Universidad de las Ciencias Informáticas

Facultad 3


Título:

# SUBSISTEMA MENSAJERÍA SLBTR PARA EL SISTEMA QUARXO

**Trabajo de Diploma para optar por el título de**

**Ingeniero en Ciencias Informáticas**

**Autores:** Evelio Clavel Rosales

Dariel Membribes Rodríguez

**Tutora:** Ing. Lissett Díaz Mesa

La Habana, 2 de julio de 2012

"Año 54 de la Revolución"

## DECLARACIÓN DE AUTORÍA

Declaramos ser autores de este trabajo 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 julio del año 2012.

| | |
|---|---|
| _____ | _____ |
| Evelio Clavel Rosales | Dariel Membribes Rodríguez |
| Autor | Autor |

_____

Ing. Lissett Díaz Mesa

Tutora

## AGRADECIMIENTOS


**De Dariel**

*A todos los que de una forma u otra contribuyeron en mi formación, en especial a todos los profesores que me han impartido clases a lo largo de mi trayectoria como estudiante. A mi compañero de tesis Evelio. A mi tutora y demás miembros del proyecto que tanto me han ayudado en mi preparación y superación, en especial a Manuel, Ray, Jorge, Rafael, Leosvel y a Adrián.*

*A todos mis compañeros de estudio que a lo largo de estos cinco años me han ayudado y apoyado, en especial a todos los que han compartido el aula conmigo. A mis amigos Ivette, Frank Solana, Lino, Karina, Frank Padrón, Mayito, Juan José y a mi brigada. En especial quiero agradecer por estar siempre a mi lado cuando lo necesito a mis hermanos y hermanas: Jessica y Yadini, Rojas, Evelio, Hector, Chavelys y Efraín.*

**De Evelio**

*A mi madre, a mi padre, a mi hermano Reyner, a mi familia.*

*A mi compañero de tesis y a mi tutora.*

*A Héctor y Chavelys.*

*A Andy y Abel.*

*A Manuel, a Bello, a Ray Williams, a Jorge Fonseca, a mis amigos del proyecto Quarxo.*

*A mis amigos de siempre, de la UCI, del IPVCE y de Buey Arriba.*

*A mi gente buena del BNC.*

*A Su, y a mis amigos del CCE.*

*A todos los que contribuyeron a mi formación.*

*A mis profesores.*

*A la FEU.*

*A la UCI.*

## DEDICATORIA

**De Dariel**

*A mi mamá Elia Idalmi y a mi papá Enelio por darme todo el apoyo que he necesitado, por su sacrificio y entrega en todo momento, por todo lo que soy en la vida, sin ellos no hubiera logrado llegar tan lejos. Por ser los mejores padres del mundo.*

*A mi hermana y a mis sobrinas.*

*A mi novia Milay y a su familia por ser tan buenas personas.*

**De Evelio**

*Con mucho amor*

*A mi madre, a mi padre, a mi hermano Reyner, a mi familia.*

## RESUMEN

El uso de las modernas tecnologías en los procesos financieros, unido a la complejidad de la estructura bancaria cubana, han propiciado que el Banco Central de Cuba (BCC) establezca mediante la resolución 48 del 2009, un sistema para la realización en tiempo real de transacciones u operaciones de pago entre las diferentes entidades, utilizando medios electrónicos de comunicación y teniendo como centro al BCC para monitorear el flujo de las transferencias interbancarias que se realicen. Este sistema se conoce como Sistema de Liquidación Bruta en Tiempo Real (SLBTR), en el cual participan instituciones que componen el sistema bancario nacional, y tiene definido para el intercambio de información entre los participantes el uso de los mensajes SLBTR. En el Banco Nacional de Cuba (BNC) dichos mensajes por su estructura flexible cambian con frecuencia provocando retrasos en los procesos de gestión. El presente trabajo contribuye a erradicar dichas dificultades mediante el desarrollo del subsistema Mensajería SLBTR de Quarxo. Se analizan las soluciones existentes para la gestión de mensajes SLBTR en el ámbito nacional e internacional, y se describen las principales tecnologías utilizadas y la metodología de desarrollo del *software*, así como los elementos fundamentales de la arquitectura, análisis, diseño, implementación y los resultados de las pruebas. Se obtuvo una aplicación web multiplataforma que reduce el tiempo y el consumo de recursos de materiales cuando se requiere gestionar los mensajes SLBTR en el BNC, además, permite la administración del formato de dichos mensajes sin la intervención de un programador en el proceso.



Palabras clave: mensaje SLBTR, Quarxo, sistema de pago, SLBTR, *software* de gestión, transacción.

# ÍNDICE

# INTRODUCCIÓN

Desde hace algunos años la industria bancaria viene haciendo uso de las nuevas Tecnologías de la Informática y las Comunicaciones (TIC) para mejorar sus procesos y servicios; evidenciándose, en el incremento de las operaciones entre las instituciones financieras, tales como órdenes de pago y transferencias [24; 28]. Por lo que se hacen necesarios mecanismos que permitan el acceso y el intercambio fluido de la información financiera entre entidades bancarias, eliminándose las barreras entre estas, pues las transacciones financieras, resultan vitales para la eficiencia y estabilidad de los sistemas bancarios [38].

Cuba cuenta con un sistema bancario compuesto por el Banco Central de Cuba (BCC), los bancos comerciales radicados en el país y las entidades financieras no bancarias con licencia para operar en el territorio nacional [2]. Debido a la complejidad de la estructura bancaria existente y al uso de las modernas tecnologías en los procesos financieros, el BCC como entidad rectora del sistema bancario cubano, establece mediante la resolución 48 del año 2009 un sistema para la realización en tiempo real de transacciones u operaciones de pago entre las diferentes entidades, utilizando medios electrónicos de comunicación y teniendo como centro al BCC para monitorear el flujo de las transferencias interbancarias que se realicen. Este sistema se conoce como Sistema de Liquidación Bruta en Tiempo Real (SLBTR), en el cual participan instituciones que componen el sistema bancario nacional, y tiene definido para el intercambio de información entre los participantes el uso de los mensajes SLBTR [3].

Con la modernización del Sistema Bancario Cubano, el Banco Nacional de Cuba (BNC) inició un proyecto de colaboración con el Centro de Informatización de la Gestión de Entidades (CEIGE), de la Universidad de las Ciencias Informáticas (UCI). El objetivo de este proyecto es desarrollar un *software* que mejore la gestión de todos sus procesos.

Actualmente, el BNC como entidad participante en el SLBTR, tiene en explotación el SABIC (Sistema Automatizado para la Banca Internacional de Comercio), en su versión para el sistema operativo MS-DOS. Pero, contrario a las indudables ventajas de su utilización, este *software* se ha quedado rezagado frente los avances de las TIC que han ido aconteciendo en los últimos años [11; 12].

El SABIC para la gestión de los mensajes SLBTR posee limitadas funcionalidades y algunos problemas como los que se mencionan a continuación:

- Los mensajes SLBTR manejados en el BNC por su estructura flexible cambian con frecuencia a partir de las orientaciones del BCC. Un cambio en la estructura interna de uno o varios mensajes provocan retrasos de varias horas o días en los procesos de gestión de los departamentos encargados de

tramitar y enviar dichos mensajes, requiriendo de personal especializado que programe dichas funcionalidades para SABIC.

- El proceso de consultar los estados de los mensajes consume mucho tiempo al operador cuando usa SABIC, pues tiene que recurrir a un departamento que consulta en un sitio web externo que ofrece BCC o cargar otro sistema diferente del SABIC, que le permita usar un navegador web para dicha tarea.
- SABIC no ofrece la posibilidad de generar reportes personalizados sobre los mensajes SLBTR enviados y recibidos. Además de que no provee funcionalidades para consultar y controlar los mensajes registrados o enviados por cada operador, provocando que se tenga que incurrir en gasto de papel por concepto de impresión de los datos como resultado de la operación, para que el supervisor revise si los datos están correctos y mantenga un histórico de los mensajes enviados.
- Los datos que se envían y reciben en los mensajes SLBTR son muy sensibles. En ellos se manejan transferencias o pagos de grandes sumas dinero que no deben contener errores en su confección. El SABIC no ofrece una minuciosa validación de los datos de entrada, permite irregularidades que pueden ser detectadas meses después, afectando la contabilidad así como sus reportes.

Considerando lo analizado anteriormente, se define el siguiente **problema a resolver**: ¿Cómo reducir el tiempo y el consumo de recursos materiales cuando se requiere gestionar los mensajes SLBTR que intercambia el Banco Nacional de Cuba con el sistema bancario nacional?

Se presenta como **objeto de estudio**: la gestión de los mensajes SLBTR en entidades bancarias y como **campo de acción**: informatización de la gestión de los mensajes SLBTR en el Banco Nacional de Cuba.

Para darle solución al problema planteado se traza el siguiente **objetivo general**: Desarrollar un subsistema de Quarxo que reduzca el tiempo y el consumo de recursos en la gestión de los mensajes SLBTR que intercambia el Banco Nacional de Cuba con el sistema bancario nacional.

**Objetivos específicos**:

1. Elaborar el marco teórico de la investigación.
2. Realizar el análisis, el diseño y la implementación del subsistema Mensajería SLBTR.
3. Validar el subsistema Mensajería SLBTR.

**Tareas de la investigación:**

- Elaboración del estado del arte sobre los estándares de mensajería para SLBTR y las características de los sistemas informáticos desarrollados para SLBTR.
- Caracterización de los procesos relacionados con la gestión de mensajes SLBTR en el BNC.
- Caracterización de las metodologías, lenguajes y herramientas definidas para el sistema Quarxo.
- Caracterización de las tecnologías y patrones de diseño establecidos para el sistema Quarxo.
- Obtención de los artefactos correspondientes al análisis, diseño e implementación de la solución.
- Validación mediante la aplicación de pruebas unitarias al subsistema Mensajería SLBTR.

**Idea a defender**

Si se desarrolla el subsistema Mensajería SLBTR para el sistema Quarxo, se podrá reducir el tiempo y el consumo de recursos materiales, cuando se requiere gestionar los mensajes SLBTR que intercambia el Banco Nacional de Cuba con el sistema bancario nacional.

**Estructura del documento**

El presente documento cuenta con un resumen, un índice de contenidos, una introducción, tres capítulos, seguido de conclusiones, recomendaciones, una sección dedicada a las referencias bibliográficas y un glosario de términos.

**Capítulo 1. Fundamentación teórica**. Este capítulo comprende el estudio del estado del arte, brinda un breve acercamiento a los principales conceptos asociados al dominio del problema, además se argumenta sobre las herramientas, metodologías y tecnologías a utilizar.

**Capítulo 2. Análisis y diseño de la propuesta de solución**. En este capítulo se realizan los primeros flujos de trabajo propuestos por RUP para el desarrollo de la solución propuesta: Requisitos, Análisis y Diseño. Se especifican los patrones de arquitectura y de diseño utilizados en la confección de la solución. Para transitar de un flujo a otro se realiza la validación correspondiente.

**Capítulo 3. Implementación y validación de la solución**. En el presente capítulo se realizan las actividades y se obtienen los artefactos más importantes de los flujos de trabajo Implementación y Pruebas de la metodología RUP. Se expone la nomenclatura usada tanto en el código como en los paquetes y clases del subsistema, se describe cómo los elementos del modelo de diseño se implementan en términos de componentes y se abordan algunos temas importantes de la implementación, como la utilización del *framework* Spring Web Flow a través de la explicación de un escenario del sistema. Se realizan pruebas de caja blanca y caja negra, para evaluar la calidad del producto obtenido en la fase de implementación. Finalmente, se validan las variables de la investigación.

# CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA.

**Introducción**

El presente capítulo tiene como objetivo el análisis de los temas fundamentales que facilitarán la realización del subsistema SLBTR para el BNC. Se define qué es un SLBTR, se caracterizan los estándares de mensajería SLBTR y se analizan los sistemas informáticos SLBTR a nivel internacional y nacional. Se abordan las características de la metodología de desarrollo definida por el proyecto Quarxo, el lenguaje de programación, la descripción de los patrones que se emplearán, finalizando con la especificación de las herramientas y tecnologías que se utilizarán en el desarrollo. El conocimiento teórico obtenido con este capítulo permitirá el buen desempeño en los próximos capítulos del presente trabajo.

