguridad, escalabilidad, concurrencia y la gestión de los componentes desplegados.

1.9.2 Lenguajes de programación

Entre los lenguajes de programación del lado del servidor más usados en el desarrollo web se encuentra PHP, JAVA y C#. PHP es un lenguaje gratuito e independiente de plataforma, rápido, con una gran librería de funciones y mucha documentación. (19) En cuanto a C# es un lenguaje simple, potente, seguro y orientado a objetos, es desarrollado y estandarizado por Microsoft como parte de la plataforma .NET; (20) la principal desventaja de este lenguaje es que es dependiente de la plataforma a menos que se utilice MonoDevelop[9] para poder hacer de C# un lenguaje multiplataforma.

Por su parte el lenguaje JAVA es fácil de usar, multiplataforma, sumamente flexible, permite la creación de programas modulares y de códigos reutilizables. Dicho lenguaje además es compilado, utiliza el paradigma orientado a objeto y en la actualidad es muy utilizado, ya que permite el desarrollo de programas seguros y de alto rendimiento. Es uno de los lenguajes más utilizado en la actualidad para el desarrollo de proyectos de software bajo la especificación de Java Enterprise Edition (JEE), el cual es una tecnología que apunta a simplificar el diseño y desarrollo de aplicaciones empresariales robustas, escalables y seguras utilizando Java como lenguaje. Simplifica las aplicaciones empresariales basándolas en componentes modulares y estandarizados, ofreciendo un completo conjunto de servicios a estos componentes, y manejando muchas de las funciones de la aplicación de forma automática, sin necesidad de una programación compleja.

Se seleccionó Java por el equipo de arquitectura del proyecto SAGEB (Sistema automatizado de gestión bancaria) primeramente por todas las características que presenta el lenguaje, atendiendo al conocimiento y preparación del equipo de desarrollo con que se contaba al iniciar el proyecto, basándose además en los

[9] **MonoDevelop:** *es un entorno de desarrollo integrado, libre y gratuito, diseñado primordialmente para poder utilizar C# en distintas distribuciones de Linux y Mac.*

resultados obtenidos por el proyecto SIGEP (Sistema de Gestión Penitenciaria), el cual usó este lenguaje y desarrolló varias librerías para el mismo con el objetivo de llevar a cabo un desarrollo mucho más rápido logrando montar una arquitectura robusta que se adaptaba a las necesidades del sistema Quarxo.

1.10 Tecnologías y Frameworks

1.10.1 Spring Framework

Del lado del servidor existen muchas tecnologías que facilitan el desarrollo de aplicaciones, entre las más populares se encuentran: Struts, JSF y Spring. (21)

El framework Struts brinda soporte para el desarrollo de aplicaciones web bajo el patrón MVC para la plataforma JEE. Struts se desarrollaba como parte del proyecto Jakarta de la Apache Software Foundation, pero actualmente es un proyecto independiente conocido como Apache Struts.

Struts permite reducir el tiempo de desarrollo. Su carácter de "software libre" y su compatibilidad con todas las plataformas en que Java Entreprise esté disponible, lo convierte en una herramienta altamente disponible.

Cuando se programan aplicaciones Web con el patrón MVC, siempre surge la duda de usar un solo controlador o varios controladores, pues si se considera mejor usar un solo controlador para tener toda la lógica en un mismo lugar, surge un inconveniente, ya que el controlador se convierte en lo que se conoce como "fat controller", es decir un controlador de peticiones, Struts surge como la solución a este problema, ya que implementa un solo controlador (ActionServlet) que evalúa las peticiones del usuario mediante un archivo configurable (struts-config.xml).

El framework JSF es un estándar para aplicaciones web basadas en Java, facilita la construcción de aplicaciones siguiendo el patrón MVC, modelo de componentes orientado a objetos, conversión de tipos, separación de responsabilidades, desarrolladores de componentes, desarrolladores de lógica de aplicación, montadores de páginas, poderoso sistema de navegación declarativa, uso de simples clases java como controladores, fácil incorporación de potencialidades AJAX, posee un conjunto prefabricado de componentes de interfaz de usuario y modelo de programación orientado a eventos. (22)

El framework Spring MVC, también conocido como Spring es un framework de código abierto de desarrollo de aplicaciones para la plataforma Java, su principal característica es brindar una fábrica de objetos basado en la inyección de dependencia. Por su diseño, el framework ofrece mucha libertad a los desarrolladores en Java, soluciones muy bien documentadas y fáciles de usar para las prácticas comunes en la industria.

Se basa en una configuración a base de *javabeans* bastante simple. Es potente en cuanto a la gestión del ciclo de vida de los componentes y fácilmente ampliable. Spring ofrece una manera simple de implementar DAOs basados en Hibernate sin necesidad de manejar instancias de sesión de Hibernate o participar en transacciones.

Entre las tecnologías anteriormente descritas el equipo de arquitectura del proyecto SAGEB seleccionó Spring MVC como herramienta de desarrollo debido a que este es ideal para aplicaciones de gran tamaño y posee una comunidad de gran prestigio a nivel internacional (http://www.springsource.org/) que facilita una buena retroalimentación. Otra razón considerada fue su uso en la UCI en proyectos exitosos, un ejemplo de ello es el proyecto SIGEP del cual se tomó la arquitectura base del Quarxo.

1.10.2 Hibernate

Los frameworks ORM[10] se utilizan para el trabajo con el acceso a datos de una aplicación, entre ellos se encuentran IBATIS, Spring JDBC e Hibernate que son soportados por la plataforma JEE.

IBATIS toma los mejores atributos e ideas de las tecnologías ORM y Data Mapper (Mapeador de datos) encontrando un equilibrio entre ellas, convirtiéndose así en una solución híbrida. Algunas de las ideas que toma son: el soporte para procedimientos almacenados, SQL dinámico, SQL en línea y el Mapeo Objeto Relacional.

Spring JDBC es un módulo perteneciente al framework Spring. El mismo posee algunas librerías de clases para trabajar con base de datos a través del API de Java JDBC. Unas de las características principales de este módulo es el soporte que brinda para invocar procedimientos y funciones almacenadas.

[10] **ORM:** ***O****bject* ***R****elational* ***M****apping(Mapeo objeto-relacional)*

Hibernate es un framework de persistencia de objetos para Java. Su función principal es el mapeo Objeto-Relacional, es decir, mapear objetos a tablas de una base de datos relacional. Además posibilita la persistencia de objetos Java y al mismo tiempo la consulta de las bases de datos para obtener dichos objetos.

Entre los beneficios que aporta está la independencia de la base de datos y el bajo acoplamiento que ofrece entre el negocio y la persistencia. Además provee un lenguaje de consulta de datos llamado HQL[11] el cual es un lenguaje de consulta similar al SQL.

Fue seleccionado Hibernate principalmente por la sólida integración con el framework Spring facilitando el trabajo ya que este libera en gran medida al programador de mantener el control sobre las sesiones. Una de las principales clases que nos brinda el framework para su integración es “HibernateTemplate” la cual brinda un gran número de métodos para el trabajo con los datos.

1.10.3 Dojo Toolkit

En el año 2007 los framework Javascript mas usados eran Prototype y Dojo. (23)

Prototype fue creado por Sam Stephenson para el desarrollo sencillo y dinámico de páginas Web. Es un framework escrito en JavaScript que implementa las técnicas AJAX y su potencial es aprovechado al máximo cuando se desarrolla con Ruby On Rails. Prototype simplifica parte del trabajo cuando se pretende desarrollar páginas altamente interactivas. (24)

Dojo Toolkit es un framework Javascript que permite el desarrollo de aplicaciones web enriquecidas en el cliente. Es popular porque está integrado en numerosos IDEs y otros frameworks para desarrollo web. Contiene un sistema de empaquetado inteligente, agradables efectos de la interfaz de usuario, y numerosos APIs[12] como: Drag and Drop (arrastrar y soltar), Widget[13], almacenamiento en el cliente, e interacción con AJAX. (25)

---

[11] ***HQL: H**ibernate **Q**uery **L**anguage (Lenguajes de consultas de Hibernate)*

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

[13] **Widget:** es una pequeña aplicación o programa presentado usualmente en archivos o ficheros pequeños.

Esta biblioteca es de código abierto y se puede descargar de forma gratuita en su página oficial. La licencia permite crear aplicaciones, utilizarlo en productos comerciales y modificarlo. Cuenta con el patrocinio de IBM, Google, AOL y Nexaweb. Estas son algunas razones por las que este framework está cubierto por una gran comunidad, con multitud de desarrolladores e información que lo hacen muy accesible y transparente de cara a nuevos usuarios. (25)

Tanto Prototype como Dojo brindan funcionalidades similares pero se seleccionó Dojo debido a la fácil integración con Spring MVC. Además se tuvo en cuenta que el proyecto SIGEP desarrolló una gran cantidad de componentes y funciones las cuales se podían reutilizar en el desarrollo del sistema Quarxo.

1.11 Patrones

1.11.1 Patrones de arquitectura

Los patrones de arquitectura definen los aspectos fundamentales de la estructura de un sistema de software. Especifican un conjunto predefinido de subsistemas con responsabilidades y una serie de recomendaciones para organizar los distintos componentes. (26)

Patrón Modelo vista controlador (MVC)

El patrón MVC separa el modelo del dominio del negocio (entidades de negocio), presentación y las acciones que se ejecutan sobre éstas.

El **modelo** administra el comportamiento y la información del modelo del negocio, de la misma forma que responde sobre acciones que se quieran efectuar sobre éste, la **vista**, es la encargada de administrar la presentación de la información, mientras que el **controlador** es el componente intermedio entre el modelo y la vista para lograr un desacoplamiento total entre estos dos. El controlador es como el “mensajero” entre la parte visual y la información a ser manipulada. (27)

1.11.2 Patrones de diseño

Los patrones de diseño son soluciones simples y elegantes a problemas específicos y comunes del diseño orientado a objetos. Son soluciones basadas en la experiencia y que se ha demostrado que funcionan. (28)

1.11.2.1 Patrones de asignación de responsabilidades (GRASP)

Los patrones **GRASP** acrónimo de **G**eneral **R**esponsability **A**ssignment **S**oftware Patterns (Patrones de Software para la asignación General de Responsabilidad) describen los principios fundamentales de diseño de objetos para la asignación de responsabilidades. Los siguientes patrones pertenecen a dicho grupo:

Patrón Bajo acoplamiento

Este patrón consiste en tener las clases lo más independientes entre sí posible y con la menor cantidad de relaciones; lo que posibilita en caso de producirse una modificación en alguna de ellas, se tenga la mínima repercusión posible en el resto de las clases, permitiendo la reutilización y disminuyendo la dependencia entre las clases, asignándoles una responsabilidad para mantener un bajo acoplamiento. (29)

Ventajas:

- No se afectan por cambios de otros componentes.
- Fáciles de entender por separado.
- Fáciles 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. (29)

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 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 o implementarlo. De este modo se obtendrá un diseño con mayor cohesión y así la información se mantendrá encapsulada. (29)

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 y los envía a las distintas clases según el método llamado. (29)

Ventajas:

- Mayor potencial de los componentes reutilizables.

1.11.2.2 Patrones GoF

Por sus siglas en inglés (Gang of Four). Llamados así debido a que fueron 4 autores los que escribieron el libro “Design Patterns” que ilustra sus funciones.

Fachada

El patrón fachada trata de simplificar la interface entre dos sistemas o componentes de software ocultando un sistema complejo detrás de una clase que actúa de fachada. La idea principal es la de ocultar todo lo posible la complejidad de un sistema, el conjunto de clases o componentes que lo conforman, de forma tal que solo se ofrezca un (o unos pocos) punto de entrada al sistema tapado por la fachada. (30)

Patrón Cadena de responsabilidad

La cadena de responsabilidad se encarga de evitar el acoplamiento del remitente de una petición a su receptor, dando a más de un objeto la posibilidad de manejar la petición. Es la cadena de objetos que reciben y pasan la petición a lo largo de la cadena hasta que un objeto la manipule. (31)

Patrón Objeto de acceso a datos

El patrón Objeto de acceso a datos (DAO por sus siglas en inglés **D**ata **A**ccess **O**bject) resuelve el problema de contar con diversas fuentes de datos, además de que encapsula y oculta la forma de acceder a la base de datos, eliminando así complejidad del uso de JDBC a la capa de presentación o de negocio. Se trata de que el software cliente se centre en los datos que necesita y se olvide de cómo se realiza el acceso a los datos o de cuál es la fuente de almacenamiento. (32)

1.11.3 Patrones J2EE

Los patrones J2EE están orientados específicamente a los problemas comunes a todas las aplicaciones desarrolladas bajo la plataforma JEE. Algunos están basados en los patrones originales mientras que otros son más específicos del tipo de problemas que surgen específicamente en JEE, bien sea por los tipos de aplicaciones que se suelen desarrollar con la plataforma o por las características de la tecnología. (33)

Patrón Vistas Compuestas (Composite View)

Este patrón proporciona un mecanismo para combinar modularmente, las porciones atómicas de una vista completa, las páginas son construidas uniendo código dentro de cada vista. Por ejemplo las páginas JSP (Java Server Pages) y los servlets proporcionan mecanismos sencillos para poder incluir porciones de una página en otra (de modo estático o dinámico). Este esquema facilita el mantenimiento de las páginas. (24)

Conclusiones Parciales

En este capítulo se realizó un estudio de los sistemas existentes en la actualidad, tanto nacionales como internacionales, llegándose a la conclusión que se hace necesario el desarrollo de los módulos que realicen la gestión de las reglas semánticas y las asociaciones de las operaciones de negocio que conforman y reciben mensajes SWIFT. Ninguno de los sistemas analizados integra toda la información necesaria para dicha gestión, además de que los procesos que los mismos automatizan no se adaptan a

las necesidades propias del BNC. Dichos sistemas son propietarios y resulta poco factible para la economía de nuestro país, el costear alguno de ellos.