## 1.1. ¿Qué es un SLBTR?

Según el Banco Internacional de Pagos un SLBTR es un sistema en el que tanto el procesamiento y la liquidación final de las instrucciones de transferencia de fondos se realizan en tiempo real. Como se trata de un sistema de liquidación bruta, los pagos se liquidan de forma inmediata de una manera transaccional, es decir, operación por operación. Una vez procesadas las transacciones, los pagos son definitivos e irrevocables [6]. El término SLBTR se encuentra comúnmente en otras bibliografías como *Real Time Gross Settlement*, RTGS por sus siglas en inglés.

En la presente investigación se define SLBTR como un sistema electrónico que permite a los participantes transferir y recibir fondos en sus cuentas en un banco central en tiempo real.

**Características**

Un SLBTR, al ser un sistema de pago o liquidación, comprende un conjunto de instrumentos, reglas, y procedimientos bancarios que conectan a los participantes para transmitir los mensajes de pago entre sí. Además, están soportados por sistemas informáticos de transferencia de fondos interbancarios que garantizan la circulación del dinero.

**Funcionamiento de un SLBTR**

La mayoría de estos sistemas, los SLBTR, son propiedad de los bancos centrales y están basados en estructuras en forma de V, de esta forma solo existe interacción de los participantes con el banco central como centro único y viceversa, pero no entre los participantes directamente, además, es requerimiento del sistema que estos posean cuentas en el banco central. Este último, el banco central, es el encargado de

monitorear el flujo de información financiera (mensajes) en tiempo real, la cual está soportada en estándares de comunicación.

El término tiempo real contenido en el nombre del sistema, se refiere a que se liquidan los pagos en el banco central de forma inmediata de una manera transaccional, es decir, operación por operación, tan pronto como el mensaje SLBTR correspondiente a la operación de pago es recibido y validado por la aplicación SLBTR del banco central.

### 1.1.1. Los SLBTR a nivel internacional

El primer sistema automatizado de liquidación bruta en tiempo real fue el Fedwire de Estados Unidos, lanzado en 1970. Para fines de los ‘80 los SLBTR comienzan introducirse en Europa, estos sistemas fueron el CHAPS en Reino Unido (1984), el FA en Holanda (1985), RIX en Suecia (1986), SIC en suiza (1987), EIL-ZV en Alemania (1987), BISS en Italia (1989). El primer SLBTR de Asia fue el BOJ-NET en Japón (1988) [6]. En España desde 1996 entró funcionamiento el Sistema de Liquidación del Banco de España (SLBE), creado y gestionado su banco central [10].

En la Unión Europea existe un SLBTR para las naciones que la componen llamado TARGET2 (Sistema Transeuropeo de Pagos Brutos en Tiempo Real). Constituye la plataforma para grandes pagos en euros del Sistema Europeo de Bancos Centrales (SEBC). Se trata de un sistema descentralizado, basado en la interconexión de los sistemas de pago de cada país [20].

En América Latina, los bancos centrales de la región, adoptaron los SLBTR dentro de sus planes estratégicos a comienzos de la década del 90 [10]. Una muestra de ello es Colombia, México y Chile.

Colombia cuenta con un Sistema de Liquidación de Operaciones en Tiempo Real de Alto Valor, denominado SEBRA, que impacta directamente sobre las cuentas corrientes que tienen las instituciones financieras en el Banco de la República o Banco Central. En México se manejan las operaciones de liquidación bruta en tiempo real a través del SIPEI (Sistema de Pagos Electrónicos Interbancario) [5]. Por otro lado, el Banco Central de Chile inició sus operaciones usando SLBTR en 2004, permitiéndole procesar pagos electrónicos interbancarios, transacciones de cuenta por clientes y transacciones en el mercado de valores [4].

En los últimos años, en la mayoría de los países de niveles de ingresos medios y altos también se han registrado progresos importantes en la implementación de SLBTR.

### 1.1.2. El SLBTR del sistema bancario cubano

Cuba no ha estado al margen de este proceso, por lo que en 1997, debido a las perspectivas del desarrollo económico del país, se hizo imprescindible desagregar las funciones de banco central y

comercial que hasta ese momento desempeñó el BNC. Por lo tanto, se creó el BCC con la responsabilidad exclusiva de ejecutar funciones propias del banco central y capaz de contribuir con las transformaciones económicas y financieras que se llevan a cabo. Así, surge además el Banco de Crédito y Comercio (BANDEC) y se mantiene el BNC, pero con otras responsabilidades [13].

Acorde a las facultades que le otorga en el Decreto Ley 172 de 1997 del Consejo de Estado de la República de Cuba el BCC estableció en el año 2009 un SLBTR. Adaptado a las condiciones del sistema bancario nacional y motivado por la necesidad de que todo el movimiento de fondos en cuentas entre bancos cubanos se realizara a través de sus cuentas en el BCC. Además, ya en esa época era práctica internacional que los documentos que originan pagos en los bancos fueran tramitados por vías electrónicas, pues las transacciones interbancarias resultan vitales para la eficiencia y estabilidad del sistema bancario en su conjunto [3].

Este SLBTR persigue los siguientes objetivos [3]:

- Simplificar las transacciones interbancarias locales a través de brindar una vía única para su realización.
- Permitir la automatización de las transacciones interbancarias de extremo a extremo.
- Monitorear en el BCC los flujos financieros interbancarios y permitirle por esta vía tomar medidas, cuando sea necesario, para disminuir los riesgos de todo el sistema.
- Garantizar una mayor agilidad y eficiencia en los movimientos de fondos interbancarios.
- Asegurar la irrevocabilidad y precisión de los movimientos de fondos interbancarios, asentando las transacciones correspondientes en las cuentas en el BCC de las instituciones financieras involucradas.

**Funcionamiento**

El intercambio de información financiera que se realiza a través del SLBTR de Cuba se basa en los mensajes en un formato estándar defino por el BCC. Su funcionamiento general (ver Figura 1) se fundamenta en que cada participante tiene un único punto de contacto con el SLBTR en el BCC, desde donde se envían y reciben todos los mensajes a través de la red interbancaria. Los mensajes se intercambian en ambos sentidos entre las entidades participantes y el BCC, no estando permitido mensajes directos entre participantes. Todos los mensajes se autentican usando firmas electrónicas, de forma que el participante que envía el mensaje no pueda repudiarlo y sea imposible de modificar el contenido del mismo sin que se detecte. El SLBTR funciona las 24 horas del día, los siete días de la semana, y el cierre contable diario se realiza a las 16:00 horas de todos los días bancarios hábiles [57].

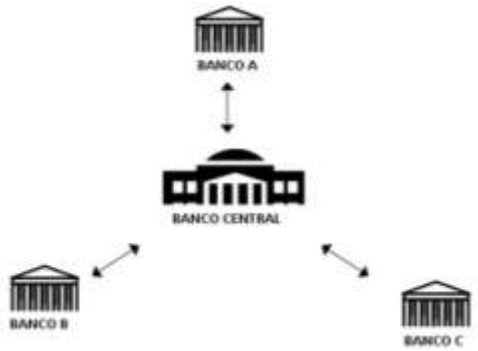



Figura 1. Esquema general de funcionamiento del SLBTR de Cuba [57]

### 1.2. ¿Qué es un mensaje SLBTR?

Según el Diccionario de la Real Academia de la Lengua Española un mensaje es el conjunto de señales, signos o símbolos que son objeto de una comunicación. En general, se puede definir como la información que un emisor envía a un receptor a través de un canal determinado o medio de comunicación.

A los mensajes utilizados en los SLBTR se les llaman mensajes SLBTR. Son elementos esenciales para la comunicación financiera entre las entidades participantes en los SLBTR. Están estructurados según el estándar de comunicación seleccionado para el SLBTR en el que son utilizados. Entre esos estándares o especificaciones se encuentran XBRL, SWIFT y SFMS.

### 1.3. Estándares de mensajería a nivel internacional

#### XBRL

XBRL (Lenguaje Extensible de Informes de Negocios) es un estándar abierto, sin costo por licencia o uso, basado en XML para la comunicación electrónica de datos de negocio y financieros. Nace de la propuesta lanzada en 1998 por Charles Hoffman, un experto contable y auditor, para simplificar la automatización del intercambio de información financiera mediante el uso del lenguaje XML, entonces emergente y hoy casi ubicuo en todo lo relacionado con Internet.

Es usado para generar y transmitir operaciones electrónicas y crear reportes en formato digital, personalizando las etiquetas y metadatos correspondientes a información de negocio de carácter financiero u operacional [67].

#### SWIFT

El estándar SWIFT fue desarrollado por la Sociedad para las Comunicaciones Interbancarias y Financieras Mundiales (SWIFT por sus siglas en inglés). Esta última, 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. Su primer paso fue el establecimiento de un lenguaje de mensajes sobre los pagos de clientes, transferencias financieras y reportes [34]. Esta compañía es una cooperativa propiedad de sus miembros, a través de la cual el sector financiero lleva a cabo sus operaciones comerciales de forma rápida, segura y fiable[62].

En los años 80, la SWIFT creó la norma ISO 7775, la cual define un catálogo de mensajes. Luego creó la norma ISO 15022 como una evolución de la anterior, teniendo como innovación fundamental la orientación al negocio. La más reciente es la ISO 20022, la cual se está adoptando internacionalmente. La estructura de los mensajes de esta última norma está basada en el formato XML, no así las dos normas anteriores que están en texto plano.

Los mensajes bajo el estándar SWIFT 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 la información que se utilizan para garantizar la seguridad de los datos del mensaje. El cuarto bloque es el más importante dentro del mensaje pues contiene la información financiera que se debe procesar.

Actualmente el estándar SWIFT es el más usando para la comunicación financiera a nivel internacional. En la mayoría de los SLBTR es utilizado a excepción de algunos países que crean su propio estándar.

**SFMS**

SFMS (*Structured Financial Messaging System*, en español Sistema Estructurado de Mensajería Financiera) es un estándar de mensajería segura desarrollado para servir como plataforma comunicación para aplicaciones en los bancos y entre los bancos. Fue lanzado el 14 de diciembre de 2001 por el Instituto para el Desarrollo e Investigación de Tecnología Bancaria (IDRBT) de la India. Es un estándar similar al SWIFT, pero con característica especiales para el sistema bancario de la India. Actualmente es utilizado en el SLBTR de dicho país.

SFMS es útil para aplicaciones de Transferencia Electrónica de Fondos (EFT), Sistema de Liquidación Bruta en Tiempo Real (SLBTR), de Entrega Contra Pago (DVP), o para Sistemas de Gestión Centralizada de Fondos (CFMS). Sigue la norma ISO 8583 que define un formato de mensaje y un flujo de comunicación para que sistemas diferentes puedan intercambiar transacciones [55].

### 1.4. El estándar de mensajería SLBTR de Cuba

Los mensajes que cumplen con el estándar SLBTR de Cuba hacen uso de XML. Este estándar está basado en la norma ISO 15022 del estándar SWIFT. El sistema informático que utiliza este formato es nombrado como SLBTR y los mensajes que utilizan este estándar se les llaman mensajes SLBTR [57].

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.

Los mensajes se agrupan en categorías de acuerdo con su primer dígito. Cada una de las categorías responde a operaciones financieras menos la última que es de propósito general. Por ejemplo:

1xx - Transferencias de clientes y cheques.
2xx - Transferencias de instituciones financieras.
3xx - Operaciones de cambio extranjero, préstamo/depósito y contratos precio convenido.
4xx - Remesas documentarias.
5xx - Valores.
6xx - Sindicaciones.
7xx - Créditos documentarios y garantías.
8xx - *Travellers* o cheques.
9xx - Mensajes de estados de las cuentas.