También se realizó un estudio de algunas de las principales metodologías, tecnologías y herramientas llegándose a conclusiones que coinciden con la máxima dirección del proyecto: con respecto a la metodología de desarrollo del software, se usará **RUP**, pues es la más adecuada dada la complejidad del sistema, con la utilización de dicha metodología se tendrá unificado todo el equipo de desarrollo de software, optimizando su comunicación y garantizando entendimiento entre sus integrantes, la misma modela al software visualmente utilizando **UML**, lenguaje de modelado más utilizado en la actualidad ya que permite toda la modelación de los diferentes artefactos generados en los distintos flujos de trabajo presente en el ciclo de vida del desarrollo exitoso de un software. Para el modelado se seleccionó el **Visual Paradigm en su versión 3.4**, por ser una herramienta multiplataforma que garantiza la calidad del software en todo el ciclo de vida y permite la comunicación entre los desarrolladores mediante un lenguaje común para todos los roles que intervienen en el proceso de desarrollo del software. Como lenguaje de programación se propone el uso de **JAVA** debido a su robustez, seguridad, portabilidad, dinamismo e independencia de la arquitectura, dicho lenguaje se utilizará bajo la especificación de **Java Enterprise Edition (JEE)**. Sobre esta plataforma se han desarrollado múltiples framework que facilitan la programación de aplicaciones, ya que encapsulan operaciones complejas en instrucciones sencillas, motivo por el cual se decidió utilizar el framework de aplicación **Spring** y el framework de soporte **Hibernate** los cuales son de código abierto al igual que **Dojo**, utilizado como framework **JavaScript** o de presentación para crear interfaces de manera fácil. Como herramientas de desarrollo se utilizará el **IDE Eclipse 3.3** con su sistema de plugins, herramienta altamente provechosa que admite la implementación con los frameworks antes mencionados. Como gestor de base de datos se utilizará el **SQL Server 2005**, por decisión del cliente; como contenedor web **Apache Tomcat 6.0.26**, el cual es un software de código abierto que brinda soporte para la plataforma de desarrollo JEE; y como controlador de versiones se utilizará **SubVersion**, que es un software libre y facilita las tareas administrativas. En conclusión, los módulos se desarrollarán con herramientas y tecnologías en su mayoría libres y multiplataforma, lo cual trae como consecuencia positiva una reducción notable del costo del sistema por concepto de licencias de software y soporte.

CAPÍTULO 2: ANÁLISIS Y DISEÑO DE LA PROPUESTA DE SOLUCIÓN

2.1 Introducción

En todo proceso de desarrollo de software, se hace necesario primeramente analizar los requisitos solicitados por el cliente, a fin de implementar un sistema acorde a sus necesidades. En el presente capítulo, se presenta un análisis de la ingeniería de requisitos desarrollada por el equipo de analistas que especificó los casos de uso por los que se rige la implementación de Quarxo. Se muestra el Modelo de Análisis y posteriormente el Modelo de Diseño, entre otros diagramas utilizados para modelar la estructura de la propuesta de solución.

2.2 Descripción del sistema

La ingeniería de requisitos ayuda a los ingenieros de software a entender mejor el problema en cuya solución trabajarán. Incluye el conjunto de tareas que conducen a comprender cuál será el impacto del software sobre el negocio, qué es lo que el cliente quiere y cómo interactuarán los usuarios finales con el producto final. (34)

Un requisito es una condición o capacidad que tiene que ser alcanzada por un sistema o componente de un sistema para satisfacer un contrato, estándar u otro documento impuesto formalmente. (35)

2.2.1 Requisitos Funcionales

Los requisitos funcionales (RF) definen el comportamiento del sistema y especifican la manera en que éste debe reaccionar en situaciones particulares. Además responden a las necesidades del cliente y son un punto de partida para especificar qué es lo que debe hacer el sistema a desarrollar. Los RF han sido agrupados aplicando el patrón especificar, con el propósito de facilitar la comprensión de la estructura definida. (36)

RF 1. Listar reglas semánticas: Permite mostrar un listado de las reglas semánticas asociadas al formato de un mensaje previamente seleccionado.

RF 2. Gestionar regla semántica.

RF 2.1. Adicionar nueva regla semántica: Se permite adicionar el fichero XML que contiene la nueva Regla semántica.

RF 2.2. Modificar regla semántica: Se permite cambiar el fichero XML de la regla semántica seleccionada.

RF 2.3. Consultar regla semántica: Muestra el texto de la regla semántica previamente seleccionada.

RF 2.4. Eliminar regla semántica: Se le permite al usuario eliminar la regla semántica seleccionada.

RF 3. Descargar fichero XML de la Regla semántica: Permite obtener en formato XML el fichero de la Regla semántica seleccionada.

RF 4. Descargar fichero XML del formato del mensaje SWIFT: Permite obtener el fichero XML del formato del mensaje SWIFT seleccionado.

RF 5. Listar las operaciones de negocio en las que se conforman mensajes SWIFT: Permite mostrar un listado de las operaciones que fueron previamente insertadas.

RF 6. Gestionar operación de negocio en la que se conforman mensajes SWIFT.

RF 6.1. Adicionar operación de negocio en la que se conforman mensajes SWIFT: Se permite adicionar una nueva operación de negocio en la que se conforman mensajes SWIFT.

RF 6.2. Actualizar operación de negocio en la que se conforman mensajes SWIFT: Se permite modificar los datos que el usuario desee de la operación seleccionada.

RF 6.3. Eliminar operación de negocio en la que se conforman mensajes SWIFT: El sistema permite eliminar la operación de negocio seleccionada.

RF 7. Listar las operaciones de negocio que se ejecutan cuando se reciben mensajes SWIFT: Permite mostrar un listado de las operaciones de negocio que fueron previamente insertadas.

RF 8. Gestionar operación de negocio que se ejecuta cuando se reciben mensajes SWIFT.

RF 8.1. Adicionar operación de negocio que se ejecuta cuando se reciben mensajes SWIFT: Se permite adicionar una nueva operación de negocio que se ejecuta cuando se reciben mensajes SWIFT.

RF 8.2. Actualizar operación de negocio que se ejecuta cuando se reciben mensajes SWIFT: Se permite modificar los datos que el usuario desee de la operación seleccionada.

RF 8.3. Eliminar operación de negocio que se ejecuta cuando se reciben mensajes SWIFT: El sistema permite eliminar la operación de negocio seleccionada.

2.2.2 Requisitos No Funcionales

Los requisitos no funcionales (RNF) no se refieren a funciones específicas que proporciona el sistema sino que son restricciones de los servicios en cuanto a la fiabilidad, tiempo de respuestas, capacidad de almacenamiento, etc. y generalmente estos se aplican al sistema en su totalidades decir, aquellas características que hacen al producto atractivo, usable, rápido y confiable. (36)

Los requisitos no funcionales más significativos son:

Funcionalidad

RNF 1. El sistema mostrará los errores en forma de mensajes.

- Todos los mensajes de error del sistema deberán incluir una descripción textual del error.

Usabilidad

RNF 2. Los formularios serán estandarizados, por tanto:

- Los campos de texto tendrán un tamaño estándar de acuerdo con el espacio que se tenga en el área de la página y en la medida que se llene esa área primaria agregar la barra de desplazamiento vertical.

- No se utilizarán textos extensos para las etiquetas de la interfaz de usuario.

RNF 3. El menú de navegación estará disponible en todas las páginas

RNF 4. En caso de que los resultados de las consultas tengan más de 10 coincidencias, estos se mostrarán de forma paginados en una tabla.

- Se mostrará en la parte inferior de la tabla el total de elementos encontrados, enlaces de navegación: ir hacia delante, hacia atrás o ir al inicio de los resultados mostrados.

Fiabilidad

RNF 5. El sistema estará disponible durante toda la jornada laboral del BNC.

Seguridad

RNF 6. El sistema permitirá la visualización de la información según el usuario autenticado.

RNF 7. El sistema implementará el uso de campos obligatorios y validaciones para garantizar la integridad de la información que se introduce por el usuario.

Restricciones de diseño

RNF 8. El sistema se implementará usando la plataforma J2EE.

RNF 9. El sistema estará basado en un estilo arquitectónico en capas.

Interfaz de Usuario

RNF 10. Todos los textos y mensajes en pantalla aparecerán en idioma español. Los errores serán visibles al usuario y en lo posible incluirán sugerencias de las posibles soluciones.

RNF 11. El sistema presentará los términos capitalizados, es decir, tendrán su primera letra en mayúsculas.

2.3 Diagrama de Casos de Uso del Sistema

Con los requisitos levantados se obtuvieron 12 casos de uso para los módulos Configuración de las reglas semánticas y Administración de las asociaciones de las operaciones de negocio que conforman y reciben mensajes SWIFT, los cuales son reflejados en el siguiente diagrama de casos de uso del sistema:

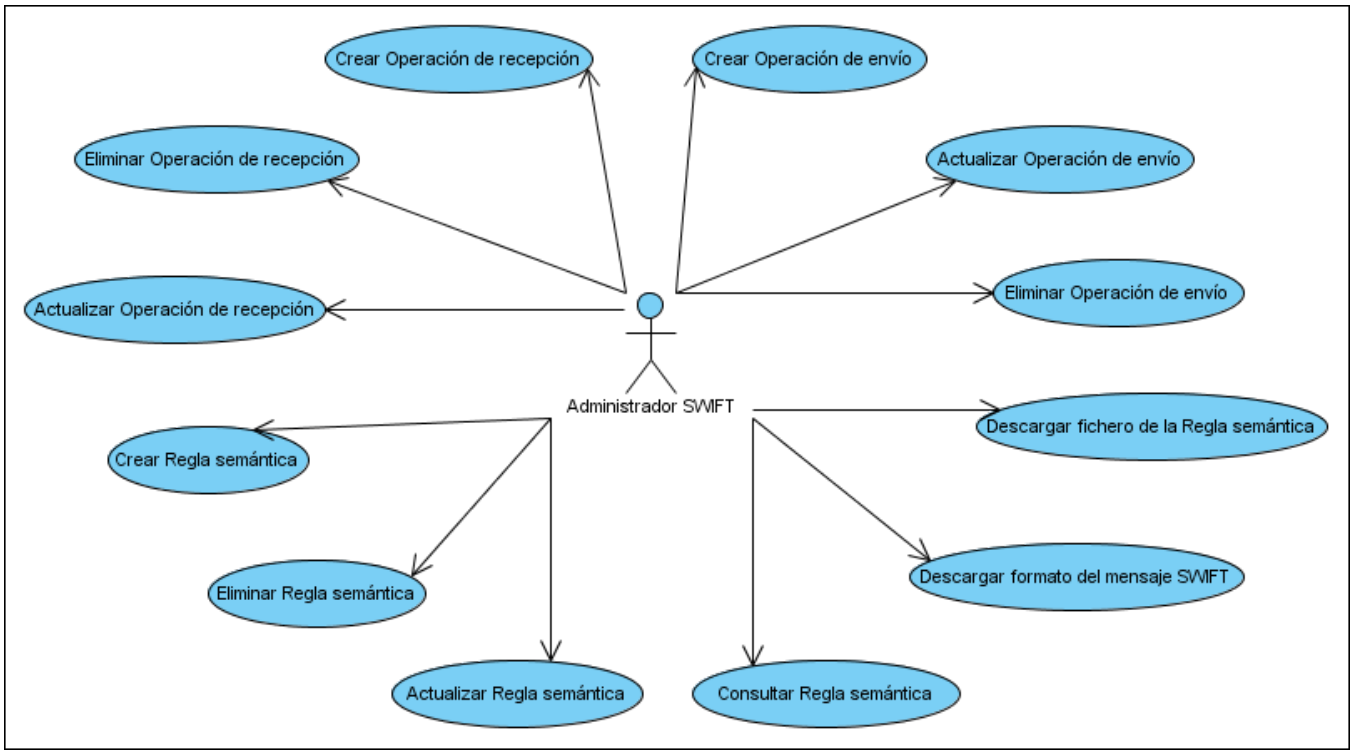


Fig. 2.1 Diagrama de Casos de uso del sistema

2.4 Especificaciones de Casos de Uso del Sistema

En el flujo de trabajo de requerimientos se definen cuestiones esenciales para el desarrollo de productos de software. Se recogen las solicitudes de los interesados y se transforman para obtener requisitos detallados sobre lo que el sistema debe hacer. Uno de los artefactos fundamentales que se generan en este flujo de trabajo son las especificaciones de casos de uso, a continuación se muestra la descripción del caso de uso configurar reglas semánticas, el resto de los procesos modelados se podrán consultar en el anexo 1.

Caso de Uso:	Configurar reglas semánticas.
Actores:	Administrador SWIFT.
Resumen:	Permite realizar acciones en el sistema sobre las reglas semánticas como: insertar, eliminar, modificar o consultar; además podrá descargar en formato XML el formato de los mensajes SWIFT, así como la definición de las reglas semánticas

	asociadas a los mensajes.
Precondiciones:	El administrador SWIFT debe estar autenticado en el sistema. Para consultar una regla semántica, debe estar seleccionada previamente. Para eliminar una regla semántica, debe estar seleccionada previamente. Para modificar una regla semántica, debe estar seleccionada previamente. Para descargar el fichero de una regla semántica, debe estar seleccionada previamente. Para descargar el formato del mensaje, debe estar seleccionado el mensaje previamente.

Flujo normal de eventos	
Acción del Actor	**Respuesta del Sistema**
1- Seleccionar la opción configurar reglas semánticas del menú principal del subsistema.	2- El sistema muestra el listado de los mensajes existentes en la BD y en una tabla las reglas semánticas asociadas a estos. Existen las opciones: adicionar regla semántica, consultar regla semántica, eliminar regla semántica, modificar regla semántica, descargar formato del mensaje, descargar fichero de la regla semántica.
3- El usuario selecciona la opción deseada.	3.1- Si selecciona adicionar regla semántica ir a **Sección** Adicionar regla semántica. 3.2- Si selecciona consultar regla semántica ir a **Sección** Consultar regla semántica. 3.3- Si selecciona eliminar regla semántica ir a **Sección** Eliminar regla semántica. 3.4- Si selecciona modificar regla semántica ir a **Sección** Modificar regla semántica. 3.5- Si selecciona descargar formato del mensaje ir a **Sección** Descargar formato del mensaje. 3.6- Si selecciona descargar fichero de la regla semántica ir a **Sección** Descargar fichero de la regla semántica.
	4- El caso de uso se termina.

Sección: Adicionar regla semántica.	
Flujo normal de eventos	
Acción de Actor	**Respuesta del sistema**
1- Selecciona adicionar regla semántica.	2- El sistema muestra el formulario para introducir el fichero con la definición de la regla semántica.
3- El usuario introduce el fichero de la regla semántica y selecciona la opción aceptar.	4- Se valida el fichero introducido. Si no está correcto. **Alternativa 1,** Fichero Incorrecto. Se guarda el fichero de la regla semántica y se muestra el nombre de este en la tabla de las reglas asociadas a los mensajes.
	5- Finaliza la acción.
Sección: Consultar regla semántica	
Flujo normal de eventos	
Acción de Actor	**Respuesta del sistema**
1- Selecciona consultar regla semántica.	2- El sistema muestra una ventana con el texto de la regla semántica seleccionada.
	3- Finaliza la acción.
Sección: Eliminar regla semántica	
Flujo normal de eventos	
Acción de Actor	**Respuesta del sistema**
1- Selecciona eliminar regla semántica.	2- El sistema elimina la regla semántica seleccionada y muestra un mensaje: “Operación realizada satisfactoriamente”.
	3- Finaliza la acción.
Sección: Actualizar regla semántica	
Flujo normal de eventos	
Acción de Actor	**Respuesta del sistema**
1- Selecciona actualizar regla semántica.	2- El sistema muestra el formulario para introducir el nuevo fichero de la regla semántica.
3- El usuario introduce el fichero de la	4- Se valida el nuevo fichero introducido. Si no está

regla semántica y selecciona la opción aceptar.	correcto. **Alternativa 1,** Fichero Incorrecto. Se guarda el fichero de la regla semántica y se muestra el nombre de este en la tabla de las reglas asociadas a los mensajes.
	5- Finaliza la acción.
Sección: Descargar formato del mensaje	
Flujo normal de eventos	
Acción de Actor	**Respuesta del sistema**
1- Selecciona descargar formato del mensaje.	2- El sistema brinda la opción de guardar el formato del mensaje en formato XML.
3- El usuario escoge la dirección en la PC donde va a guardar el fichero del formato del mensaje y selecciona la opción aceptar.	4- Finaliza la acción.
Sección: Descargar fichero de la regla semántica	
Flujo normal de eventos	
Acción de Actor	**Respuesta del sistema**
1- Selecciona descargar fichero de la regla semántica.	2- El sistema brinda la opción de guardar la regla semántica en formato XML.
3- El usuario escoge la dirección en la PC donde va a guardar el fichero de la regla semántica y selecciona la opción aceptar.	4- Finaliza la acción.
Flujo alterno: Fichero Incorrecto	
Acción del actor	**Respuesta del sistema**
	1- El sistema muestra un mensaje referente al error que se está cometiendo.

2.5 Modelo de análisis

El Modelo de Análisis puede considerarse como una primera aproximación al Modelo de Diseño, aunque es un modelo por sí mismo. Es un modelo objetual que describe la realización de casos de uso,

proporciona una estructura centrada en el mantenimiento, la flexibilidad ante los cambios y la reutilización. El mismo contiene los resultados del análisis de los casos de uso expresados en clases del análisis. (34)

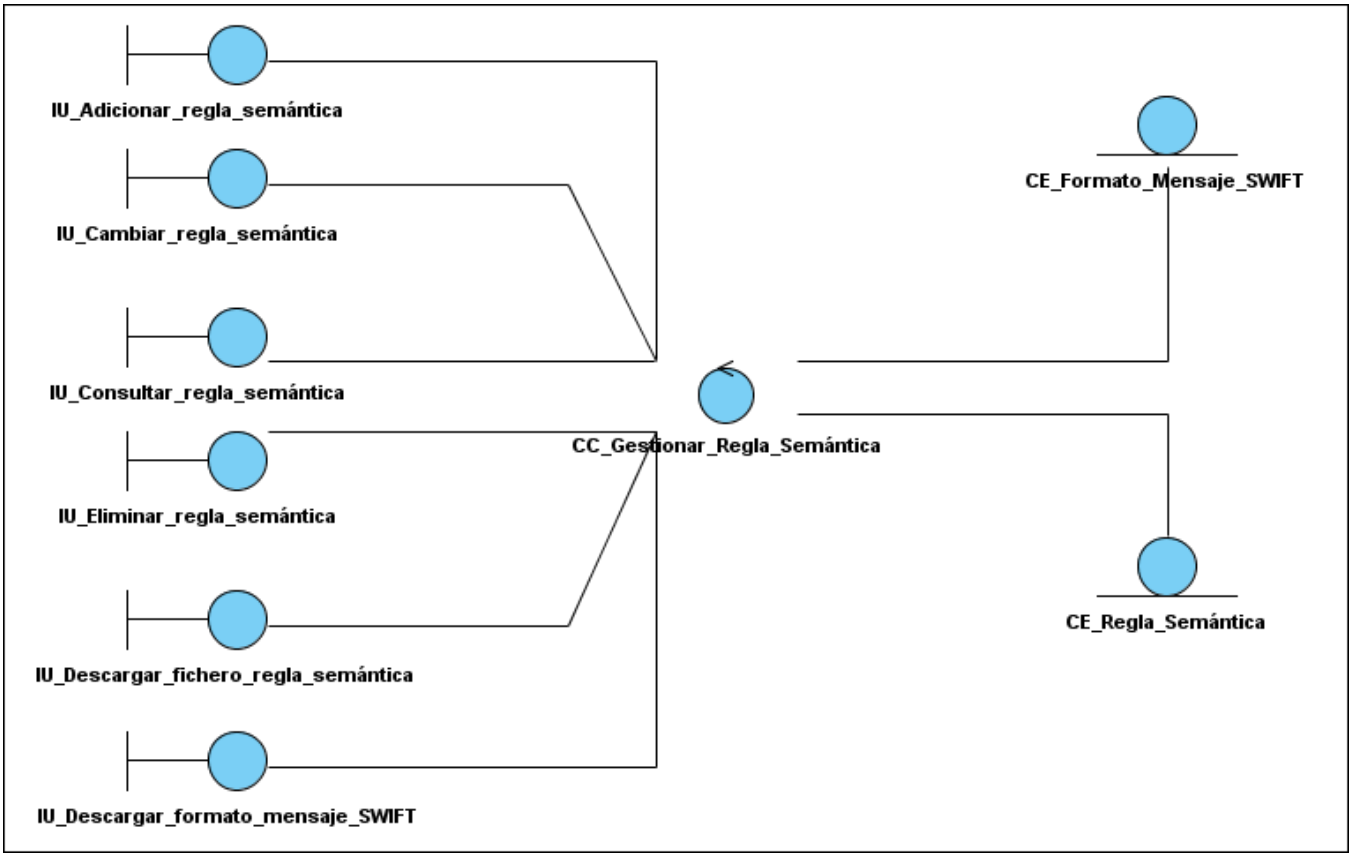


Fig. 2.2 Diagrama de clases del módulo Configuración de las reglas semánticas

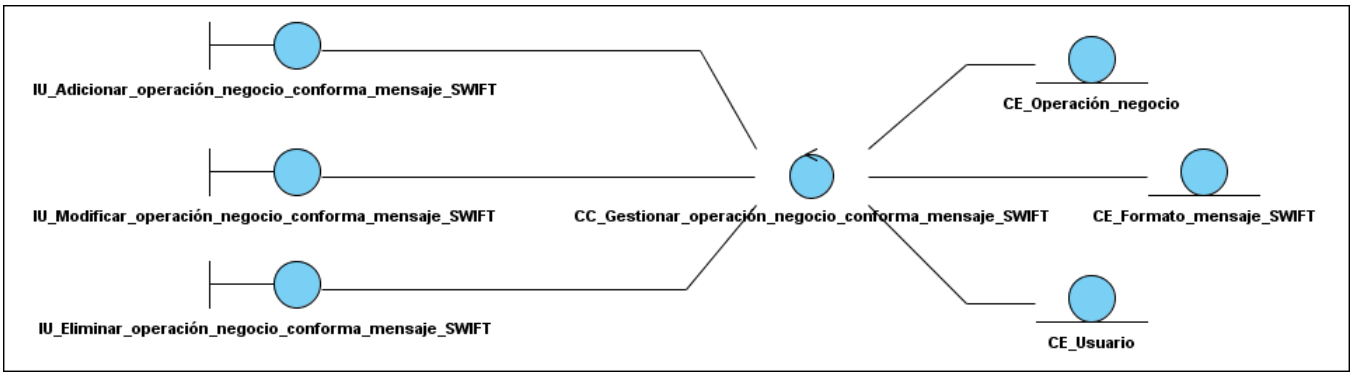


Fig. 2.3 Diagrama de clases del módulo Administración de las asociaciones de las operaciones de negocio que conforman mensajes SWIFT.

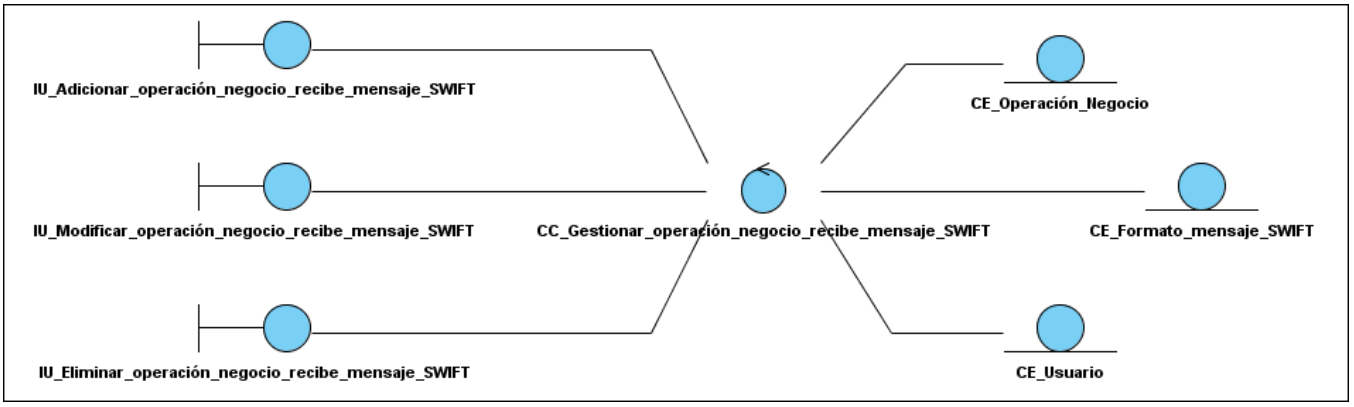


Fig. 2.4 Diagrama de clases del módulo Administración de las asociaciones de las operaciones de negocio que se ejecutan cuando se reciben mensajes SWIFT.

2.6 Modelo de Diseño

El modelo de diseño describe la realización física de los casos de uso centrándose en cómo los requisitos funcionales y no funcionales, junto con otras restricciones relacionadas con el entorno de implementación, tienen impacto en el sistema. Además, el modelo de diseño sirve de abstracción de la implementación del sistema. (34)

2.6.1 Estructura de capas del sistema

La estructura de capas escogida para el proyecto y sobre la que se realiza el desarrollo del software está compuesta por tres capas principales:

- Capa de Presentación.
- Capa de Negocio.
- Capa de Acceso a Datos.

Y además 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. (1)

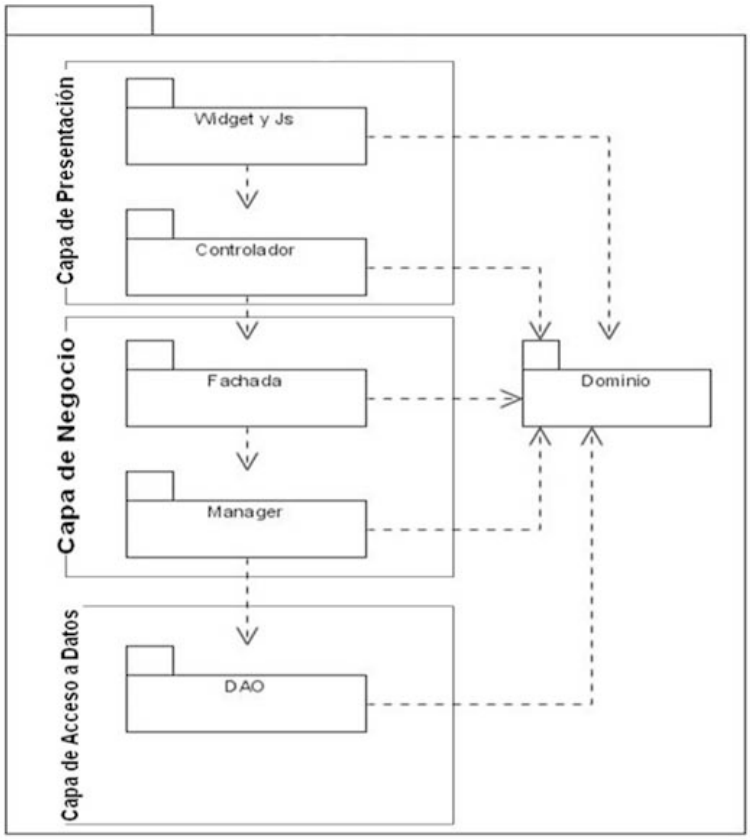


Fig. 2.5 Estructura de Capas de Quarxo

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 construir las interfaces que interactuarán con el usuario, además del 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 demanden el acceso a los datos de la aplicación. Para el desarrollo de esta capa se utilizará el patrón DAO (**D**ata **A**cces **O**bject en inglés), y la interacción con la capa de negocio será a través de sus interfaces.

2.6.2 Paquetes del diseño

Un paquete de diseño es una colección de clases, relaciones, realizaciones de casos de usos, diagramas y otros paquetes. Se utiliza para estructurar el modelo de diseño dividiéndolo en pequeñas partes. Para el desarrollo de este artefacto a continuación se detallan el diagrama de paquetes, las clases del diseño y la realización de casos de uso.

2.6.2.1 Diagrama de paquetes

El objetivo de estos diagramas es obtener una visión más clara del sistema de información orientado a objetos, organizándolo en subsistemas y/o módulos, agrupando los elementos del análisis, diseño o construcción y detallando las relaciones de dependencia entre ellos. A continuación se muestra el diagrama de paquete para los módulos Configuración de las reglas semánticas y Administración de las asociaciones de las operaciones de negocio que conforman y reciben mensajes SWIFT.