La estructura de los mensajes SLBTR está compuesta por:

***<MT_ número del mensaje xml: "MT_ número del mensaje.xml">***

***<ENCABEZADO>. . . </ENCABEZADO>***

***<TEXTO>. . . </TEXTO>***

***<SEGURIDAD>. . . </SEGURIDAD>***

***</MT_ número del mensaje>***

La etiqueta raíz ***<MT_ número del mensaje xml: "MT_ número del mensaje.xml">*** indica el tipo de mensaje. La etiqueta **<ENCABEZADO>...</ENCABEZADO>** se utiliza para identificar el mensaje. La etiqueta **<TEXTO>...</TEXTO>** contendrá los datos del mensaje y la etiqueta **<SEGURIDAD>...</SEGURIDAD>** tendrá información para asegurar los datos del mensaje [57].

## 1.4. Soluciones informáticas para la gestión de mensajería SLBTR

### 1.4.1. Soluciones a nivel internacional

**Montran RTGS**

El Montran RTGS está específicamente diseñado para satisfacer y cumplir las normas exigidas por los bancos centrales de hoy en día para el procedimiento de liquidación bruta en tiempo real. Maneja grandes volúmenes de información de pago de alto valor exigidas por los SLBTR a nivel mundial [31].

La solución está basada en tecnología web moderna, utilizando la arquitectura JEE y componentes combinados con RDBMS y seguridad basadas en PKI. Tiene una arquitectura abierta y utiliza estándares abiertos que simplifican la integración con otros sistemas financieros. Al estar basado en Java, puede funcionar en una amplia variedad de sistemas operativos, de *hardware* y bases de datos. Cuenta con una arquitectura que es independiente de la plataforma de *hardware* de se vaya a utilizar, siendo uno de los beneficios de la solución, tanto desde el punto de vista de costes y la fiabilidad [42].

Tiene como características el control completo de los saldos de las cuentas, tanto para el banco central y las instituciones participantes en el SLBTR. Posee una base de datos estadísticos con facilidades de consulta y elaboración de informes. Utiliza el estándar de mensaje SWIFT para gestionar procesamiento de las entradas de pago. Es un sistema multimoneda y multilenguaje [42].

**PAYplus RTGS**

Desarrollado por la empresa Fundtech, PAYplus RTGS, automatiza el procesamiento de pagos en los SLBTR. Utiliza el estándar SWIFT para la mensajería interbancaria. Entre las opciones que brinda está la posibilidad de hacer pagos multimonedas. El sistema permite a los bancos clientes maximizar los volúmenes de procesamiento y por lo tanto aumentar la productividad y reducir los costos de las operaciones de transferencia de fondos [23].

PAYplus RTGS es uno de los pocos sistemas que ha sido certificado en el mundo con la categoría de *SWIFT Ready Gold.* Se ha instalado con éxito en países como Australia, Grecia, Irlanda, Italia, Grecia, Sudáfrica, Suecia, el Reino Unido [46].

**STAR Liquidity Management**

Desarrollado por la empresa Software Integrators Ltd, está diseñado para satisfacer los requisitos de los bancos participantes en los sistemas de pago de alto valor nacional e internacional en donde se implante el sistema. Lo que permite supervisar y gestionar la liquidez de las operaciones con mayor eficacia mediante la programación de pagos de alto valor a intervalos regulares y controlables en los sistemas de

compensación SLBTR como el CHAPS de Reino Unido, el TARGET de la Unión Europea y el Fedwire de Estados Unidos [58].

Los mensajes gestionados en STAR Liquidity Management se basan en el estándar SWIFT. La empresa que provee esta solución no permite a los usuarios que compran esta solución el acceso al código fuente de la aplicación.

**Global Funds Exchange (GFX)**

Desarrollado por la empresa BankServ, es un *software* que automatiza el proceso de transferencias electrónicas en los SLBTR, permitiendo a los bancos el incremento de su capacidad operacional para ofrecer el mismo tipo de servicios integrales que eran, hasta ahora posibles en los grandes bancos que podían permitirse una gran inversión en equipos, *software* y de personal [9].

El sistema GFX se ejecuta sobre el sistema operativo Microsoft Windows y gestiona todos los pasos de una transferencia bancaria desde una sola aplicación e incluye varias características como la comprobación automática de la cuenta del cliente y los saldos antes de que el traslado se haga efectivo, también permite notificar al destinatario de pago y que la integración de las transacciones en el *software* de su banco sea acogida por la contabilidad [7].

**Perago RTGS**

Perago RTGS cumple con los principios establecidos por el BIS para los sistemas de pago de importancia sistémica, y ofrece su tecnología a los bancos del mundo. Es una solución totalmente compatible con el estándar SWIFT. El sistema está diseñado para integrarse con la red SWIFT o cualquier otra red privada. Su diseño garantiza la integridad de los datos, la seguridad y la confidencialidad. Puede ser configurado en una o más monedas. RTGS cuenta con un módulo de administración, un módulo de autorización y control de acceso, una interfaz de red y aplicaciones clientes para los bancos y los agentes de liquidación [48; 64].

### 1.4.2. Solución informática para el SLBTR de Cuba

La solución informática para el SLBTR de Cuba está divida en dos tipos de aplicaciones principales. Una es la aplicación SLBTR núcleo que se instala en el BCC y la otra es la aplicación SLBTR cliente que se instala en cada banco participante en el SLBTR.

La aplicación SLBTR núcleo y la SLBTR cliente están implementadas con los lenguajes de programación FoxPro y C# y utilizan como sistema gestor de base de datos Microsoft SQL Server 2005. Sus interfaces gráficas fueron implementadas en FoxPro de Microsoft, los componentes que procesan y contabilizan los datos del mensaje están desarrollados en procedimientos almacenados dentro el gestor Microsoft SQL

Server 2005, y el envío y recepción de los mensajes está programado en C# de la plataforma .NET de Microsoft. El entorno de ejecución de dichas aplicaciones es el sistema operativo MS-DOS [57].

Cada una de estas aplicaciones está asociada al sistema informático contable de cada entidad participante en el SLBTR. Dicho sistema informático contable es el que permite la gestión de los mensajes SLBTR bajo el estándar SLBTR de Cuba, y se vale de él para la obtención de los datos con los que se confeccionan los mensajes SLBTR.

SABIC es el sistema informático contable utilizado en la mayoría de los bancos de Cuba participantes en el SLBTR, por ejemplo en el BNC, que está implementado en FoxPro, utiliza el sistema gestor de base de datos Microsoft SQL Server 2005, y se ejecuta sobre el sistema operativo MS-DOS. Este posee un módulo para la gestión de los mensajes SLBTR que permite únicamente el registro y envío de mensajes SLBTR.

Tanto SABIC como las aplicaciones SLBTR núcleo y SLBTR cliente fueron desarrolladas por la Empresa de Servicios Informáticos para el Banco Central (SIBANC) de Cuba, que tiene la responsabilidad de administrar el funcionamiento informático del SLBTR cubano.

SIBANC para la consulta de los mensajes que se tramitan en el SLBTR ofrece una aplicación web desarrollada en ASPx y C#, disponible para todos los bancos participantes en la red interbancaria. A través dicha aplicación externa al banco participante se puede consultar los estados por los que transitan los mensajes SLBTR y hacer reportes acerca de estos.

### 1.4.3. Análisis de las soluciones informáticas estudiadas

Las soluciones informáticas internacionales antes estudiadas no utilizan ni implementan el estándar cubano de mensajería SLBTR además de que hay que pagar licencias por su manejo. Al ser aplicaciones privativas o de código cerrado es un riesgo su uso en cuanto seguridad se refiere, pues no proveen la documentación suficiente para un estudio completo de cómo fueron concebidas, independientemente de que puedan estar certificadas o no por organizaciones internacionales especializadas en los SLBTR. Poseen características deseables como son la multimoneda, el multilenguaje, la posibilidad de ejecutarse en varios sistemas operativos, la gestión completa de la mensajería SLBTR en lo relacionado con funcionalidades como confección, actualización, eliminación y envío. Son soluciones prácticas para SLBTR que implementan el estándar SWIFT, además, están adaptadas a normas que difieren de la política del sistema bancario cubano.

La solución informática cubana tiene como principal ventaja que implementa el estándar cubano de mensaje SLBTR y los bancos que la utilizan no tienen que pagar por su utilización. Permite en un nivel

muy básico la gestión de la mensajería, pues solo admite la confección y el envío de mensajes. Tiene como desventajas que se ejecuta solamente en sistemas operativos privativos como son MS-DOS y Microsoft Windows, además que está implementado en tecnología propietaria de Microsoft como son FoxPro, ASPx, C# y SQL Server 2005. El uso de FoxPro trae un riesgo de seguridad, pues la empresa Microsoft puso fin a su desarrollo y soporte técnico [41].

El módulo para la gestión de la mensajería SLBTR de SABIC carece de funcionalidades que permitan consultar los estados de los mensajes, realizar modificaciones antes de enviarlos, y realizar reportes acerca de los mismos. No cuenta con funcionalidades que permitan administrar los formatos de los mensajes en el estándar soportado, pues la gestión de estos se realiza modificando el código fuente en FoxPro, lo que equivale a la paralización del proceso de gestión de los mensajes definidos en un nuevo formato. La modificación del código fuente depende de especialistas, no del usuario final del módulo en el banco, por lo que es una característica deseable que el usuario final pueda modificar el formato de los mensajes sin tener que programar.

Se evidencia la necesidad de implementar un sistema informático web multiplataforma para la gestión de la mensajería SLBTR y la administración de los formatos asociados a dichos mensajes.

### 1.5. Metodología de desarrollo

Una metodología es una colección de procedimientos, técnicas, herramientas y documentos auxiliares que ayudan a los desarrolladores de *software* en sus esfuerzos por implementar nuevos sistemas de información [26]. Una metodología impone un proceso disciplinado sobre el desarrollo de *software* con el objetivo de hacerlo predecible y eficiente [1].

**Rational Unified Process (RUP)** es la metodología empleada para el desarrollo del sistema QUARXO. Seleccionada por la dirección del proyecto por su facilidad de adaptación y configuración a proyectos con un período de duración considerable y por ser una metodología robusta y bien definida, probada anteriormente en otros proyectos con características similares a Quarxo.

Entre sus principales características se encuentran las siguientes: está dirigida por casos de uso, es centrada en la arquitectura y además iterativo e incremental [35]. Incluye artefactos, que son los productos tangibles del proceso, y roles, que no es más que el papel que desempeña una persona en un determinado momento. RUP divide el proceso de desarrollo de *software* en cuatro fases dentro de las cuales se realizan tantas iteraciones como sean necesarias en dependencia de las características del proyecto. Para el desarrollo del presente trabajo es necesario apoyarse en las disciplinas de Análisis y Diseño, Implementación y Pruebas.

## 1.6. Lenguaje y herramienta de modelado

### UML

La metodología RUP propone utilizar el Lenguaje Unificado de Modelado (UML por sus siglas en inglés) para preparar todos los esquemas de un sistema de *software*. UML se define como un lenguaje que permite especificar, visualizar y construir los artefactos de los sistemas de *software*. Es un sistema notacional destinado a los sistemas de modelado que utilizan conceptos orientados a objetos [37].

Para el modelado UML existen varias aplicaciones, estas constituyen las herramientas CASE. Una herramienta CASE se define como un producto basado en computadora destinado a apoyar a una o más actividades de ingeniería de *software* en un proceso de desarrollo de *software* [16]. La principal ventaja de la utilización de estas herramientas es el aumento de la productividad y la mejora de la calidad del producto final.

### Visual Paradigm

Visual Paradigm en su versión 6.4 es una herramienta CASE multiplataforma, que permite el modelado UML, y soporta el ciclo de vida completo de RUP, permitiendo generar casi cualquier documento o diagrama. Puede integrarse con entornos integrados de desarrollo como el Eclipse y el NetBeans. Permite hacer diagramas UML, generación automática de código, sincronización entre modelos y código, entre otras posibilidades. Posee licencia comercial, con versión libre para la comunidad [65].