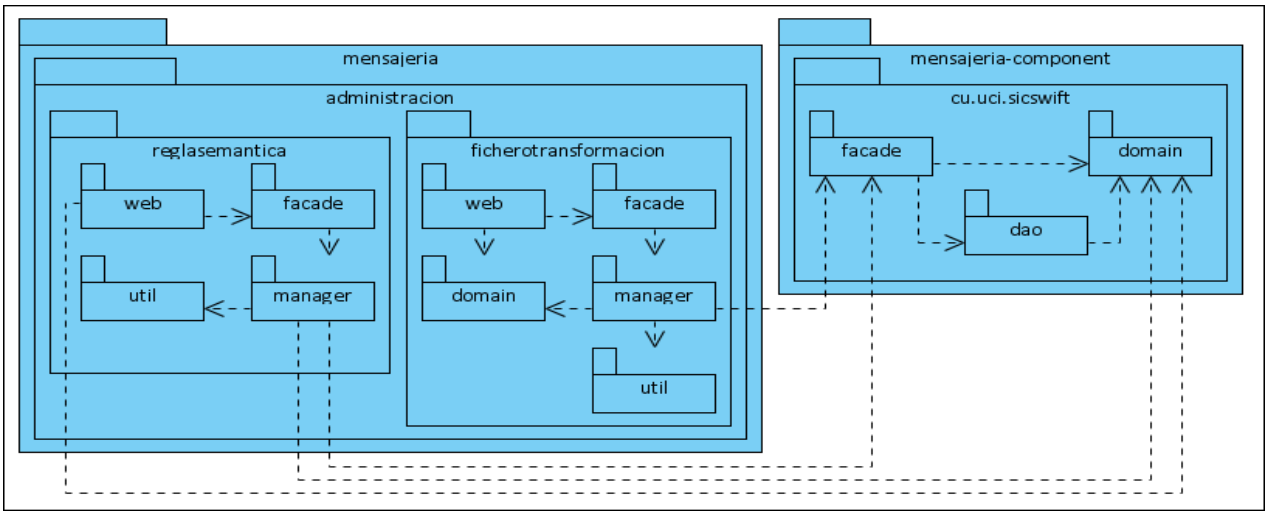


Fig. 2.6 Diagrama de paquetes de la propuesta de solución.

2.6.2.2 Clases del diseño

Las clases del diseño son representaciones de las clases utilizadas en la implementación del sistema, en ellas se describen las responsabilidades de los métodos implicados en el desarrollo de una funcionalidad determinada. Para estructurar las clases del diseño y asignarles las responsabilidades se utilizaron patrones de diseño que permiten lograr una arquitectura consistente.

Los patrones de diseño son descripciones de clases cuyas instancias colaboran entre sí. Cada patrón es adecuado para ser adaptado a un cierto tipo de problema. (37)

Para el diseño e implementación de los módulos Configuración de las reglas semánticas y Administración de las asociaciones de las operaciones de negocio que conforman y reciben mensajes SWIFT se utilizaron tanto patrones arquitectónicos, de diseño, como de J2EE. A continuación se encuentra la descripción de cómo fueron aplicados los mismos.

Patrones de arquitectura

Patrón Modelo vista controlador

Este patrón define que el sistema debe ser dividido en capas. Esto trae consigo una cantidad importante de beneficios y tanto en el componente de mensajería como en el subsistema se evidencian:

- Se puede entender una capa como un todo sin considerar las otras.
- Las capas se pueden sustituir con implementaciones alternativas de los mismos servicios básicos.
- Se minimizan dependencias entre capas.
- Las capas posibilitan la estandarización de servicios.
- Luego de tener una capa construida, puede ser utilizada por muchos servicios de mayor nivel.

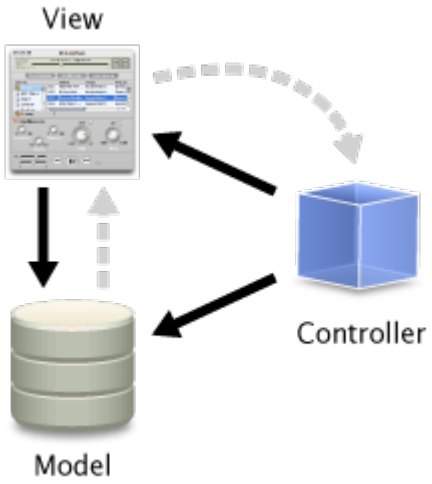


Fig. 2.7 Imagen del Patrón: Modelo Vista Controlador

Patrones de diseño: Asignación de responsabilidades (GRASP)

Patrón Bajo acoplamiento

El patrón bajo acoplamiento se evidencia con la definición de interfaces e implementaciones, como las clases interfaces ReglaSemanticaFacade y TransformacionFacade y sus implementaciones, estas permiten que las clases controladoras ReglaSemanticaMultiActionContoller y TransformacionMultiActionController interactúen con las clases de presentación y se relacionen únicamente con ellas para realizar sus operaciones, reduciendo el impacto de cambios posteriores en el negocio del sistema.

Patrón Alta cohesión

El sistema en general fue diseñado en módulos, por ejemplo: Configurar reglas semánticas, Administrar las asociaciones de las operaciones de negocio que conforman mensajes SWIFT y Administrar las

asociaciones de las operaciones de negocio que se ejecutan cuando se reciben mensajes SWIFT. Estos módulos fueron definidos para que en los mismos se manejaran la menor cantidad de entidades posibles con el objetivo de que a las clases pertenecientes a las diferentes capas se le asignaran las responsabilidades necesarias y bien delimitadas para garantizar una alta cohesión y generando así un bajo acoplamiento.

Patrón Experto

El patrón experto se evidencia en la definición de las clases de acuerdo a las funcionalidades que deben realizar a partir de la información que manejan. Específicamente: las clases ReglaDAO y OperacionDAO, serán las responsables de efectuar las operaciones que corresponden a la gestión de las reglas y las operaciones que conforman y reciben mensajes SWIFT.

Patrón Controlador

Las clases controladoras ReglaSemanticaMultiActionContoller y TransformacionMultiActionController constituyen un ejemplo de la aplicación de este patrón, las mismas tendrán en cuenta la responsabilidad de manejar los eventos que consisten en gestionar tanto las reglas semánticas como las operaciones que conforman y reciben mensajes SWIFT.

Patrones de diseño: GoF

Fachada

La utilización del patrón fachada se evidencia en la definición de las interfaces ReglaSemanticaFacade y TransformacionFacade y sus implementaciones, estas clases son las responsables de la comunicación entre la presentación y el grupo de clases e interfaces más complejas que se encargan de la lógica de negocio y el acceso a datos.

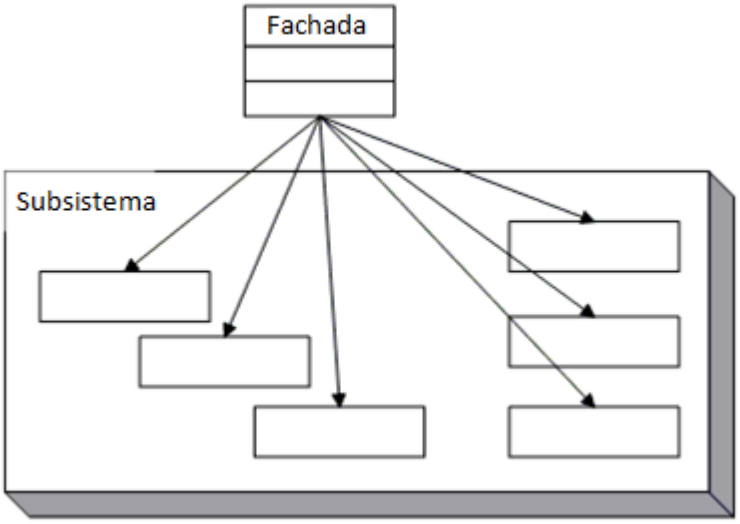


Fig. 2.8 Imagen del Patrón: Fachada

Patrón Cadena de responsabilidad

El patrón cadena de responsabilidad se aplica cuando desde la vista se solicita la información que se encuentra en la base de datos, esta es atendida primeramente por los controladores, luego por la fachada, el Manager y finalmente por el objeto de acceso a datos (DAO), evidenciándose de esta manera la utilización de dicho patrón.

Patrón Objeto de acceso a datos (DAO)

El patrón Objeto de acceso a datos (DAO) se evidencia en la definición de las clases interfaces ReglaDAO y OperacionDAO, estas son implementadas por las clases ReglaDAOImpl y OperacionDAOImpl donde se encuentran las funcionalidades específicas a realizar sobre la base de datos. De esta forma el negocio no será afectado de posibles cambios que puedan producirse en la lógica de acceso a datos y la fuente de datos.

Patrones de J2EE

Vistas Compuestas (Composite View)

El patrón de vistas compuestas se evidencia en la construcción de vistas complejas, específicamente se puede encontrar en la creación de la vista gestionarReglaSemanticaVista.jsp, la cual usa en su

construcción las vistas consultarReglaVista.jsp, subirFicheroReglaVista.jsp y subirFicheroReglaActualizarVista.jsp.

2.6.2.3 Diagrama de clases del Diseño

El diagrama de clases del diseño describe gráficamente las especificaciones de las clases de software y de las interfaces en una aplicación. Este contiene la información siguiente:

- Clases, asociaciones y atributos.
- Interfaces, con sus operaciones y constantes.
- Métodos.
- Información sobre los tipos de los atributos.
- Navegabilidad.
- Dependencias.

El diagrama de clases del diseño Adicionar regla semántica se encuentra en el anexo 2. En el mismo intervienen un grupo de clases, a continuación se describen las más importantes:

ReglaSemanticaMultiActionController

Es la encargada de atender las peticiones que relacionadas con el caso de uso adicionar Regla semántica, consta de los métodos *listarMensajes()*, y *subirFichero()*. El método *listarMensajes()* es el encargado de devolver a la vista los mensajes almacenados en la base de datos (BD) para que estos sean escogidos posteriormente por el usuario y asociarle el fichero de la regla semántica deseado. Por otra parte el método *subirFichero()* es el que recibe el mensaje seleccionado y el fichero de la regla semántica que se desea asociar al mismo.

ReglaSemanticaFacadeImpl

Esta clase está diseñada cumpliendo lo planteado por el patrón Facade y su objetivo es servir de único punto de acceso a la implementación del negocio de la aplicación.

ReglaSemanticaManagerImpl

Esta clase se encarga de manejar el negocio del módulo. Cuenta con métodos como *validarFicheroReglaSemantica()*, en donde se valida el contenido del fichero de la regla semántica subido a la aplicación haciendo uso de las clases que se encuentran en el paquete *util*: *ReglaSemanticaUtil* y *ValidaReglaXSD*. Otro método con que cuenta esta clase es *persistirRegla()* que se encarga luego de haberse validado el fichero de la regla, persistirlo en la BD, auxiliándose de la clase de acceso a datos *RuleDAOImpl*.

En el diagrama de clases del diseño del CU Adicionar operación del negocio que conforma mensajes SWIFT (consultar anexo 2) se hizo uso del framework Spring Webflow para la definición de los flujos. La clase *TransformacionServiceFlow* es la encargada de atender los eventos que se generen en la presentación. La clase *TransformacionManagerImpl* se encarga de implementar todo el negocio referido al CU haciendo uso de las clases contenidas en el paquete *util* para validar que los ficheros de transformación adicionados concuerden con el formato de los mensajes seleccionados que se encuentran almacenados en la BD, además la clase *TransformacionManagerImpl* usa las clases contenidas en el paquete *cu.uci.sicswift.dao* del componente mensajería para todo el acceso a datos; aclarar que en este paquete se encuentran solamente declaradas las interfaces para ahorrar espacio en la imagen.

El siguiente diagrama de clases del diseño es el que modela el CU Eliminar operación del negocio que se ejecuta cuando se reciben mensajes SWIFT (consultar anexo 2). En este la clase *TransformacionMultiactionController* es la encargada de atender las peticiones que se realicen en la presentación, y a su vez acceder a través de la fachada *TransformacionFacade* a las funcionalidades implementadas en la clase *TransformacionManagerImpl*. Esta última contiene métodos que responden al negocio del CU, y el acceso a datos lo realiza a través de las clases declaradas en el paquete *cu.uci.sicswift.dao* del componente mensajería. Aclarar que en este paquete se encuentran solamente declaradas las interfaces para ahorrar espacio en la imagen y que esta contribuya a un mejor entendimiento.

2.6.2.4 Diagramas de Interacción

Los diagramas de interacción son diagramas que describen cómo grupos de objetos colaboran para conseguir algún fin. Estos diagramas muestran los objetos, así como los mensajes que pasan entre ellos

dentro del caso de uso. Los diagramas de interacción capturan el comportamiento de los casos de uso y se expresan de dos formas: diagramas de colaboración y diagramas de secuencias.

A continuación se muestran los diagramas de secuencia de mayor impacto para la propuesta de solución.

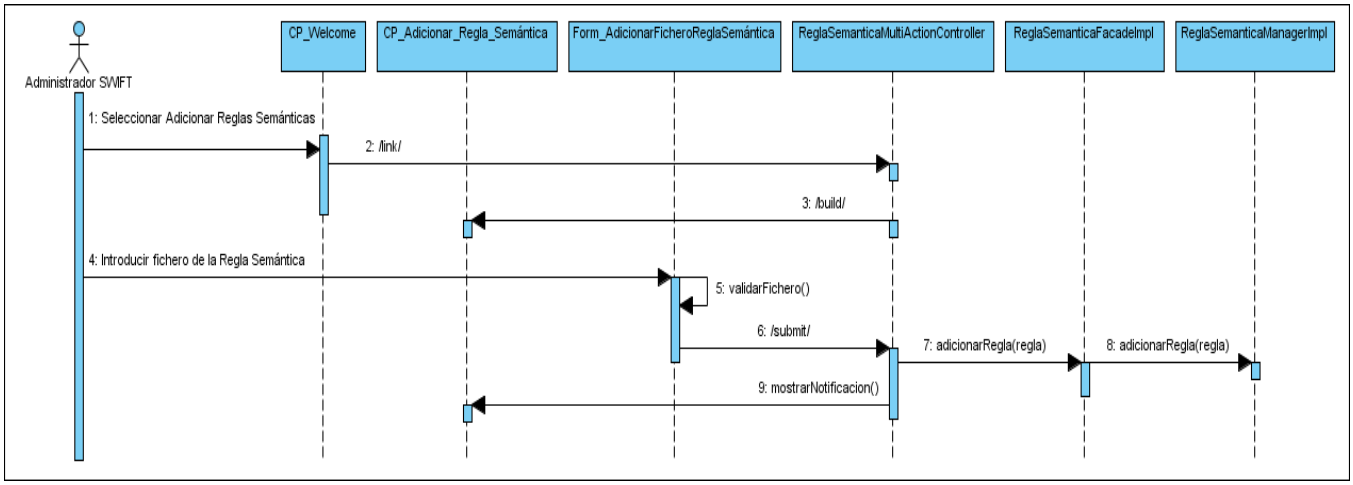


Fig. 2.9 Diagrama de Secuencia: Adicionar Regla semántica

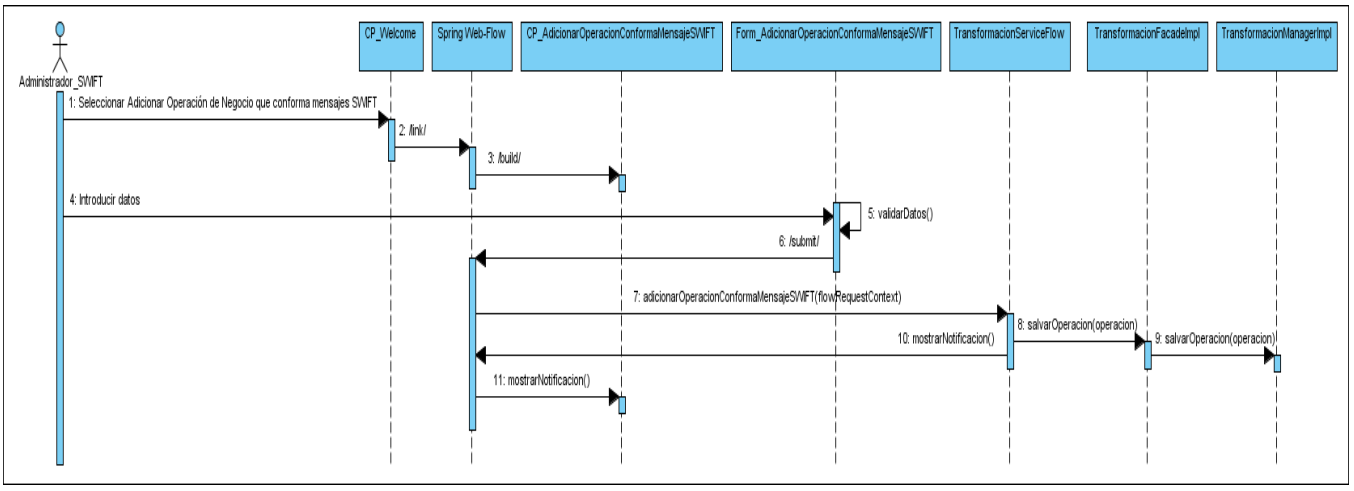


Fig. 2.10 Diagrama de Secuencia: Adicionar Operación de Negocio que conforma mensajes SWIFT

2.6.3 Modelo de datos

Los modelos de datos aportan la base conceptual para diseñar aplicaciones que hacen un uso intensivo de datos, así como la base formal para las herramientas y técnicas empleadas en el desarrollo y uso de sistemas de información. Permiten además, describir los elementos de la realidad que intervienen en un problema dado y la forma en que se relacionan esos elementos entre sí. Luego de realizar un estudio e identificar un conjunto de clases persistentes se creó un modelo de datos que responde al dominio de la propuesta de solución.

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_numero_mensaje_operacion, o_numero_mensaje_operacion_usuario, o_operacion* y *o_numero_mensaje* se encargan almacenar y relacionar cada una de las operaciones con los mensajes correspondientes y con los usuarios que pueden ejecutar dicha operación.

Por último las tablas o_rule y o_formato_swift_o_rule son las encargadas de almacenar el texto de las reglas semánticas y relacionar las reglas con los mensajes correspondientes.

El modelo de datos se presenta a continuación:

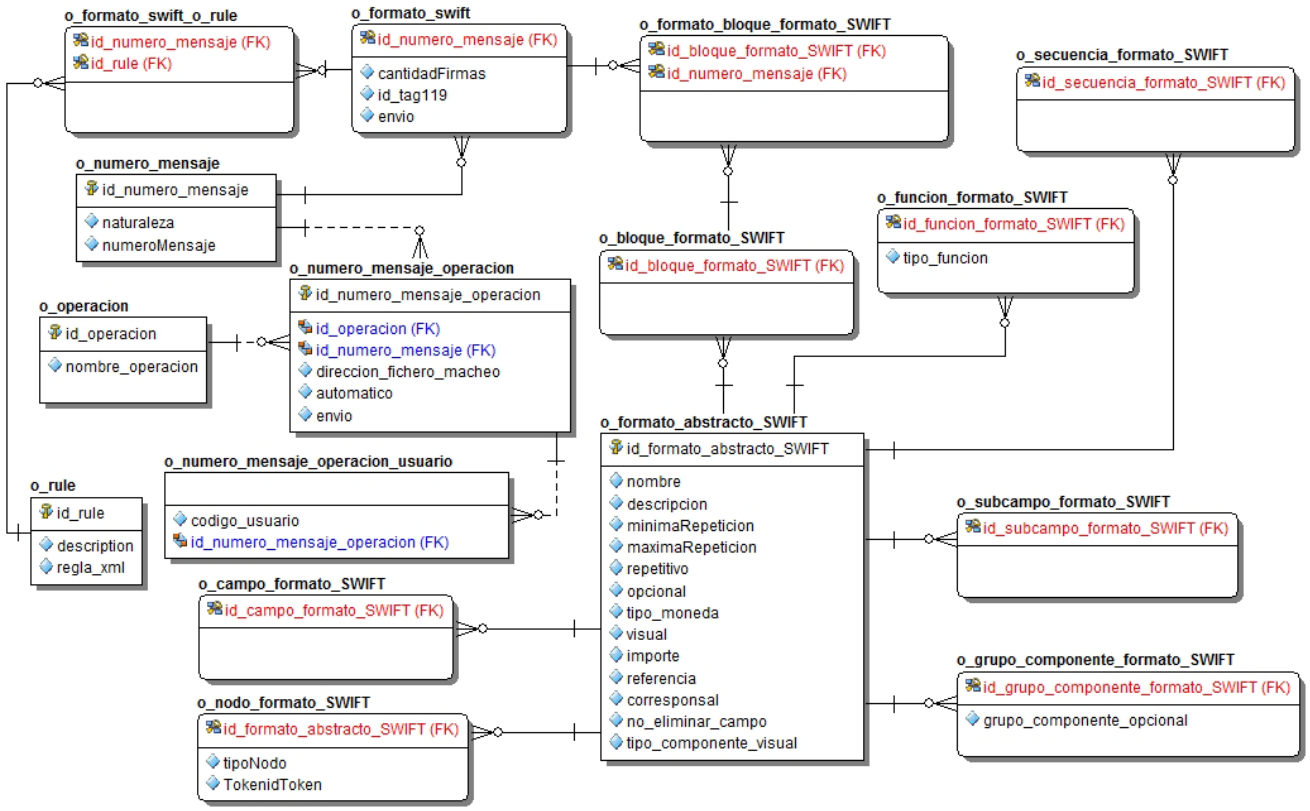


Fig. 2.11 Modelo de datos de la propuesta de solución

Conclusiones Parciales

En este capítulo se realizó el análisis y diseño de la propuesta de solución, para esto se utilizaron varios patrones que organizan el trabajo y posibilitan que el diseño sea entendible, flexible y sus componentes reutilizables. Además guiándose por la metodología seleccionada (RUP) se crearon una serie de diagramas y especificaciones que posteriormente servirán de guía para realizar el modelo de implementación, ayudando así a que el desarrollo de los módulos propuestos se realice en el menor tiempo posible.

CAPÍTULO 3: IMPLEMENTACIÓN Y PRUEBA DE LA PROPUESTA DE SOLUCIÓN

3.1 Introducción

En el presente capítulo se aborda lo referente a la implementación y prueba del sistema. Se describe el modelo de implementación utilizado, los diagramas de componente y de despliegue, así como las pruebas efectuadas para verificar el buen funcionamiento del sistema.

3.2 Modelo de implementación

El modelo de implementación describe cómo los elementos del modelo de diseño, como las clases, se implementan en términos de componentes, como ficheros de código fuente, ejecutables, etc. El modelo de implementación describe también cómo se organizan los componentes de acuerdo con los mecanismos de estructuración y modularización disponibles en el entorno de implementación y en el lenguaje o lenguajes de programación utilizados, y cómo dependen los componentes unos de otros. (34)

3.2.1 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. Estos diagramas contienen relaciones de dependencia que se utilizan para indicar que un componente se refiere a los servicios ofrecidos por otro componente. En el siguiente diagrama se muestran los componentes de los módulos siguiendo la arquitectura Modelo Vista Controlador. (38)

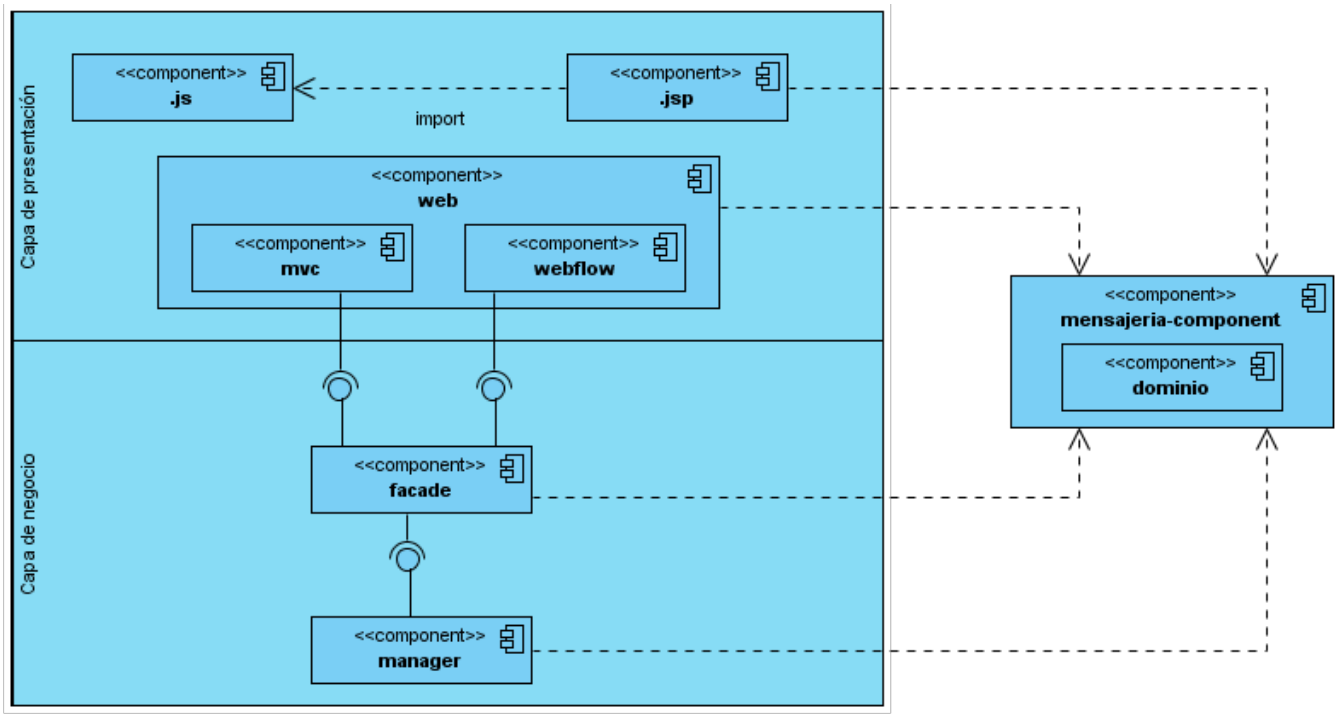


Fig. 3.1 Diagrama de componentes

3.2.2 Estándares de codificación

Los estándares de codificación son pautas de programación que no están enfocadas a la lógica del programa, sino a su estructura 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.

3.2.2.1 Convenciones de nombres

Como principio todos los nombres serán en español exceptuando aquellos que correspondan al nombre de un patrón conocido, ejemplo: Facade, Builder, etc.

Clases

Los nombres de las clases deben ser sustantivos, en el caso de ser compuestos tendrán la primera letra de cada palabra que lo forme en mayúscula. Los nombres de las clases deben ser simples y descriptivos.

Además, se deben usar palabras completas, evitar acrónimos y abreviaturas (exceptuando los casos en los que la abreviatura sea mucho más conocida que el nombre completo, como URL o HTML).

Métodos

Los métodos deben ser en infinitivo, cuando son compuestos tendrán la primera letra en minúscula, y la primera letra de las siguientes palabras que lo forma en mayúscula.

Ejemplo: consultarReglaSemantica().

Atributos

Los nombres de los atributos de las clases y variables internas comienzan con la primera letra en minúscula, en caso de que sea un nombre compuesto las siguientes palabras comenzarán con la primera letra en mayúscula.

Ejemplo: transformacionFacade.

3.2.2.2 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: TransformacionMultiActionController.

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

Ejemplo: TransformacionServiceFlow.

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: ReglaSemanticaFacade, ReglaSemanticaFacadeImpl.

Las clases que se encuentran dentro del paquete manager se les agrega después del nombre la palabra Manager. Para el subpaquete impl se les nombra igual que la interfaz que implementan y se le agrega la palabra Impl.

Ejemplo: ReglaSemanticaManager, ReglaSemanticaManagerImpl.

3.2.3 Descripción de las principales clases utilizadas

A continuación se describen las clases más importantes desde el punto de vista funcional para los módulos Configuración de las reglas semánticas y Administración de las asociaciones de las operaciones de negocio que conforman y reciben mensajes SWIFT.

Nombre: ReglaSemanticaMultiActionController	
Tipo de clase: Controladora	
Atributo	Tipo
resourceLoader	ResourceLoader
upload	ServletFileUpload
reglaSemanticaFacade	ReglaSemanticaFacade
Para cada responsabilidad:	
Nombre:	Descripción:
listarMensajes(HttpServletRequest request, HttpServletResponse response)	Permite obtener un listado con todos los nombres e id de los mensajes definidos en la base de datos.
consultarRegla(HttpServletRequest request, HttpServletResponse response)	Permite obtener la definición de una regla semántica en forma de texto.
listarReglas(HttpServletRequest request,	Permite obtener un listado con todos los

HttpServletResponse response)	nombres e id de las Reglas semánticas asociadas a un mensaje.
cargarTextoRegla (HttpServletRequest request, HttpServletResponse response)	Permite obtener el texto de una regla semántica en forma de un archivo en formato XML.
cargarTextoFormato(HttpServletRequest request, HttpServletResponse response)	Permite obtener el texto del formato de un mensaje en forma de un archivo en formato XML.
subirFicheroRegla(HttpServletRequest request, HttpServletResponse response)	Permite adicionar una regla semántica y asociarla al mensaje deseado.
subirFicheroReglaActualizar(HttpServletRequest request, HttpServletResponse response)	Permite cambiar una regla semántica ya insertada por una nueva.
eliminarRegla(HttpServletRequest request, HttpServletResponse response)	Permite eliminar una regla semántica.