## 1.7. Arquitectura de *software*

Es uno de los elementos clave en todo proceso de desarrollo de software, facilita la comprensión de sistemas de *software* complejos, asimismo hace que aumenten las posibilidades de reutilizar tanto la arquitectura como los componentes que aparecen en ella. Sus principios básicos son cohesión, utilidad y flexibilidad de los componentes.

"La arquitectura del *software* es la organización fundamental de un sistema formada por sus componentes, las relaciones entre ellos y el contexto en el que se implantarán y los principios que orientan su diseño y evolución" [33].

## 1.8. Patrones

### 1.8.1. Patrones de arquitectura

Expresan un esquema fundamental de organización estructural para sistemas de *software*. Proveen subsistemas predefinidos, especificando sus responsabilidades, e incluyen reglas y guías para organizar las relaciones entre ello [8].

**Patrón Modelo Vista Controlador (MVC)**

El patrón MVC separa el modelo del dominio del negocio (clases de negocio), presentación y las acciones que se ejecutan sobre estas. 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 este. La vista, es la encargada de administrar la presentación de la información, y por último el controlador es el componente intermedio entre el modelo y la vista para lograr un desacoplamiento total entre los dos [21; 25; 56].

### 1.8.2. Patrones de diseño

Un patrón es una pareja problema-solución con un nombre y que es aplicable a otros contextos, con una sugerencia sobre la manera de usarlo en situaciones nuevas. Los patrones de diseño son la base para la búsqueda de soluciones a problemas comunes en el desarrollo de *software* y otros ámbitos referentes al diseño de interacción o interfaces [52; 56].

Con el constante perfeccionamiento de las arquitecturas de *software* se han definido algunos patrones de diseño como los que se mencionan a continuación:

**Patrón DAO**

Plantea utilizar un DAO (*Data Access Object* en inglés) para abstraer y encapsular todos los accesos a la fuente de datos. Maneja la conexión con la fuente de datos para obtener y almacenar datos, implementa el mecanismo de acceso requerido para trabajar con la fuente de datos y oculta completamente los detalles de implementación de la fuente de datos a sus clientes. Esencialmente, actúa como un adaptador entre el componente y la fuente de datos [25; 45].

**Patrones GRASP**

Los Patrones Generales para Asignar Responsabilidades: GRASP (en inglés: *General Responsibility Asignment Software Patterns*) describen los principios fundamentales de la asignación de responsabilidades a objetos, expresados en forma de patrones, su nombre se debe a la importancia de captar estos principios, si se quiere diseñar eficazmente el *software* orientado a objetos. GRASP destaca 5 patrones principales: Experto, Creador, Bajo acoplamiento, Alta cohesión, Controlador [37; 56].

- **Patrón Experto:** Este patrón consiste en asignar una responsabilidad al experto en información, es decir, a la clase que cuenta con la información necesaria para cumplir con la responsabilidad. Se refuerza el encapsulamiento y se favorece el bajo acoplamiento.

- **Patrón Creador:** El patrón guía la asignación de responsabilidades relacionadas con la creación de objetos, tarea muy frecuente en los sistemas orientados a objetos. Su propósito principal es encontrar un creador que debemos conectar con el objeto producido en cualquier evento. Brinda soporte al bajo acoplamiento.
- **Patrón Bajo Acoplamiento:** Pretende asignar una responsabilidad para mantener el bajo acoplamiento, es decir, el diseño de clases más independientes, que no se relacionen con muchas otras, que reducen el impacto de los cambios, que son más reutilizables y acrecienten la oportunidad de una mayor productividad.
- **Patrón Alta Cohesión:** Su objetivo es asignar una responsabilidad, de modo que la cohesión siga siendo alta. Las clases con alta cohesión se caracterizan por tener responsabilidades estrechamente relacionadas y no realizar un trabajo enorme. Una clase con alta cohesión es útil porque es bastante fácil darle mantenimiento, entenderla y reutilizarla. Es un patrón evaluativo que el desarrollador aplica al valorar sus decisiones de diseño.
- **Patrón Controlador:** Consiste en asignar la responsabilidad a una clase de manejar los mensajes correspondientes a eventos de un sistema. El controlador es un intermediario entre la interfaz de usuario y el núcleo de las clases donde reside la lógica de la aplicación. Este patrón sugiere que la lógica de negocios debe estar separada de la capa de presentación para aumentar la reutilización de código y a la vez tener un mayor control.

**Patrones GOF [25; 56].**

- **Patrón Fachada:** Patrón estructural que trata de simplificar la interfaz entre dos sistemas o componentes de *software* ocultando un sistema complejo detrás de una clase que funciona como pantalla o fachada. La idea principal es la de ocultar todo lo posible la complejidad de un sistema, el conjunto de clases o componentes que lo definen, de forma que solo se ofrezca un punto de entrada al sistema cubierto por la fachada. Su uso aísla los posibles cambios que se puedan producir en alguna de las partes.
- **Patrón Mediador:** Patrón de comportamiento que coordina las relaciones entre sus asociados. Permite la interacción de varios objetos, sin generar acoples fuertes en esas relaciones.
- **Patrón Cadena de Responsabilidad:** La cadena de responsabilidad se encarga de evitar el acoplamiento del remitente de una petición a su receptor, dando a más de un objeto la posibilidad de manejar la petición.

- **Patrón *Singleton*:** Patrón creacional diseñado para restringir la creación de objetos pertenecientes a una clase. Su objetivo es garantizar que una determinada clase solo tenga una instancia y proporcionar un punto de acceso global a esta. Es un patrón muy sencillo de diseñar y a menudo es implementado por otros patrones.

**Patrones de presentación**

Los patrones de presentación surgen con el objetivo de brindar un mecanismo que controle y provea mejores prácticas a la hora de llevar a cabo el diseño de la presentación en un proyecto de *software*.

- **Patrón *Composite View* (Vistas Compuestas):** Este patrón recomienda si la vista es muy compleja formarla a partir de la inclusión de otras más pequeñas y específicas. Por ejemplo, una página JSP puede incluir otras que se especialicen en tareas muy particulares [44].

## 1.9. Ambiente de desarrollo

### 1.9.1. Lenguaje de programación en el lado del servidor

**Java**

Java es un lenguaje de programación orientado a objetos, que permite a los programadores realizar aplicaciones de múltiples tipos, ya sean de escritorio o web. Se caracteriza por ser un lenguaje simple, robusto y poderoso que se torna fácil de aprender, debido a que elimina sentencias de bajo nivel además cuenta con un *garbage collector* (en español: recolector de basura), haciendo transparente para los programadores el manejo de la memoria. Se destaca por ser un lenguaje de código abierto, multiplataforma por lo cual ha logrado una gran expansión por todo el mundo. En la actualidad incluye un gran número de bibliotecas para múltiples trabajos como: el trabajo con la red, tratamiento de excepciones, hilos para el procesamiento concurrente, entre otras [61].

**Plataforma JEE**

JEE (en inglés: *Java Enterprise Edition*) es una plataforma de programación, que define estándares para desarrollar y ejecutar aplicaciones en el lenguaje de programación Java, empleando arquitecturas que definen un modelo multicapa y que se apoyan en componentes de *software* modulares. JEE incluye tecnologías, tales como Servlets, JSP (en inglés: *Java Server Pages*) y varias tecnologías de servicios web. Las aplicaciones desarrolladas en esta plataforma tienden a ser portables, escalables, robustas y seguras a la vez que son integrables con tecnologías anteriores [61].

### 1.9.2. Lenguajes en el lado del cliente

En el caso de los lenguajes que sustentan la aplicación frente al cliente, se definieron por parte del proyecto el uso de:

**XHTML**

XHTML (*Extensible Hypertext Markup Language* en inglés) es el lenguaje de marcado pensado para sustituir a HTML (*Hypertext Markup Language* en inglés), es una reformulación del mismo que es compatible con XML (*Extensible Markup Language* en inglés) Se utiliza para generar documentos y contenidos de hipertexto generalmente publicados en la WEB. El documento se escribe en forma de etiquetas, que hasta cierto punto pueden establecer la apariencia del documento, aunque en la actualidad se suele mezclar con lenguajes *script* como JavaScript, para lograr mayor interacción con los usuarios[66].

**JavaScript**

JavaScript es un lenguaje de programación que se utiliza principalmente para crear páginas web dinámicas. Técnicamente, JavaScript es un lenguaje de programación interpretado, por lo que no es necesario compilar los programas para ejecutarlos. En otras palabras, los programas escritos con JavaScript se pueden probar directamente en cualquier navegador sin necesidad de procesos intermedios[22].

### 1.9.3. Framework

Un *framework* es una estructura de soporte definida en la cual otro proyecto de *software* puede ser organizado y desarrollado. Típicamente, puede incluir soporte de programas, bibliotecas y un lenguaje interpretado entre otros *software* para ayudar a desarrollar y unir los diferentes componentes de un proyecto [36].

El equipo de arquitectura decidió utilizar los que se caracterizan a continuación:

**Spring Framework**

Es un *framework* bajo licencia de código abierto concebido para el desarrollo de aplicaciones basadas en la plataforma JEE. Provee un modelo consistente de programación, ayuda a promover la reusabilidad de código, facilita el diseño orientado a objetos en aplicaciones JEE [36; 60]. Se hace uso de Spring Framework en su versión 2.5.

**Spring Web Flow**

Spring Web Flow es un subproyecto de Spring Framework que permite operar la navegación de una aplicación web, para lo cual proporciona la definición de un lenguaje declarativo de flujos. Surge como

solución a las limitaciones del flujo de página de los *frameworks* MVC clásicos. Se utiliza cuando el caso de uso que se desarrolla presenta un flujo complejo y/o no lineal. La configuración del desarrollo del flujo es especificada a través de un archivo de configuración XML, permitiendo establecer reglas de navegación complejas de una manera sencilla. Un flujo en Spring Web Flow maneja la conversación (vacío entre sesión y petición) completa, de inicio a fin, punto en el que limpia la memoria automáticamente liberando al usuario de la responsabilidad de borrar los recursos en memoria que no estén siendo utilizado [30]. Se hace uso de Spring Web Flow en su versión 2.0.8.

**Hibernate Framework**

Es un framework ORM y un generador de sentencias SQL. Hibernate es una herramienta para la plataforma Java que facilita el mapeo de atributos entre una base de datos relacional y el modelo de objetos de una aplicación, mediante archivos declarativos XML que permiten establecer estas relaciones. Permite diseñar objetos persistentes que podrán incluir polimorfismo, relaciones, colecciones, y un gran número de tipos de datos. Soporta diferentes entornos, por ejemplo: MySQL, Oracle, Microsoft SQL Server. Se adapta a cualquier proceso de desarrollo, ya sea comenzando un diseño desde cero o trabajando con una base de datos existente, y apoyará cualquier arquitectura de aplicaciones [29]. Se hace uso de Hibernate Framework en su versión 3.5.

**Dojo Toolkit**

Unifica estándares de codificación resolviendo los problemas de compatibilidad entre los navegadores. Incluye un gran cúmulo de componentes visuales basados en XHTML, CSS y JavaScript para enriquecer la interfaz de usuario. Se destaca por permitir desarrollar aplicaciones que pueden funcionar con independencia del navegador web donde se ejecuten, hace transparente el desarrollo para diferentes implementaciones del DOM, ofrece soporte para la internacionalización. Presenta una arquitectura modular donde destacan los módulos: Dojo, Dijit y Dojox [17; 54]. Se hace uso de Dojo Toolkit en su versión 1.3.

### 1.9.4. Herramientas de desarrollo

**Control de versiones**

Se le denomina control de versiones a la gestión de cambios que sufre los componentes o elementos de un producto o configuraciones del mismo. Un sistema de control de versiones debe poseer un mecanismo donde se pueda controlar el registro histórico de los cambios que se realicen a cada elemento además de los medios de almacenaje [49].

**Subversion**

Para el control de las versiones del sistema se utiliza el Subversion 1.6.6, que está desarrollado sobre tecnología de código abierto y se integra con el IDE Eclipse mediante el *plugin* SubEclipse. Es un sistema de control de versiones libre y de código abierto conocido habitualmente como SVN que se ha expandido en gran medida dentro de la comunidad del desarrollo de *software*. Permite recuperar versiones antiguas de los datos registrados y examinar el historial de los mismos, manejando la información de ficheros y directorios a través del tiempo, la cual radica en un árbol de ficheros en un repositorio central [39; 49].

**Entorno de desarrollo integrado: Eclipse**

Se selecciona como entorno de desarrollo integrado Eclipse en su versión 3.3. Esta herramienta es multiplataforma, de código abierto y presenta una arquitectura de *plugins* que lo hace ser bastante flexible y configurable. La arquitectura de *plugins* permite integrar diversos lenguajes, introducir otras aplicaciones accesorias que pueden resultar útiles durante el proceso de desarrollo tales como: herramientas UML, editores visuales de interfaces, ayuda en línea para librerías, entre otros [18; 43].

**Servidor de aplicaciones web: Apache Tomcat**

El Apache Tomcat es un *software* de código abierto implementado para las tecnologías Java Servlet y Java Server Pages. Apache Tomcat es desarrollado en un entorno abierto, participativo y publicado bajo la licencia del *software* de Apache que presenta la ventaja de ser multiplataforma y funciona como servidor web autónomo en entornos con alto nivel de tráfico y alta disponibilidad [14]. Se hace uso de Apache Tomcat en su versión 6.

**Servidor de Base de Datos: Microsoft SQL Server 2005**

Microsoft SQL Server 2005 es un servidor de base de datos basado en el modelo relacional. Se caracteriza por brindar soporte para transacciones y procedimientos almacenados y posee como lenguajes de consulta al SQL y al T-SQL (en inglés: *Transact-SQL*) permite además trabajar en modo cliente-servidor y promueve la escalabilidad y seguridad de la información [15].

**Conclusiones parciales**

- Se concluye que no es factible costear cualquiera de los sistemas informáticos ya existentes a nivel internacional para la gestión de la mensajería SLBTR, pues no implementan el estándar cubano de mensaje SLBTR, además, no están en consonancia con la política bancaria cubana.
- El sistema informático nacional para la gestión de mensajería SLBTR aunque implementa el estándar cubano de mensaje SLBTR y los bancos que lo utilizan no tienen que pagar por su

utilización; permite en un nivel muy básico la gestión de la mensajería, admitiendo solamente operaciones de confección y envío. No cuenta con funcionalidades que permitan administrar los formatos de los mensajes en el estándar soportado.

- Se evidencia la necesidad de implementar un sistema informático web multiplataforma para la gestión de la mensajería SLBTR y la administración de los formatos asociados a dichos mensajes.
- En correspondencia con lo definido por el proyecto Quarxo se utilizará RUP como metodología de desarrollo del *software*, UML como lenguaje de modelado, Visual Paradigm como herramienta CASE, Java (en su especificación JEE) como lenguaje de programación, Spring como framework núcleo de la aplicación apoyado por los *frameworks* Hibernate, para la persistencia de datos, y Dojo Toolkit, para enriquecer las interfaces que interactúan con el usuario; Eclipse como herramienta de desarrollo, SQL Server 2005 como gestor de base de datos, y como contenedor web el Apache Tomcat.

# CAPÍTULO 2: ANÁLISIS Y DISEÑO DE LA PROPUESTA DE SOLUCIÓN

**Introducción**

En este capítulo se realizan los primeros flujos de trabajo propuestos por RUP para el desarrollo de la solución propuesta: Requisitos, Análisis y Diseño. En la etapa de Requisitos se identifican y describen los requisitos funcionales y no funcionales, se construye el modelo de casos de uso del subsistema propuesto. En el Análisis y el Diseño, se definen todos los artefactos propuestos por RUP para cada caso, destacando el modelo de análisis y modelo de diseño, y la descripción de la arquitectura como base para el desarrollo de este tipo de aplicación. Igualmente, se especifican los patrones de arquitectura y de diseño utilizados en la concepción de la solución. Para transitar de un flujo a otro se realiza la validación correspondiente.

## 2.1. Comprensión del contexto del sistema. Modelo de dominio

Existen por lo menos dos aproximaciones para expresar el contexto de un sistema en una forma utilizable para los desarrolladores de *software*: modelado del dominio y modelado del negocio [35]. En la presente investigación se seleccionó el modelado del dominio, pues el contexto del sistema a desarrollar no es muy complejo.

Un modelo de dominio describe los conceptos importantes del contexto como objetos del dominio, y enlaza estos objetos unos con otros. La identificación y la asignación de un nombre para estos objetos ayudan a desarrollar un glosario de términos que permitirán comunicarse mejor a todos los que están trabajando en el sistema [35].

El modelo de dominio se describe mediante diagramas de UML (especialmente mediante diagramas de clases). Estos diagramas muestran a los clientes, usuarios, revisores y a otros desarrolladores las clases del dominio y cómo se relacionan unas con otras mediante asociaciones.

A continuación se modela el dominio del sistema, como paso previo a la definición de los requerimientos y teniendo como motivación la ayuda que ofrece para comprender los conceptos claves del ámbito del problema a resolver y porque disminuye la brecha entre cómo ven los clientes el problema y la representación en software de la solución, usando un modelado orientado a objetos.

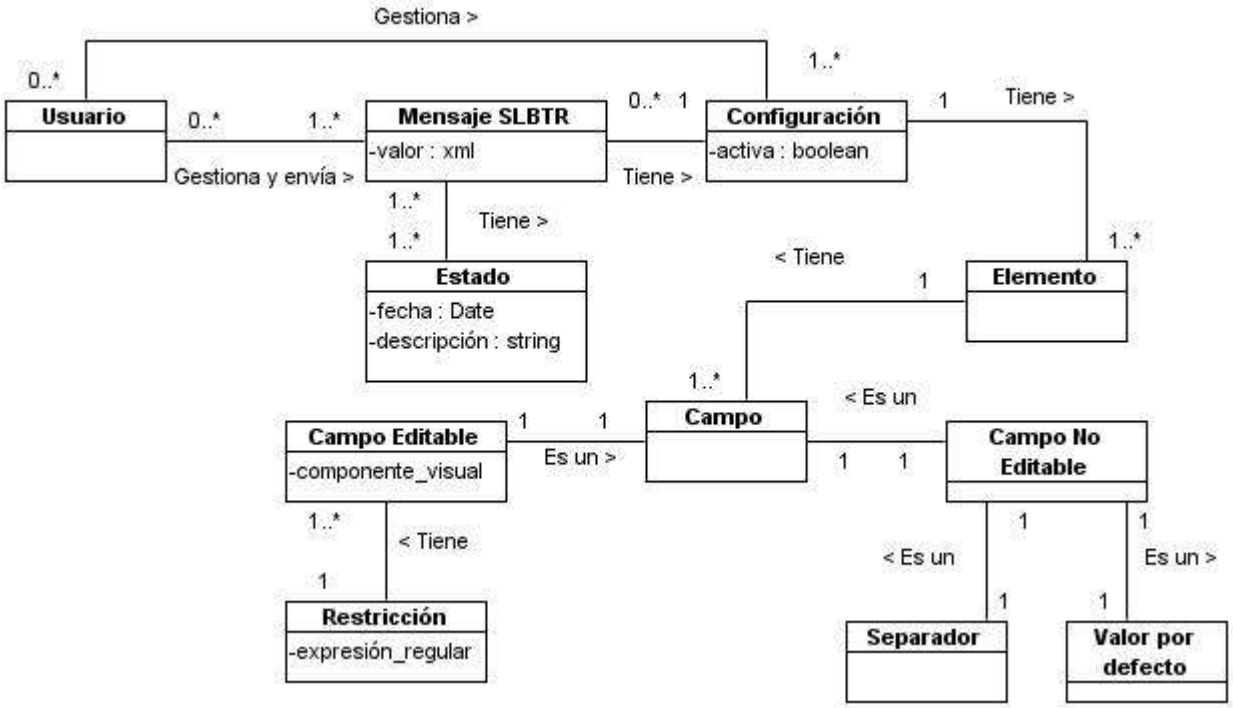



Figura 2 Modelo de dominio

## 2.2. Requisitos

Esta disciplina o flujo de trabajo de RUP tiene como propósito fundamental guiar el desarrollo hacia un sistema correcto. Eso se consigue mediante una descripción de los requisitos del sistema suficientemente buena como para que pueda llegarse a un acuerdo entre el cliente y los desarrolladores sobre qué debe y qué no debe hacer el sistema [35].

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 [32].

### 2.1.1. Requisitos funcionales

Los requisitos funcionales (RF) son declaraciones de los servicios que debe proporcionar el sistema, de la manera en que este debe reaccionar a entradas particulares y de cómo se debe comportar en situaciones particulares. En algunos casos, los requerimientos funcionales de los sistemas también pueden declarar explícitamente lo que el sistema no debe hacer [59].

Los requisitos funcionales asociados con la propuesta de solución son:

**RF1. Gestionar mensaje SLBTR MT199 con código de solicitud 06.**

El sistema permitirá registrar, buscar, actualizar, consultar, enviar y eliminar mensajes SLBTR MT199 con código de solicitud 06 **(Mensaje SLBTR de solicitud de reembolso por asignación contra recursos propios por los bancos a Sucursal de Cuenta Única (SCU))**

**RF1.1.** Registrar mensaje SLBTR MT199 con código de solicitud 06.

**RF1.2**. Buscar mensaje SLBTR MT199 con código de solicitud 06.

**RF1.3**. Actualizar mensaje SLBTR MT199 con código de solicitud 06.

**RF1.4.** Consultar mensaje SLBTR MT199 con código de solicitud 06.

**RF1.5**. Enviar mensaje SLBTR MT199 con código de solicitud 06.

**RF1.6.** Eliminar mensaje SLBTR MT199 con código de solicitud 06.

**RF2. Gestionar mensaje SLBTR MT199 con código de solicitud 10.**

El sistema permitirá registrar, buscar, actualizar, consultar, enviar y eliminar mensajes SLBTR MT199 con código de solicitud 10 **(Mensaje SLBTR de solicitud de reembolso de comisiones y gastos por los bancos a SCU)**

**RF2.1.** Registrar mensaje SLBTR MT199 con código de solicitud 10.

**RF2.2**. Buscar mensaje SLBTR MT199 con código de solicitud 10.

**RF2.3**. Actualizar mensaje SLBTR MT199 con código de solicitud 10.

**RF2.4.** Consultar mensaje SLBTR MT199 con código de solicitud 10.

**RF2.5**. Enviar mensaje SLBTR MT199 con código de solicitud 10.

**RF2.6.** Eliminar mensaje SLBTR MT199 con código de solicitud 10.

**RF3. Gestionar mensaje SLBTR MT199 con código de solicitud 13.**

El sistema permitirá registrar, buscar, actualizar, consultar, enviar y eliminar mensajes SLBTR MT199 con código de solicitud 13 **(Mensaje SLBTR de solicitud de reembolso contra garantías por los bancos a SCU)**

**RF3.1.** Registrar mensaje SLBTR MT199 con código de solicitud 13.

**RF3.2**. Buscar mensaje SLBTR MT199 con código de solicitud 13.

**RF3.3**. Actualizar mensaje SLBTR MT199 con código de solicitud 13.

**RF3.4.** Consultar mensaje SLBTR MT199 con código de solicitud 13.

**RF3.5**. Enviar mensaje SLBTR MT199 con código de solicitud 13.

**RF3.6.** Eliminar mensaje SLBTR MT199 con código de solicitud 13.

**RF4. Gestionar mensaje SLBTR MT199 con código de solicitud 21.**

El sistema permitirá registrar, buscar, actualizar, consultar, enviar y eliminar mensajes SLBTR MT199 con código de solicitud 21 **(Mensaje SLBTR de apertura del instrumento de pago por los bancos a SCU)**