Nombre: ReglaSemanticaManagerImpl	
Tipo de clase: Manager	
Atributo	**Tipo**
formatoMensajeSwiftDAO	FormatoMensajeSwiftDAO
reglaDAO	ReglaDAO
validaReglaXSD	ValidaReglaXSD
Para cada responsabilidad:	
Nombre:	**Descripción**:
listarMensajesConId()	Permite obtener un listado de nombre de mensajes con su id.
listadoReglasPorMsg(String idNumeroMensaje)	Permite obtener un listado con todas las reglas

	semánticas asociadas a un mensaje.
getReglaPorId(String idRegla)	Permite obtener una regla semántica por su id.
getXMLBloquePorIdFormato(int idFormato)	Permite obtener el texto del formato de un mensaje.
persistirRegla(Rule regla, int idFormato)	Permite persistir una regla en la base de datos y asociarla al mensaje deseado.
actualizarRegla(Rule rule, String idRegla, int idFormato)	Permite actualizar una regla semántica.
validarFicheroReglaSemantica(String reglaPath, FileItem reglaFile, int idFormato)	Permite validar el fichero de la regla semántica subido a la aplicación.

Nombre: TransformacionMultiActionController	
Tipo de clase: Controladora	
Atributo	Tipo
resourceLoader	ResourceLoader
upload	ServletFileUpload
transformacionFacade	TransformacionFacade
Para cada responsabilidad:	
Nombre:	Descripción:
subirFicheroXSLT(HttpServletRequest request, HttpServletResponse response)	Permite persistir una operación de negocio en la base de datos y asociarla a los mensajes deseados.
actualizarOperacionRecepcion(HttpServletRequest request, HttpServletResponse response)	Permite actualizar una operación de negocio.

listarOperaciones(HttpServletRequest request, HttpServletResponse response)	Permite obtener un listado con todos los nombres e id de las operaciones de negocio almacenadas en la base de datos.
listarNumeroMensajeOperacionPorIdOperacion(HttpServletRequest request, HttpServletResponse response)	Permite obtener un listado con todos los nombres e id de las operaciones de negocio, así como los nombres de los ficheros de transformación.
listarFicherosXSLT(HttpServletRequest request, HttpServletResponse response)	Permite obtener un listado con todos los nombres e id de los ficheros de transformación.
eliminarXSLT(HttpServletRequest request, HttpServletResponse response)	Permite eliminar la asociación entre una operación contable y un formato de un mensaje.
eliminarOperacionRecepcion(HttpServletRequest request, HttpServletResponse response)	Permite eliminar una operación.

3.3 Pruebas

Las pruebas en un proceso de desarrollo de software son los diferentes procesos que se deben realizar con el objetivo de asegurar la terminación y calidad de la solución propuesta. Existen diferentes métodos de pruebas, entre ellos se pueden encontrar:

Las pruebas de caja negra: 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, además 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. (39)

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. En estas pruebas se examina el programa en varios puntos sobre el código para determinar si el estado real coincide con el esperado. 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. Estas pruebas se llevan a cabo en primer lugar sobre un módulo concreto, para luego realizar las de caja negra sobre varios subsistemas.

El objetivo de las pruebas es la detección de defectos en el software (descubrir un error es el éxito de una prueba). Con el diseño de las pruebas se pretende encontrar y documentar los defectos que puedan afectar la calidad del software, asegurar que el software trabaje como fue diseñado, además de validar que los requisitos fueron implementados correctamente.

3.3.1 Pruebas de caja blanca

Objetivo

El objetivo de realizar este tipo de prueba al sistema es que se garantice que se ejerciten por lo menos una vez todos los caminos independientes de cada módulo o método, todos los bucles en sus límites operacionales así como las estructuras internas de datos para asegurar su validez. (39)

Alcance

El proceso de pruebas de caja blanca se va a concentrar principalmente en validar a través del framework de software libre JUnit, que cada uno de los módulos o segmentos de códigos funcione apropiadamente.

Descripción

La prueba de caja blanca es considerada como uno de los tipos de pruebas más importantes que se le aplican a los software, logrando como resultado que disminuya en un gran por ciento el número de errores existentes en los sistemas y por ende una mayor calidad y confiabilidad. (39)

Para el desarrollo de estas pruebas unitarias se utilizó framework JUnit, ya que ofrece las funcionalidades necesarias para implementar pruebas en un proyecto desarrollado en Java. Además cuenta con una

interface simple que informa si cada una de las pruebas realizadas o conjunto de pruebas falló, pasó o fue ignorada.

Inicialmente se definieron los casos de prueba, uno por cada clase implementada. Luego se definieron e implementaron los ficheros de configuración necesarios para establecer la comunicación entre las diferentes capas de la aplicación. Se identificaron los métodos a probar dentro de cada caso de prueba, así como el resultado esperado en cada uno. Y por último se procedió a realizar los casos de prueba diseñados.

Con el objetivo de mostrar la realización de los casos de prueba, a continuación se muestran un grupo de imágenes con sus descripciones.

```
25 public class ReglaSemanticaTest extends TestCase {
26
27     private ReglaSemanticaMultiActionController reglaSemanticaMultiActionController;
28
29     private MockControl controlHttpServletRequest;
30     private HttpServletRequest mockHttpServletRequest;
31
32     private MockControl controlHttpServletResponse;
33     private HttpServletResponse mockHttpServletResponse;
34
35     private SessionFactory sf;
36
```

Fig. 3.2 Declaración de los atributos en la clase de prueba

Se crea una clase en la que se van a definir los casos de prueba implementados para probar las funcionalidades que se encuentran en la clase *ReglaSemanticaMultiActionController*. Primero se declara un atributo de dicha clase, luego dos *mocks*[14] para simular los objetos *HttpServletRequest* y *HttpServletResponse* que se le deben pasar a los métodos a probar, estos objetos son controlados por objetos de tipo *MockControl*, por último uno de tipo *SessionFactory* necesario para que Hibernate se comunique con la base de datos.

[14] **Mocks:** *son objetos simulados que imitan el comportamiento de los objetos reales en forma controlada. Normalmente se crea un objeto mock para probar el comportamiento de algún otro objeto.*

```
37    @Before
38    protected void setUp() throws Exception {
39
40        ApplicationContext context = new ClassPathXmlApplicationContext(
41                "classpath:/cu/uci/finixubnc/mensajeria/administracion/test/mensajeria-context.xml");
42
43        reglaSemanticaMultiActionController = (ReglaSemanticaMultiActionController) context
44                .getBean("ReglaSemanticaMultiAction");
45
46        sf = (SessionFactory) context.getBean("mensajeriaSessionFactory");
47        TransactionSynchronizationManager.bindResource(sf, new SessionHolder(sf.openSession()));
48
49        controlHttpServletRequest = MockControl
50                .createControl(HttpServletRequest.class);
51        mockHttpServletRequest = (HttpServletRequest) controlHttpServletRequest
52                .getMock();
53
54        controlHttpServletResponse = MockControl
55                .createControl(HttpServletResponse.class);
56        mockHttpServletResponse = (HttpServletResponse) controlHttpServletResponse
57                .getMock();
58
59        super.setUp();
60    }
```

Fig. 3.3 Implementación del método setUp() en la clase de prueba

El método *setUp()* es donde se inicializan todos los atributos declarados. Para esconder y hacer más entendible el caso de prueba, se decidió poner en el fichero *mensajeria-context.xml* todo el proceso de inicializar el objeto *reglaSemanticaMultiActionController* debido a su complejidad. Los objetos *mockHttpServletRequest, controlHttpServletRequest, mockHttpServletResponse, y controlHttpServletResponse* son creados usando la librería *EasyMock*.

```
70    @Test
71    public void testConsultarReglaSemanticaC12() {
72        String idRegla = "C12.xml";
73        String esperado = getResultadoEsperadoPorIdRegla(idRegla);
74
75        mockHttpServletRequest.getParameter("id");
76        controlHttpServletRequest.setReturnValue(idRegla);
77
78        controlHttpServletRequest.replay();
79
80        assertEquals(esperado, reglaSemanticaMultiActionController.consultarRegla(
81                mockHttpServletRequest, mockHttpServletResponse).get("texto"));
82    }
```

Fig 3.4 Implementación del caso de prueba #1 al método consultarRegla() de la clase controladora ReglaSemanticaMultiActionController

```
84    @Test
85    public void testConsultarReglaSemanticaC02_6() {
86        String idRegla = "C02-6.xml";
87        String esperado = getResultadoEsperadoPorIdRegla(idRegla);
88
89        mockHttpServletRequest.getParameter("id");
90        controlHttpServletRequest.setReturnValue(idRegla);
91
92        controlHttpServletRequest.replay();
93
94        assertEquals(esperado, reglaSemanticaMultiActionController.consultarRegla(
95                mockHttpServletRequest, mockHttpServletResponse).get("texto"));
96
97    }
```

Fig. 3.2 Implementación del caso de prueba #2 al método consultarRegla() de la clase controladora ReglaSemanticaMultiActionController

En las imágenes anteriores se muestran dos pruebas al método *consultarRegla(HttpServletRequest request, HttpServletResponse response)*, primero se preparan los objetos mock para pasarle los datos necesarios al método que se desea probar. En este caso se le agrega al objeto mockHttpServletRequest el parámetro *id*, primero pasándole el valor *C12.xml* y luego *C02-6.xml*. En la última línea del método de prueba se hace la llamada al método *assertEquals* el cual recibe como parámetros el resultado esperado y la ejecución del método a probar para que este evalúe si coinciden ambos.

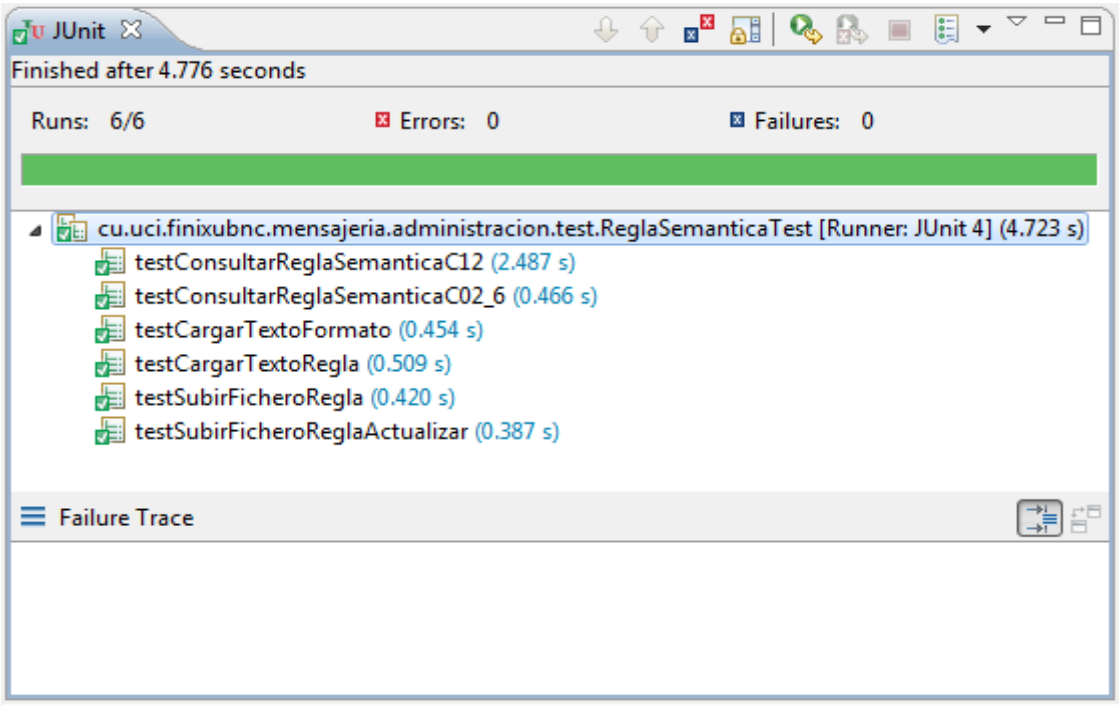


Fig. 3.6 Resultado de la ejecución de las pruebas

JUnit muestra una interfaz con los resultados de las pruebas realizadas, en este caso se muestran 6 pruebas a los métodos de la clase *ReglaSemanticaMultiActionController* donde todas arrojaron resultados satisfactorios. La implementación de los métodos más importantes probados se puede consultar en el anexo 3 de la versión digital.

3.3.2 Pruebas de caja negra

Objetivo

El objetivo de realizar este tipo de prueba al sistema es para detectar el incorrecto o incompleto funcionamiento de este, así como los errores de interfaces, rendimiento y errores de inicialización y terminación.

Alcance

El proceso de pruebas de caja negra se va a centrar principalmente en los requisitos funcionales del software para verificar el comportamiento de la unidad observable externamente y la calidad funcional.

Descripción

Estas pruebas permiten obtener un conjunto de condiciones de entrada que ejerciten completamente todos los requisitos funcionales del programa. En ellas se ignora la estructura de control, concentrándose en los requisitos funcionales del sistema y ejercitándolos.

La prueba de caja negra no es una alternativa a las técnicas de prueba de la caja blanca, sino un enfoque complementario que intenta descubrir diferentes tipos de errores a los encontrados en los métodos de la caja blanca. (40)

Existen varias técnicas para desarrollar la prueba de caja negra, entre ellas están: (41)

- Técnica de la Partición de Equivalencia: divide el campo de entrada de un programa en clases de datos de los que se pueden derivar casos de prueba. Un caso de prueba ideal descubre de forma inmediata una clase de errores que, de otro modo, requerirían la ejecución de muchos casos antes de detectar el error genérico. En esencia esta técnica se dirige a la definición de casos de prueba que descubran clases de errores, reduciendo así el número total de casos de prueba que hay que desarrollar.

- Técnica del Análisis de Valores Límites: los errores tienden a darse más en los límites del campo de entrada que en el centro. Esta técnica lleva a una elección de casos de prueba que ejerciten los valores límites.

    - ✓ Condiciones sublímite: las condiciones límite normales son las más obvias de descubrir. Estas son definidas en la especificación o son evidentes al momento de utilizar el software. Algunos límites, sin embargo, son internos al software, no son necesariamente aparentes al usuario final pero aún así deben ser probadas por el probador. Estas son conocidas como condiciones sublímite o condiciones límite internas.

- Técnica de prueba basada en grafos: permite al encargado de la prueba validar complejos conjuntos de acciones y condiciones.

    - ✓ Modelado del flujo de transacción: los nodos representan los pasos de alguna transacción, y los enlaces representan las conexiones lógicas entre los pasos.

- ✓ Modelado de estado finito: los nodos representan diferentes estados del software observables por el usuario, y los enlaces representan las transiciones que ocurren para moverse de estado a estado.
- ✓ Modelado de flujo de datos: los nodos objetos de datos y los enlaces son las transformaciones que ocurren para convertir un objeto de datos en otro.
- ✓ Modelado de planificación: los nodos son objetos de programa y los enlaces son las conexiones secuenciales entre esos objetos. Los pesos de enlace se usan para especificar los tiempos de ejecución requeridos al ejecutarse el programa.
- ✓ Gráfica Causa-efecto: la gráfica Causa-efecto representa una ayuda gráfica en seleccionar, de una manera sistemática, un gran conjunto de casos de prueba. Tiene un efecto secundario beneficioso en precisar estados incompletos y ambigüedades en la especificación.

Dentro de las técnicas de prueba de caja negra se utilizó la de Partición Equivalente. Esta técnica permite obtener un conjunto de pruebas que reducen el número de casos de prueba que se deben realizar para evaluar correctamente el software, esto se debe a que se centra en la evaluación de clases de equivalencia que representan un conjunto de estados válidos o no válidos para condiciones de entradas existentes en el software.

Para definir las clases de equivalencia se tuvieron en cuenta un conjunto de reglas: (40)

- Si una condición de entrada especifica un rango, entonces se confeccionan una clase de equivalencia válida y dos inválidas.
- Si una condición de entrada especifica la cantidad de valores, se identifica una clase de equivalencia válida y dos inválidas.
- Si una condición de entrada especifica un conjunto de valores de entrada y existen razones para creer que el programa trata en forma diferente a cada uno de ellos, se identifica una clase válida para cada uno de ellos y una clase inválida.

- Si una condición de entrada especifica una situación de tipo “debe ser”, se identifica una clase válida y una inválida.

Luego de tener las clases válidas e inválidas definidas, se diseñaron los casos de prueba para el caso de uso Configurar reglas semánticas, el mismo puede ser consultado en el anexo 4 de la versión digital. Cada uno de estos casos de prueba se definió teniendo en cuenta lo siguiente: (40)

- Escribir un nuevo caso de prueba que cubra tantas clases de equivalencia válidas no cubiertas como sea posible hasta que todas las clases de equivalencia hayan sido cubiertas por casos de prueba.

- Escribir un nuevo caso de prueba que cubra una y solo una clase de equivalencia inválida hasta que todas las clases de equivalencias inválidas hayan sido cubiertas por casos de prueba.

3.3.3 Resultado de las pruebas

Luego de la realización de los métodos de prueba, como parte de las pruebas internas realizadas a la propuesta de solución se obtuvieron resultados satisfactorios desde el punto de vista interno y funcional, atendiendo al correcto comportamiento de la misma ante diferentes situaciones. Se aplicaron los métodos de prueba de caja blanca y caja negra para validar tanto la interfaz como el correcto funcionamiento interno del software. Los resultados arrojados por ambas pruebas contribuyeron a validar la calidad de la solución implementada.

Los módulos desarrollados fueron sometidos además a pruebas de aceptación por parte del cliente. Las mismas fueron realizadas en el Banco Nacional de Cuba donde sus resultados fueron satisfactorios, obteniendo así un aval por parte del Banco Central de Cuba como constancia de la validación de la solución propuesta, el mismo puede ser consultado en el anexo 3.

Conclusiones Parciales

En este capítulo se creó el modelo de implementación, se definieron y pusieron en práctica estándares de codificación en la implementación de la solución propuesta, lo que ayudó a tener un código más uniforme, claro y fácil de mantener. Además, al finalizar el mismo ya se cuenta con la solución implementada completamente, pudiéndose contar con las funcionalidades necesarias para hacer frente a los cambios

que puedan ocurrir en el estándar de mensajería SWIFT y en las asociaciones de las operaciones de negocio con los mensajes SWIFT correspondientes. También se realizaron pruebas estructurales y funcionales donde todas arrojaron resultados satisfactorios desde el punto de vista interno y funcional, corroborando así la calidad con la que fueron desarrollados los módulos implementados. Por último se realizaron las pruebas de aceptación por parte del cliente donde este dio constancia de que la solución propuesta cumple con todos los requisitos exigidos por ellos.

CONCLUSIONES GENERALES

Durante el desarrollo del presente trabajo se analizaron diferentes cuestiones que aportaron conocimientos sobre los Sistemas de Gestión Bancaria, centrándose en los procesos de comunicación financieros a los que estos deben someterse. El estudio de sistemas similares demostró la necesidad e importancia de desarrollar los módulos propuestos con el objetivo de flexibilizar el sistema de intercambio de mensajes SWIFT, ya que los sistemas estudiados son propietarios y por tanto a nuestro país se le dificulta el pago de altos precios por ellos.

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 el análisis, diseño e implementación de la propuesta de solución, el lenguaje UML facilitó la realización de los modelos, el lenguaje de programación proporcionó la codificación y la metodología ayudó a documentar todo lo relacionado con la solución.

Una vez desarrollados e integrados los módulos implementados, se les hicieron pruebas estructurales y funcionales para detectar y corregir los errores existentes, luego les fue presentada la propuesta de solución a los clientes donde estos emitieron un aval exponiendo que el software cumple con los requisitos solicitados por ellos. Quedando así cumplido el objetivo propuesto.

La utilización de ambos módulos contribuye a flexibilizar el sistema de intercambio de mensajes SWIFT dotando al subsistema mensajería de Quarxo con las funcionalidades necesarias para hacer frente a los cambios que ocurran anualmente en el estándar de mensajería utilizado y permitiendo de una forma más fácil la integración de las operaciones contables con el envío y recepción de mensajes SWIFT.

RECOMENDACIONES

Tomando como base la investigación realizada y la experiencia acumulada durante el desarrollo del presente trabajo se recomienda:

- Añadir los módulos implementados al subsistema de mensajería SWIFT del sistema Quarxo que se encuentra en explotación en el BNC.
- Continuar el estudio del tema con el objetivo de incluir una interfaz de usuario que permita la creación de las reglas semánticas de una manera más fácil para los usuarios finales.

REFERENCIAS BIBLIOGRÁFICAS

1. **Chaviano, Adolfo Miguel Iglesias.** *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.

2. —. *Análisis de las tecnologías compatibles con la norma ISO 15022 de SWIFT.* La Habana : s.n., 2011.

3. **Microsoft Corporation.** Transformar archivos XML con XSLT en Access 2002. [En línea] http://office.microsoft.com/es-es/access-help/transformar-archivos-xml-con-xslt-en-access-2002-HA001034576.aspx?CTT=1.

4. **SIBANC.** *Manual de Usuario del SISCOM.* La Habana : s.n., 2006.

5. **Corporation, Informatica.** Natixis basa su mensajería SWIFT e infraestructura de datos en la solución integrada de Informatica PowerCenter y B2B Data Transformation. [En línea] http://www.informatica.com/INFA_Resources/cs_natixis_6728_es.pdf.

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

7. Clavelink, Consultoría y Servicios. [En línea] http://www.clavelink.com/incentage.php?idioma=es.

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

9. **Sanchez, María A. Mendoza.** Metodologías de Desarrollo de Software. [En línea] 2009. http://www.informatizate.net/articulos/metodologias_de_desarrollo_de_software_07062004.html.

10. eumed.net - Enciclopedia Virtual. *Por qué utilizar RUP para desarrollar aplicaciones web.* [En línea] http://www.eumed.net/libros/2009c/584/Por%20que%20utilizar%20RUP%20para%20desarrollar%20aplicaciones%20web.htm.

11. Best Java IDE. [En línea] http://faq.programmerworld.net/programming/best-java-ide-review.html.

12. **Agramonte, Alberto Jesus La Rosa y Del Castillo Rodríguez, Hiran.** *Herramienta para la instalación y configuración de clústeres del Contenedor de Servlets Apache Tomcat.* La Habana : s.n., 2009.

13. Community driven open source middleware. [En línea] http://www.jboss.org/.

14. GlassFish - Open Source Application Server. [En línea] http://glassfish.java.net/.

15. Apache Tomcat. [En línea] http://tomcat.apache.org/.

16. Control de versiones. [En línea] http://producingoss.com/es/vc.html.

17. Control de versiones con Git. [En línea] http://ecentinela.com/control-de-versiones-con-git/.

18. **Corporation, Microsoft.** Información general del producto SQL Server 2005. [En línea] Enero de 2007. http://www.microsoft.com/spain/sql/productinfo/overview/default.mspx.

19. ¿Qué es PHP? *Dattatec Ayuda.* [En línea] http://dattatecayuda.com/%C2%BFque-es-php/1860.

20. Microsoft Corporation. [En línea] 2012. http://www.microsoft.com/visualstudio/en-us/products/2010-editions/visual-csharp-express.

21. Chapter 15. Integrating with other web frameworks. [En línea] http://static.springsource.org/spring/docs/2.5.x/reference/web-integration.html.

22. *Core JavaServer™ Faces, Second Edition.* Santa Clara, California : s.n., 2007. 978-0-13-173886-7.

23. Los frameworks JS más usados. [En línea] 2007. http://www.anieto2k.com/2006/10/11/los-frameworks-js-mas-usados/.

24. **Vera Santana, Diana Cristina, Lara Vásquez, Rubén Alejandro y Echeverría Briones, Pedro Fabricio.** *Implementación de un portal web para la automatización del proceso de consultorías de mentores GOLD de la Región Latinoamericana del IEEE (R9), utilizando arquitectura Java 2 Enterprise Edition - J2EE y tecnología AJAX.* Guayaquil, Ecuador : s.n., 2009.

25. **The Dojo Foundation.** Insuperable Herramienta Javascript. [En línea] http://dojotoolkit.org/.

26. **Cuevas, David Benavides.** Departamentos de lenguajes y sistemas informáticos. [En línea] http://www.lsi.us.es/docencia/get.php?id=1130.

27. **Marquina, Ernesto y Parra, José David.** *Guía de Patrones, Prácticas y Arquitectura .NET.* 2007.

28. **Joaquin Gracia Murugarren.** Prácticas y métodos para mejorar el desarrollo de Proyectos de Software. *Patrones de diseño. Diseño de Software Orientado a Objetos.* [En línea] Mayo de 2005. http://www.ingenierosoftware.com/analisisydiseno/patrones-diseno.php.

29. **Visconti, Marcello y Astudillo, Hernán.** Fundamentos de Ingeniería de Software. *Patrones de Diseño.* [En línea] http://www.inf.utfsm.cl/~visconti/ili236/Documentos/08-Patrones.pdf.

30. Diseño de software con patrones. [En línea] http://www.programacion.com/articulo/diseno_de_software_con_patrones_parte_4_145#joa_patrones3_fachada.

31. Patrones de diseño - La cadena de responsabilidades. [En línea] http://www.portalfox.com/index.php?name=Sections&req=viewarticle&artid=157&page=1.

32. **Ramiro Lago Bagüés.** Tecnología orientada a procesos de negocio. *Patrón "Data Access Object".* [En línea] Abril de 2007. http://www.proactiva-calidad.com/java/patrones/DAO.html.

33. **Pardo, Otto Colomina.** Patrones de diseño. [En línea] http://www.jtech.ua.es/j2ee/2003-2004/bloque_pd.htm.

34. **Jacobson, Ivar, Booch, Grady y Rumbaugh, James.** *El proceso unificado de desarrollo de software.* Madrid : Pearson Educación S.A, 2000. 84-7829-036-2.

35. Definición de Requisito - Concepto y Significado. [En línea] http://www.carlospes.com/minidiccionario/requisito.php.

36. Tipos de requisitos: Funcional vs. No Funcional. [En línea] 2008. http://synergix.wordpress.com/2008/07/07/requisito-funcional-y-no-funcional/.

37. **Lago, Ramiro.** Patrones de diseño software. [En línea] Abril de 2007. http://www.proactiva-calidad.com/java/patrones/index.html.

38. Departamento de Informática, Chile. [En línea] http://www.inf.utfsm.cl/~visconti/ili236/Documentos/15-Implementacion.pdf.

39. **Pressman, Roger S.** *Ingeniería del Software. Un enfoque práctico.* España : McGraw-Hil, 2002.

40. Grupo Alarcos - Universidad de Castilla-La Mancha. [En línea] http://alarcos.inf-cr.uclm.es/doc/ISOFTWAREI/Tema09.pdf.

41. Métodos de prueba de caja negra. [En línea] http://gemini.udistrital.edu.co/comunidad/grupos/arquisoft/fileadmin/Estudiantes/Pruebas/HTML%20-%20Pruebas%20de%20software/node28.html.

42. **Márque, Dayanis Pacheco y Rangel Garcé, Sandra.** *Componente para la configuración de Reportes Web.* La Habana : s.n., 2011.

ANEXOS

Anexo 1: Especificaciones de casos de uso

Caso de Uso:	Administración asociación de las operaciones de negocio que conforman mensajes SWIFT.
Actores:	Administrador SWIFT.
Resumen:	Permite realizar acciones en el sistema sobre las operaciones de negocio como: insertar, eliminar o modificar.
Precondiciones:	El administrador SWIFT debe estar autenticado en el sistema. Para eliminar una operación de negocio, debe estar seleccionada previamente. Para modificar una operación de negocio, debe estar seleccionada previamente.
Flujo normal de eventos	
Acción del Actor	**Respuesta del Sistema**
1- Seleccionar la opción administrar operaciones de negocio que conforman mensajes SWIFT del menú principal del subsistema.	2- El sistema muestra el listado de las operaciones existentes en la BD. Existen las opciones: registrar nueva operación, actualizar operación y eliminar operación.
3- El usuario selecciona la opción deseada.	3.1- Si selecciona registrar nueva operación ir a **Sección** Registrar nueva operación. 3.2- Si selecciona actualizar operación ir a **Sección** Actualizar operación. 3.3- Si selecciona eliminar operación ir a **Sección** Eliminar operación.
	4- El caso de uso se termina.
Sección: Registrar nueva operación.	
Flujo normal de eventos	
Acción de Actor	**Respuesta del sistema**
1- Selecciona registrar nueva operación.	2- El sistema muestra el formulario para introducir los datos de la operación.