**RF4.1.** Registrar mensaje SLBTR MT199 con código de solicitud 21.

**RF4.2**. Buscar mensaje SLBTR MT199 con código de solicitud 21.

**RF4.3**. Actualizar mensaje SLBTR MT199 con código de solicitud 21.

**RF4.4.** Consultar mensaje SLBTR MT199 con código de solicitud 21.

**RF4.5**. Enviar mensaje SLBTR MT199 con código de solicitud 21.

**RF4.6.** Eliminar mensaje SLBTR MT199 con código de solicitud 21.

**RF5. Gestionar mensaje SLBTR MT199 con código de solicitud 22.**

El sistema permitirá registrar, buscar, actualizar, consultar, enviar y eliminar mensajes SLBTR MT199 con código de solicitud 22 **(Mensaje SLBTR de apertura del instrumento de pago por los bancos a SCU)**

**RF5.1.** Registrar mensaje SLBTR MT199 con código de solicitud 22.

**RF5.2**. Buscar mensaje SLBTR MT199 con código de solicitud 22.

**RF5.3**. Actualizar mensaje SLBTR MT199 con código de solicitud 22.

**RF5.4.** Consultar mensaje SLBTR MT199 con código de solicitud 22.

**RF5.5**. Enviar mensaje SLBTR MT199 con código de solicitud 22.

**RF5.6.** Eliminar mensaje SLBTR MT199 con código de solicitud 22.

**RF6. Gestionar mensaje SLBTR MT199 con código de solicitud 23.**

El sistema permitirá registrar, buscar, actualizar, consultar, enviar y eliminar mensajes SLBTR MT199 con código de solicitud 23 **(Mensaje SLBTR de reclamación del reembolso del principal más intereses por los bancos a SCU)**

**RF6.1.** Registrar mensaje SLBTR MT199 con código de solicitud 23.

**RF6.2**. Buscar mensaje SLBTR MT199 con código de solicitud 23.

**RF6.3**. Actualizar mensaje SLBTR MT199 con código de solicitud 23.

**RF6.4.** Consultar mensaje SLBTR MT199 con código de solicitud 23.

**RF6.5**. Enviar mensaje SLBTR MT199 con código de solicitud 23.

**RF6.6.** Eliminar mensaje SLBTR MT199 con código de solicitud 23.

**RF7. Gestionar mensaje SLBTR MT199 con código de solicitud 24.**

El sistema permitirá registrar, buscar, actualizar, consultar, enviar y eliminar mensajes SLBTR MT199 con código de solicitud 24 **(Mensaje SLBTR de solicitud de reembolso por los pagos efectuados por los bancos a SCU)**

**RF7.1.** Registrar mensaje SLBTR MT199 con código de solicitud 24.

**RF7.2**. Buscar mensaje SLBTR MT199 con código de solicitud 24.

**RF7.3**. Actualizar mensaje SLBTR MT199 con código de solicitud 24.

**RF7.4.** Consultar mensaje SLBTR MT199 con código de solicitud 24.

**RF7.5**. Enviar mensaje SLBTR MT199 con código de solicitud 24.

**RF7.6.** Eliminar mensaje SLBTR MT199 con código de solicitud 24.

**RF8. Gestionar mensaje SLBTR MT199 con código de solicitud 27.**

El sistema permitirá registrar, buscar, actualizar, consultar, enviar y eliminar mensajes SLBTR MT199 con código de solicitud 28 **(Mensaje SLBTR de solicitud de reembolso por principal de renegociación por los bancos a SCU)**

**RF8.1.** Registrar mensaje SLBTR MT199 con código de solicitud 27.

**RF8.2**. Buscar mensaje SLBTR MT199 con código de solicitud 27.

**RF8.3**. Actualizar mensaje SLBTR MT199 con código de solicitud 27.

**RF8.4.** Consultar mensaje SLBTR MT199 con código de solicitud 27.

**RF8.5**. Enviar mensaje SLBTR MT199 con código de solicitud 27.

**RF8.6.** Eliminar mensaje SLBTR MT199 con código de solicitud 27.

**RF9. Gestionar mensaje SLBTR MT199 con código de solicitud 28.**

El sistema permitirá registrar, buscar, actualizar, consultar, enviar y eliminar mensajes SLBTR MT199 con código de solicitud 28 **(Mensaje SLBTR de solicitud de reembolso por intereses de renegociación por los bancos a SCU)**

**RF9.1.** Registrar mensaje SLBTR MT199 con código de solicitud 28.

**RF9.2**. Buscar mensaje SLBTR MT199 con código de solicitud 28.

**RF9.3**. Actualizar mensaje SLBTR MT199 con código de solicitud 28.

**RF9.4.** Consultar mensaje SLBTR MT199 con código de solicitud 28.

**RF9.5**. Enviar mensaje SLBTR MT199 con código de solicitud 28.

**RF9.6.** Eliminar mensaje SLBTR MT199 con código de solicitud 28

**RF10. Buscar mensaje SLBTR enviado.**

**RF11. Consultar mensajes SLBTR enviado.**

**RF12. Buscar mensaje SLBTR recibido**

**RF13. Consultar mensajes SLBTR recibido.**

**RF14. Gestionar configuración de mensaje SLBTR.**

**RF14.1.** Registrar configuración de mensaje SLBTR.

**RF14.2.** Buscar configuración de mensaje SLBTR.

**RF14.3.** Actualizar configuración de mensaje SLBTR.

**RF14.4**. Consultar configuración de mensaje SLBTR.

**RF14.5.** Cambiar estado de configuración de mensaje SLBTR.

**RF14.6.** Eliminar configuración de mensaje SLBTR.

**RF15. Gestionar componente de configuración de mensaje SLBTR**.

**RF15.1.** Registrar componente de configuración de mensaje SLBTR.

**RF15.2.** Buscar componente de configuración de mensaje SLBTR.

**RF15.3.** Actualizar componente de configuración de mensaje SLBTR.

**RF15.4**. Consultar componente de configuración de mensaje SLBTR.

**RF15.5.** Eliminar componente de configuración de mensaje SLBTR.

**RF16. Gestionar expresión regular asociada a componente de configuración de mensaje SLBTR**.

**RF16.1.** Registrar expresión regular asociada a componente de configuración de mensaje SLBTR.

**RF16.2.** Buscar expresión regular asociada a componente de configuración de mensaje SLBTR.

**RF16.3.** Actualizar expresión regular asociada a componente de configuración de mensaje SLBTR.

**RF16.4**. Consultar expresión regular asociada a componente de configuración de mensaje SLBTR.

**RF16.5.** Eliminar expresión regular asociada a componente de configuración de mensaje SLBTR.

### 2.1.2. Requisitos no funcionales

Los requisitos no funcionales (RNF) son aquellos requerimientos que no se refieren directamente a las funciones específicas que entrega el sistema, sino a las propiedades emergentes de este como la fiabilidad, la respuesta en el tiempo y la capacidad de almacenamiento. Definen propiedades y restricciones del sistema [59]. De acuerdo con las características sobre las cuales se desarrolla este subsistema se toman en cuenta los siguientes RNF: (consultar Anexo # 2**).**

### 2.1.3. Modelo de caso de uso

Un modelo de caso de uso es un modelo del sistema que contiene actores, casos de usos y sus relaciones[35].

**Actores.**

| Actor | Descripción |
|---|---|
| **Usuario** | Operador del BNC encargado de la gestión y envío de los mensajes SLBTR. |

#### 2.1.3.1. Diagrama de casos de uso del sistema (DCUS)

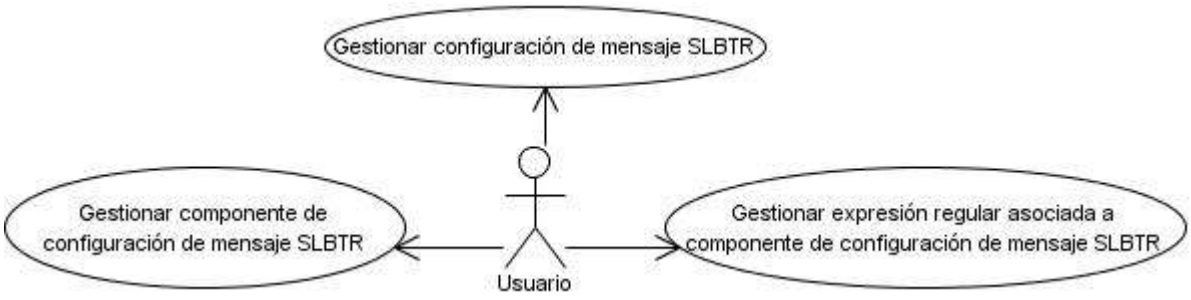



Figura 3 DCUS asociados a la gestión de las configuraciones de mensajes SLBTR

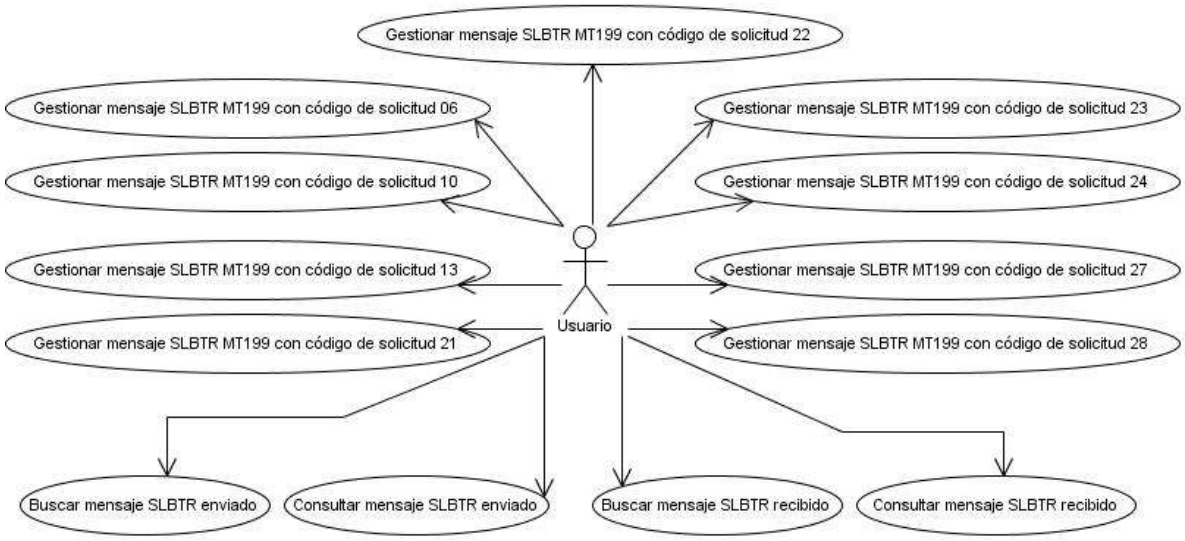



Figura 4 DCUS asociados a la gestión de mensajes SLBTR

### 2.1.3.2. Descripción de los casos de uso del sistema

La forma en que los actores usan el sistema se representa mediante un caso de uso. Los casos de usos son fragmentos de funcionalidad que el sistema brinda para contribuir con un resultado de valor para sus actores. Un caso de uso (CU) describe una secuencia de acciones que el sistema puede llevar a cabo interactuando con sus actores, incluyendo alternativas dentro de la secuencia [35].

A continuación se describe uno de los CUS más significativos: el CU Gestionar mensaje SLBTR MT199 con código de solicitud 21. Las restantes descripciones se pueden consultar en el Anexo # 3.

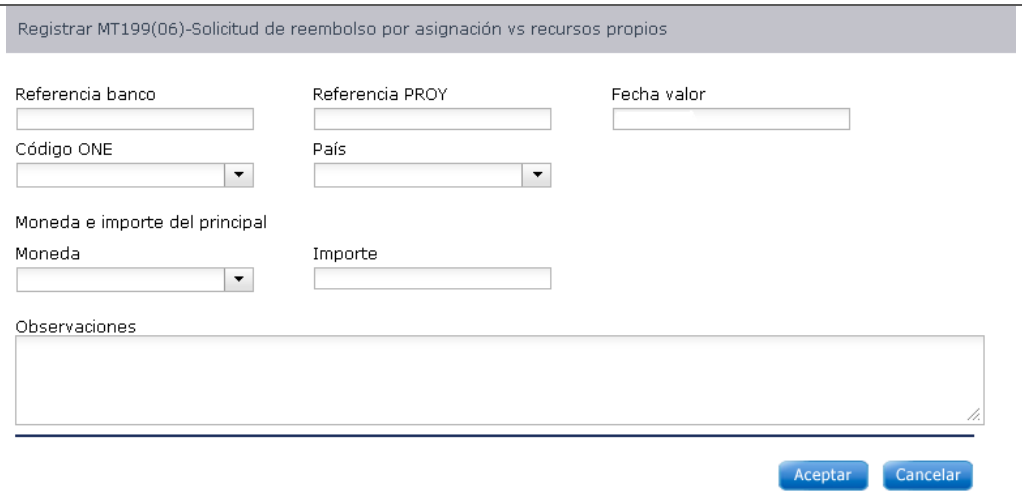
Tabla 1 Descripción del CUS Gestionar mensaje SLBTR MT199 con código de solicitud 21

| **Caso de uso:** | **Gestionar mensaje SLBTR MT199 con código de solicitud 21** |
|---|---|
| **Actores:** | Usuario |
| **Resumen:** | El caso de uso se inicia cuando el usuario selecciona la opción “Gestionar mensaje SLBTR MT199 con código de solicitud 21”. El sistema registra, actualiza, consulta, elimina, envía y permite realizar búsquedas de mensajes SLBTR con código de solicitud 21. El caso de uso termina cuando el usuario acepta realizar la operación. |
| **Precondiciones:** | • El usuario debe estar autenticado con los permisos necesarios. |
| **Referencias** | RF 4.1, RF 4.2, RF 4.3, RF 4.4, RF 4.5, RF 4.6 |
| **Prioridad** | Alta |
| **Flujo normal de eventos** | |
| **Sección “Gestionar configuración de mensaje SLBTR”** | |
| **Acción del actor** | **Respuesta del sistema** |
| 1. El usuario selecciona la opción “Gestionar mensaje SLBTR MT199 con código de solicitud 21”. | 2. El sistema muestra las opciones de registrar, actualizar, consultar, eliminar, enviar y buscar mensajes SLBTR con código de solicitud 21 |
| 3. El usuario puede seleccionar una de las opciones siguientes: | |

| | |
|---|---|
| a. Registrar mensaje SLBTR con código solicitud 21.<br>Ver sección “Registrar mensaje SLBTR con código solicitud 21”.<br>b. Actualizar mensaje SLBTR con código solicitud 21.<br>Ver sección “Actualizar mensaje SLBTR con código solicitud 21”. (Anexo # 3)<br>c. Consultar mensaje SLBTR con código solicitud 21.<br>Ver sección “Consultar mensaje SLBTR con código solicitud 21”. (Anexo # 3)<br>d. Eliminar mensaje SLBTR con código solicitud 21.<br>Ver sección “Eliminar mensaje SLBTR con código solicitud 21”. (Anexo # 3)<br>e. Buscar mensaje SLBTR con código solicitud 21.<br>Ver sección “Buscar mensaje SLBTR con código solicitud 21”. (Anexo # 3)<br>f. Enviar mensaje SLBTR con código solicitud 21.<br>Ver sección “Enviar mensaje SLBTR con código solicitud 21”. (Anexo # 3) | |

| **Sección “Registrar mensaje SLBTR con código solicitud 21”** | |
|---|---|
| **Acción del actor** | **Respuesta del sistema** |
| | 1. El sistema carga la interfaz “Registrar mensaje SLBTR MT199 con código de solicitud 21” |
| 2.El usuario introduce los datos de los campos del mensaje y presiona “aceptar”<br>a. En caso de presionar el botón cancelar<br>Ver sección “Cancelar operación” | 3. El sistema valida los datos.<br>4. El sistema persiste el mensaje y finaliza el caso de uso. |

***Prototipo de Interfaz***




| Flujos alternos | |
|---|---|
| **Sección "Datos incorrectos"** | |
| **Acción del actor** | **Respuesta del sistema** |
| | 1. El sistema señala los datos incorrectos y muestra el mensaje "Datos incorrectos"<br>2. El sistema regresa al flujo normal de eventos. |
| **Sección "Cancelar operación"** | |
| **Acción del actor** | **Respuesta del sistema** |
| | 1. El sistema cancela la operación y finaliza el caso de uso. |

### 2.1.4. Validación de requisitos

Esta actividad se realizó con el objetivo de garantizar que los requisitos fueran correctos y cumplieran con las necesidades del cliente. Se desarrolló mediante las siguientes técnicas:

- **Validación de requisitos mediante prototipos y escenarios [59]**

Se presentaron los prototipos elaborados durante la especificación a especialistas funcionales del BNC, para corroborar que responden a las necesidades y aspiraciones del cliente. Para ello, se desarrollaron varios escenarios posibles con el auxilio de juegos de datos, de forma tal que se visualizaron las diferentes funcionalidades que tendría el subsistema. Las no conformidades fueron documentadas y corregidas.

- **Revisión técnica de artefactos [59]**

Se realizaron revisiones de los artefactos por parte del equipo de calidad del proyecto, se corrigieron las no conformidades identificadas hasta ser liberada la documentación.

## 2.3. El análisis

Este flujo de trabajo de trabajo tiene como objetivos generales analizar los requisitos, refinarlos y estructurarlos en un modelo de objetos que sirva como primera impresión del modelo de diseño. Por ende, se pretende es conseguir una comprensión más precisa de los requisitos y una descripción de los mismos que sea fácil de mantener y que ayude a estructurar el sistema entero, incluyendo su arquitectura [35].

### 2.3.1. Modelo del análisis

En un nivel técnico, la ingeniería del *software* empieza con una serie de tareas de modelado que llevan a una especificación completa de los requisitos y a una representación del diseño general del *software* a construir. El modelo de análisis, que realmente es un conjunto de modelos, es la primera representación técnica de un sistema. Debe lograr tres objetivos primarios:

- Describir lo que requiere el cliente.
- Establecer una base para la creación de un diseño de *software*
- Definir un conjunto de requisitos que se pueda validar una vez que se construye el *software* [50].

#### 2.3.1.1. Clases del análisis

Una clase de análisis representa una abstracción de una o varias clases y/o subsistemas del diseño del sistema[35]. Esta abstracción posee las siguientes características:

- Se centra en el tratamiento de los requisitos funcionales.
- Las clases de análisis siempre encajan en uno de tres estereotipos básicos: de interfaz, de control o de entidad.

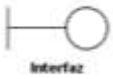
| Nombre | Características | Estereotipos |
|---|---|---|
| **Interfaz** | Las clases de interfaz se utilizan para modelar la interacción entre el sistema y sus actores. Esta información a menudo implica recibir y presentar información y peticiones de y hacia los usuarios y los sistemas externos. |  |
| **Control** | Las clases de control representan coordinación, secuencia, transacciones, y control de otros objetos y se usan con frecuencia para encapsular el control de un caso de uso en concreto. |  |
| **Entidad** | Las clases de entidad se utilizan para modelar información que posee una larga vida y que es a menudo persistente. Las clases de entidad modelan la información y el comportamiento asociado de algún fenómeno o concepto, como una persona, un objeto o un suceso del mundo real. |  |

***Tabla # 1. Clases del análisis.***

### 2.3.1.2. Diagramas de clases del análisis

A continuación se ilustra el diagrama de clases del análisis de los CUS: “CU Gestionar mensaje SLBTR MT199 con código de solicitud 21” y “CU Registrar configuración de mensaje SLBTR”.

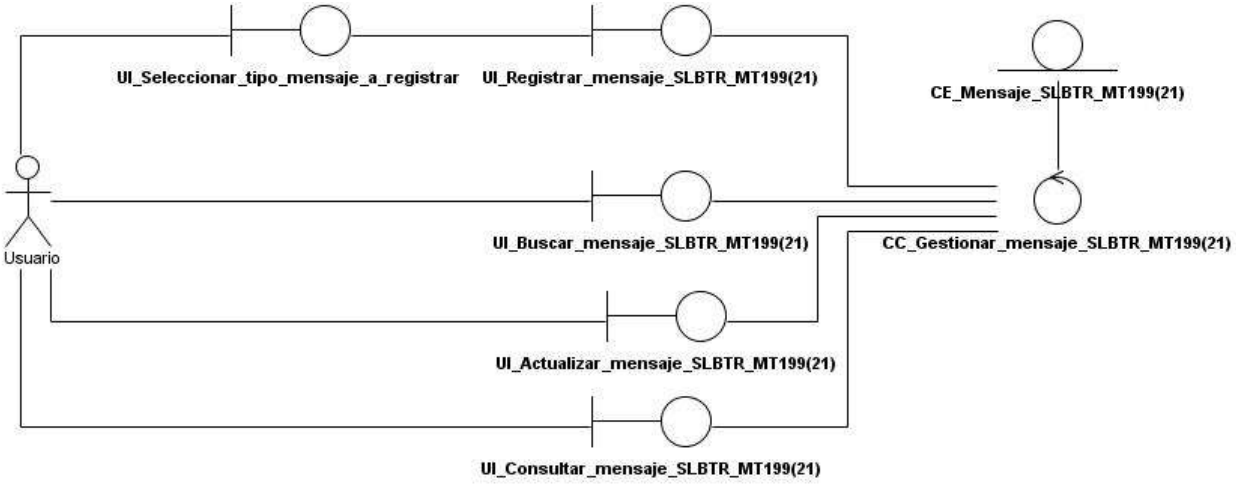



Figura 5 Diagrama de clases de análisis “CU Gestionar mensaje SLBTR MT199 con código de solicitud 21”

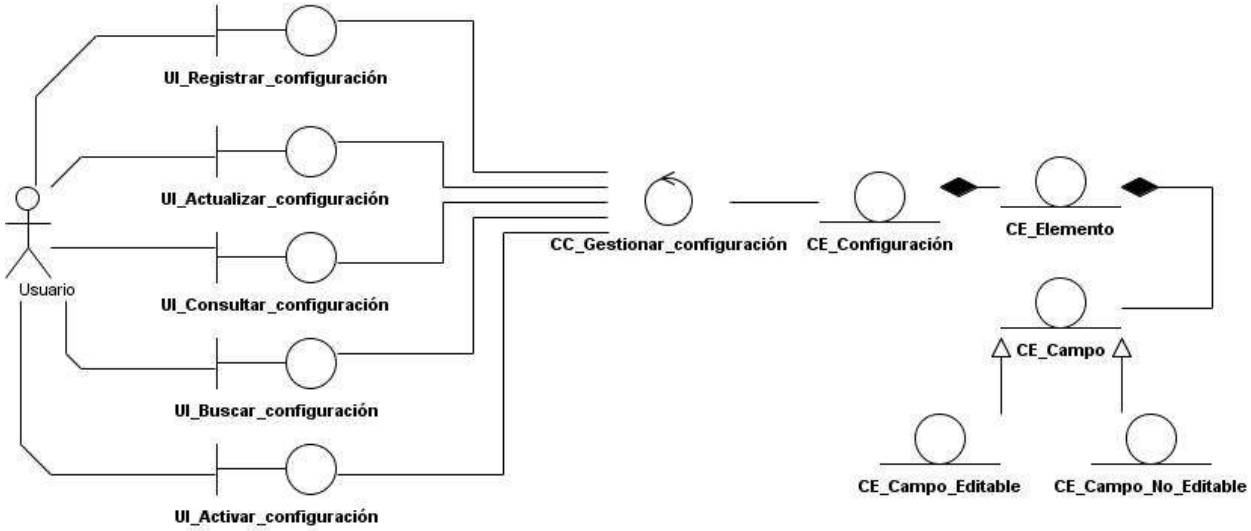



Figura 6 Diagrama de clases del análisis “CU Gestionar configuración de mensaje SLBTR”