3- El usuario introduce los datos y selecciona la opción aceptar.	4- Se validan los datos introducidos. Si no están correctos. **Alternativa 1,** Datos incorrectos. Se registra la operación y se muestra el nombre de esta en la tabla de las operaciones registradas.
	5- Finaliza la acción.
Sección: Eliminar operación	
Flujo normal de eventos	
Acción de Actor	**Respuesta del sistema**
1- Selecciona eliminar operación.	2- El sistema elimina la operación seleccionada y muestra un mensaje: “Operación realizada satisfactoriamente”.
	3- Finaliza la acción.
Sección: Actualizar operación	
Flujo normal de eventos	
Acción de Actor	**Respuesta del sistema**
1- Selecciona actualizar operación.	2- El sistema muestra el formulario con los datos de la operación seleccionada para que sean modificados.
3- El usuario modifica los datos deseados y selecciona la opción aceptar.	4- Se validan los datos de la operación. Si no están correctos. **Alternativa 1,** Datos incorrectos. Se actualiza la operación y se muestra la tabla de las operaciones registradas.
	5- Finaliza la acción.
Flujo alterno: Datos incorrectos	
Acción del actor	**Respuesta del sistema**
	1- El sistema muestra un mensaje referente al error que se está cometiendo.

Anexo 2: Diagramas de clases del diseño

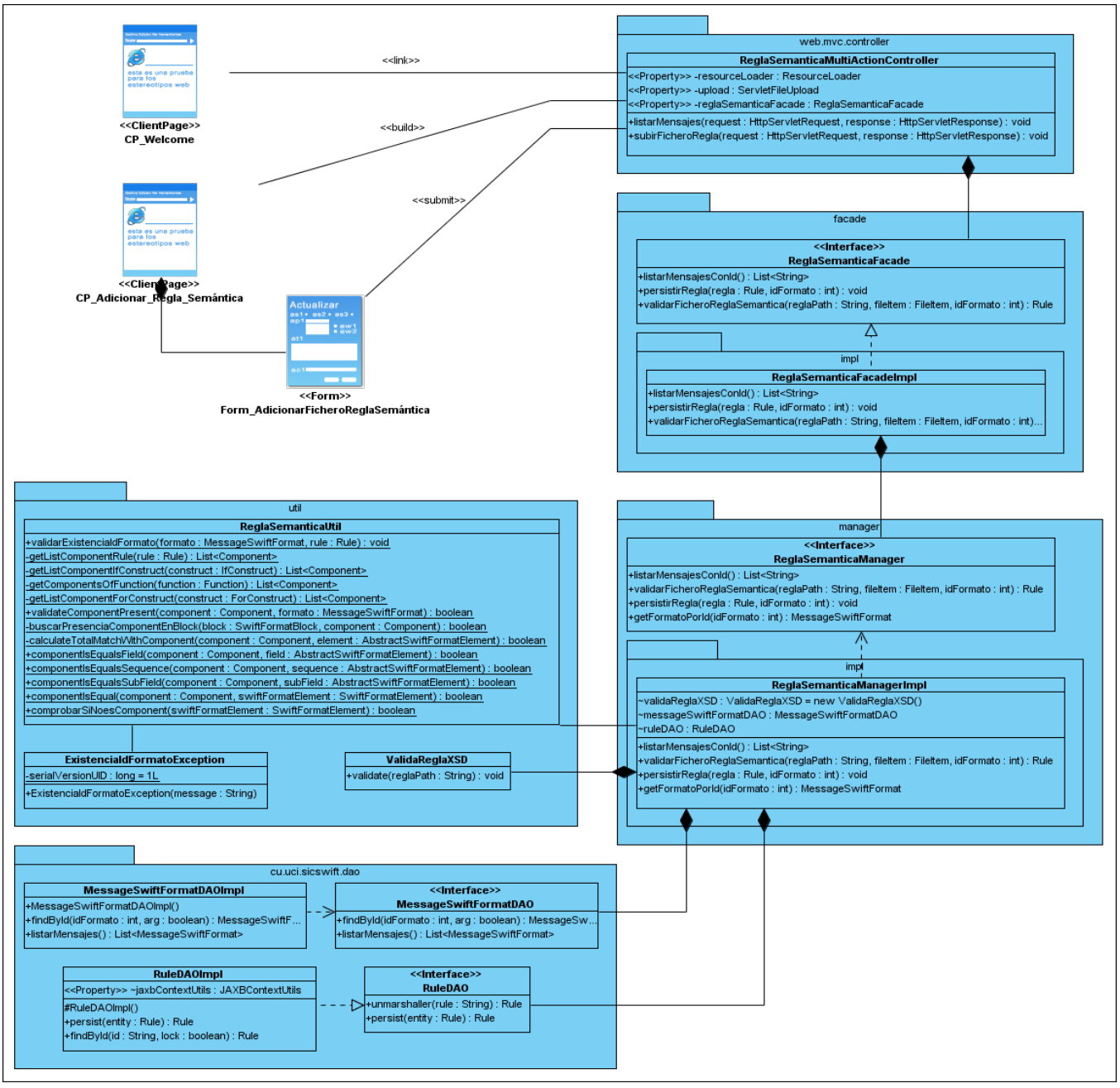


Ilustración 1. Diagrama de clases del diseño del CU: Adicionar Regla semántica

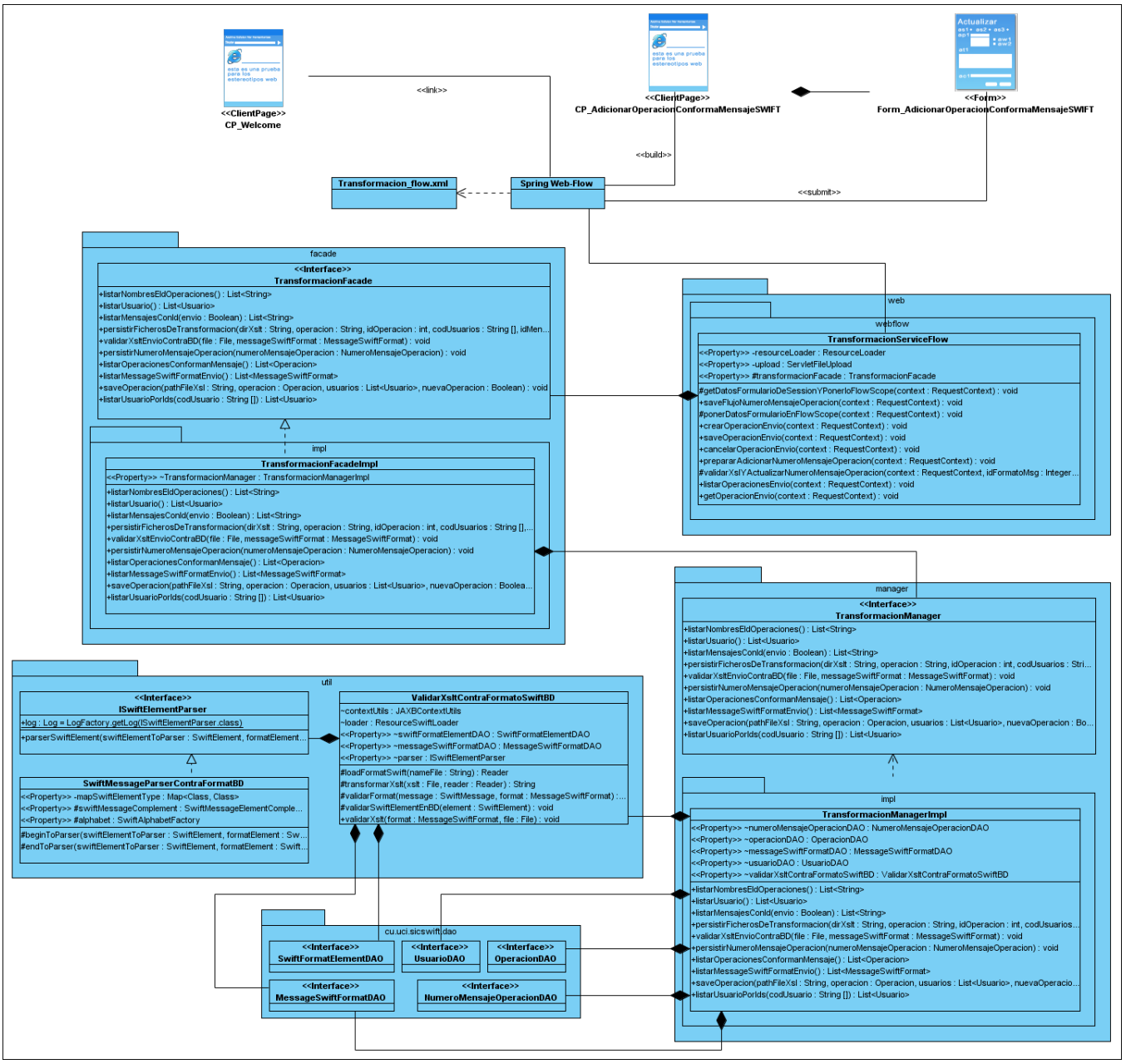


Ilustración 2. Diagrama de clases del diseño del CU: Adicionar operación del negocio que conforma mensajes SWIFT

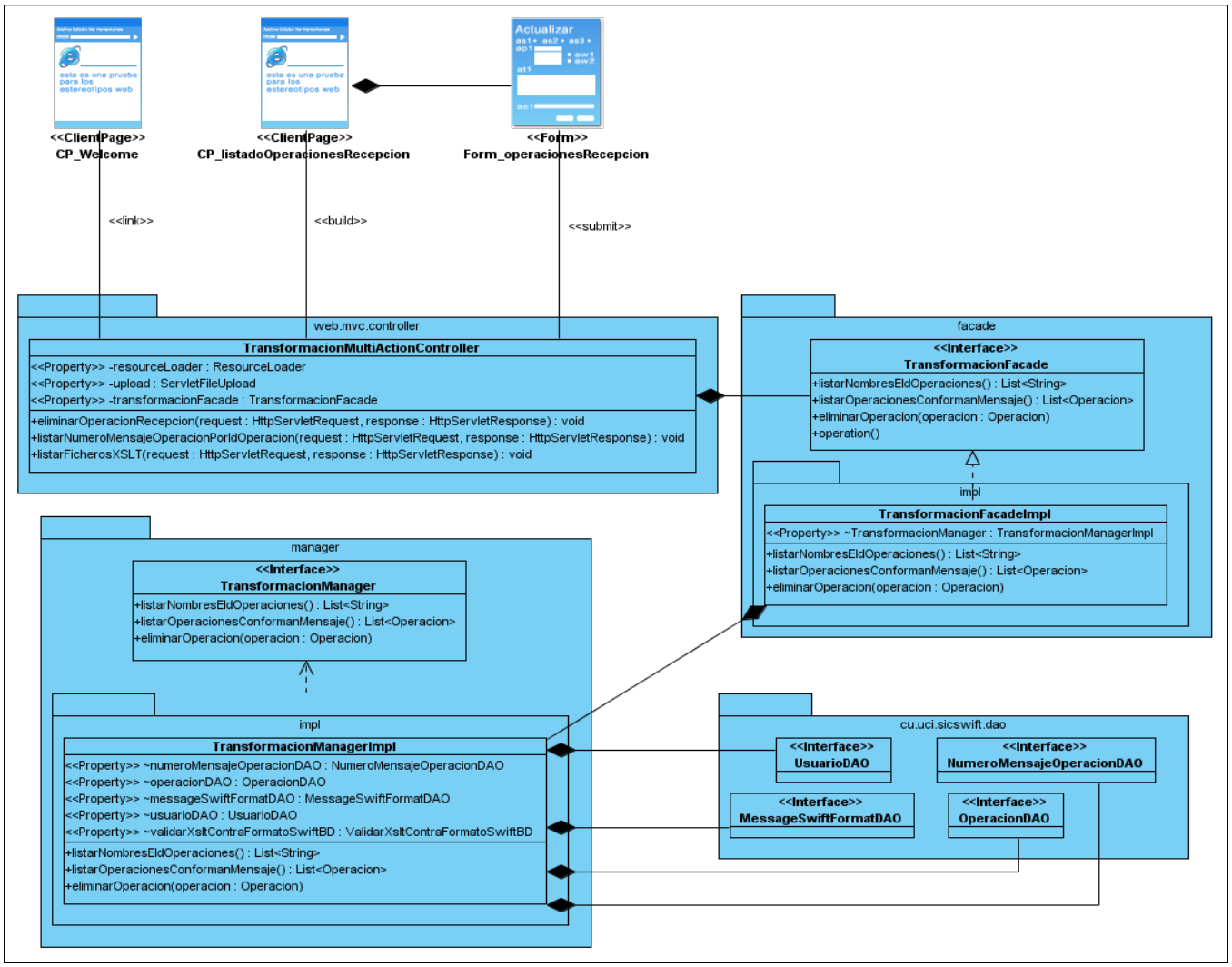


Ilustración 3. Diagrama de clases del diseño del CU: Eliminar operación del negocio que se ejecuta cuando se reciben mensajes SWIFT

Anexo 3: Certificado de aceptación por el BNC del subsistema Mensajería SWIFT del sistema Quarxo.

BANCO CENTRAL DE CUBA
Servicios Informáticos
Dirección de Sistemas

Aguiar 411 Esq. Lamparilla
La Habana, Cuba CP 10100

Tel. :(537) 862 7141
Fax: (537) 866 6258

5 de Marzo de 2012
"Año 54 de la Revolución"

Asunto: Certificación para el subsistema de mensajería SWIFT desarrollado por la UCI para el Banco Nacional de Cuba en el sistema Quarxo.

A favor de Adolfo Miguel Iglesias Chaviano, actual maestrante de la UCI y jefe del proyecto para el tratamiento de la mensajería SWIFT, cuyo significado de sus siglas en inglés corresponde a: Sociedad Mundial Internacional para el manejo de Transacciones Financieras. Certifico el cumplimiento de los estándares de SWIFT compatible con la Norma ISO 15022.

El proyecto cumple con la estructura general de los mensajes, las validaciones sintácticas y semánticas; soportando los continuos cambios que puedan ocurrir en la Norma con respecto a los elementos que conforman el formato de los mensajes: bloques, secuencias de texto, campos, subcampos, atributos o componentes y funciones básicas.

Señalando como elementos positivos, la garantía de una estructura de clases para representar visualmente los mensajes sin contener plantillas predefinidas y la integración con Aplicaciones Empresariales y/o integración con bases datos para la aplicación BackOffice, desarrollada en el banco que se conecta a la red SWIFT.

Atendiendo a lo anterior y para que conste ante las instancias pertinentes de la UCI, firmo el documento.

Fraternalmente,

Verónica Elvira Torres
Gerente de Operaciones
Dirección de Sistemas de la Unidad Adscrita de Servicios Informáticos
Banco Central de Cuba

Ilustración 4. Certificado de aceptación por el BNC del subsistema Mensajería SWIFT del sistema Quarxo