### 2.3.1.3. Descripción de la arquitectura

Se utiliza la arquitectura base del sistema informático Quarxo desarrollado bajo la tecnología JEE. La misma, está compuesta por tres capas: Presentación, Negocio y Acceso a Datos y una capa transversal a las otras con las clases del dominio como se muestra a continuación [51]:

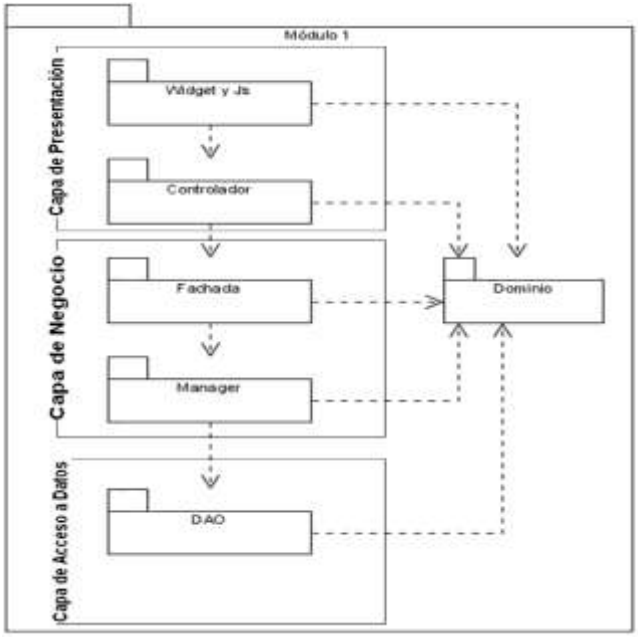



Figura 7 Arquitectura del sistema

- **Capa de Presentación**

  En esta capa se desarrolla la lógica de presentación. En el lado del cliente se utiliza la biblioteca Dojo Toolkit para generar las interfaces que interactúan con el usuario. En el lado del servidor se emplea Spring MVC para recibir, controlar y enviar respuestas a las peticiones realizadas desde el cliente y también se utilizará Spring Web Flow para representar y controlar los flujos complejos y reutilizables de la aplicación. La capa de Presentación está relacionada con la capa de Negocio y la capa de Dominio.

- **Capa de Negocio**

  La capa de Negocio está divida en dos subcapas. Estas subcapas son Facade y Manager. La Facade es el punto de intercambio entre la capa de Presentación y la capa de Negocio. Esta capa no tiene lógica de negocio, sino que agrupa las funcionalidades según su naturaleza para que pueda ser invocada desde la capa de Presentación. La subcapa Facade delega a la subcapa Manager la realización de la lógica del negocio. Por otro lado, la subcapa Manager tiene la jerarquía de clases suficiente para implementar el negocio de la aplicación.

  Esta subcapa utiliza la capa de Acceso a Datos para obtener los datos persistidos y la capa de Dominio para generar las entidades del negocio. Desde la capa de Negocio se envuelven transaccionalmente las funcionalidades de la aplicación para evitar inconsistencia en los datos. Se utiliza para esto, las políticas de transacciones que propone Spring Framework. Se utiliza el

contenedor de Spring Framework para declarar y representar las relaciones de dependencia de cada una de las clases, Spring Security para asegurar las invocaciones a los métodos, Spring AOP para ejecutar las transacciones y la auditoría de cada uno de los métodos del negocio.

- **Capa de Acceso a Datos**

  En esta capa se encuentran las operaciones que permiten la interacción con el gestor de base de datos desde la aplicación. Desde aquí se ejerce la conexión con el gestor de base de datos, permitiendo así la persistencia y el acceso a la información. La interacción con la capa de negocio se realiza a través de interfaces. Se utiliza el patrón DAO para el desarrollo de la capa, el *framework* de persistencia Hibernate y los módulos Spring ORM y Spring DAO. Hibernate como tecnología ORM para el manejo objetual de acciones de persistencia con la base de datos.

  Spring ORM y Spring DAO para soportar la integración con Hibernate y la utilización del patrón DAO. La arquitectura definida presenta como otra de sus características la utilización del patrón MVC, patrón que propone dividir la aplicación en tres capas distintas, el Modelo, la Vista y el Controlador, potenciando la flexibilidad y la adaptabilidad a futuros cambios. Específicamente el Modelo es la representación de la información que maneja la aplicación, la Vista constituye la representación del modelo en forma gráfica disponible para la interacción con el usuario y el Controlador se encarga de responder a las solicitudes del usuario desde la Interfaz, manejando los diferentes eventos a través de las funcionalidades necesarias y la información perteneciente al Modelo.

#### 2.3.1.4. Paquetes del análisis

El sistema está estructurado a través de subsistemas, módulos y componentes: Los subsistemas, módulos y componentes son definidos según las funcionalidades identificadas en la captura de requisitos. Los subsistemas agrupan un conjunto de módulos relacionados con los procesos que ejecutan. Los módulos agrupan un conjunto de casos de uso que representan uno o más procesos bancarios estrechamente relacionados y por último los componentes son un conjunto de funcionalidades comunes que son reutilizados por otros módulos del sistema [51].

## 2.4. El diseño

A partir de los resultados obtenidos en la fase de análisis se comienza con el desarrollo de los artefactos de la fase de diseño. Se modela el sistema y se obtiene su forma (incluida la arquitectura) para que soporte todos los requisitos –incluyendo los requisitos no funcionales y otras restricciones- que se le suponen. El diseño proporciona una comprensión detallada los requisitos e impone una estructura del sistema que se pretende conservar lo más fielmente posible cuando se da forma al sistema [35].

El diseño posee los siguientes propósitos:

- Adquirir una comprensión de los aspectos relacionados con los requisitos no funcionales y restricciones relacionadas con los lenguajes de programación, componentes reutilizables, sistemas operativos, tecnologías de distribución y de interfaz de usuario.
- Crear una entrada apropiada y un punto de partida para actividades de implementación, capturando los requisitos o subsistemas individuales, interfaces y clases.
- Descomponer los trabajos de implementación en partes más manejables que puedan ser llevadas a cabo por diferentes equipos de desarrollo [35].

### 2.4.1. Modelo del diseño

El modelo de diseño es un modelo de objetos que describe la realización física de los casos de uso centrándose en como los requisitos funcionales y no funcionales, junto con otras restricciones relacionadas con el entorno de implementación, tienen impacto en el sistema a considerar. Sirve de abstracción de la implementación y es utilizada como entrada fundamental de las actividades de implementación [35].

#### 2.4.2.1. Diagrama de paquetes del diseño

Durante el desarrollo de *software* resulta muy conveniente agrupar clases y ficheros por diferentes criterios para lograr la organización y facilitar la comprensión del código de la aplicación, resultando conveniente el desarrollo de los diagramas de paquetes, los cuales muestran cómo está dividido el sistema en agrupaciones lógicas mostrando las dependencias entre las mismas[53]. En la siguiente figura se ilustra el diagrama de paquetes del subsistema.

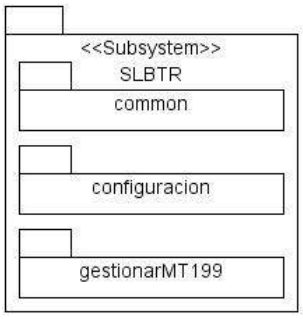



Figura 8 Diagrama de paquetes del subsistema SLBTR

A continuación se muestra la estructura y dependencia de paquetes del módulo “Configuración” del subsistema con su correspondiente explicación. El diagrama de paquetes que comprende al módulo “Gestionar MT199” asociado a la gestión y envío de los mensajes SLBTR se comporta de manera similar.

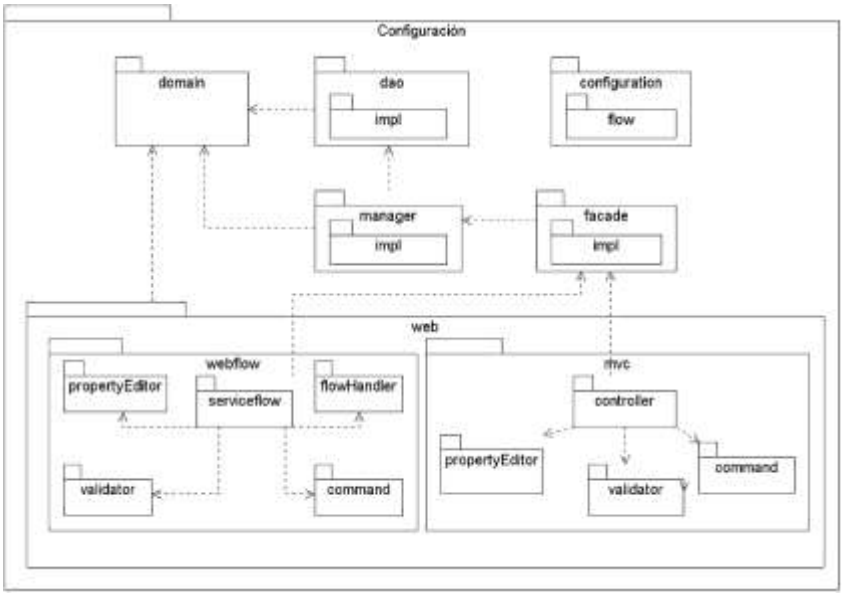



Figura 9 Diagrama de paquetes del diseño del módulo Configuración

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

- **-servlet.xml**: Define el contexto de Spring MVC.
- **-bussiness.xml**: Define el contexto para el negocio.
- -**webflow.xml**: Define el contexto para Spring Web Flow.
- -**dataaccess.xml**: Define el contexto para acceso a datos.

**Paquete *flow***: Se encuentran presentes los archivos XML que definen los flujos para Spring Web Flow.

**Paquete *facade*:** Agrupa las interfaces y sus respectivas implementaciones, encargadas de brindar las funcionalidades que serán usadas por la presentación.

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

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

**Paquete *domain*:** Paquete con las clases relacionadas con el dominio del módulo en cuestión.

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

- **Paquete *mvc*:** Agrupa todo lo referente a la lógica de presentación cuando se hace uso de Spring MVC.
- **Paquete *webflow*:** Se agrupa todo lo referente a la lógica de presentación cuando se hace uso de Spring Web Flow.
- **Paquete *controller*:** Clases que extienden de los controladores propuestos por Spring MVC, encargadas de responder a las peticiones realizadas por el cliente.
- **Paquete *serviceFlow*:** Contiene las clases encargadas de establecer la comunicación entre el flujo y la fachada del módulo.
- **Paquete flowHandler:** Clases utilizadas para personalizar el trabajo con el Web Flow.
- **Paquete *command*:** Clases representan los datos que son introducidos y mostrados en los formularios de la interfaz de usuario.
- **Paquete *propertyEditor*:** Agrupa las clases que actúan como intermediarias para convertir datos de la interfaz de usuario en objetos y viceversa.
- **Paquete *validator*:** Cuenta con las clases encargadas de realizar la validación de datos en el lado del servidor.

#### 2.4.1.2. Diagramas de clases del diseño

Una realización de caso de uso del diseño es una colaboración del modelo de diseño que describe como se realiza un caso de uso específico y como se ejecuta en término de clases de diseño y sus objetos. Una realización contiene diagramas de clases que muestran sus clases de diseño participantes y diagramas de interacción que muestran la realización de un flujo o escenario en concreto, en términos de interacción entre objetos [35].

Específicamente los diagramas de clases de diseño son muy útiles porque muestran a través de atributos y métodos la estructura de las clases que después serán escritas en algún lenguaje de programación.

Un diagrama de clases del diseño es una representación más concreta que el diagrama de clases del análisis. Además, representa la parte estática del sistema y las clases y sus relaciones.

El caso de uso “CU Registrar configuración de mensaje SLBTR” es uno de los más significativos, para su realización en el diseño se hace uso del *framework* Spring Web Flow. Mediante un flujo de páginas (vistas) se recogen los datos ingresados por el usuario, que componen una configuración, y se registran. A continuación se representa el diagrama de clases del diseño para dicho caso de uso.

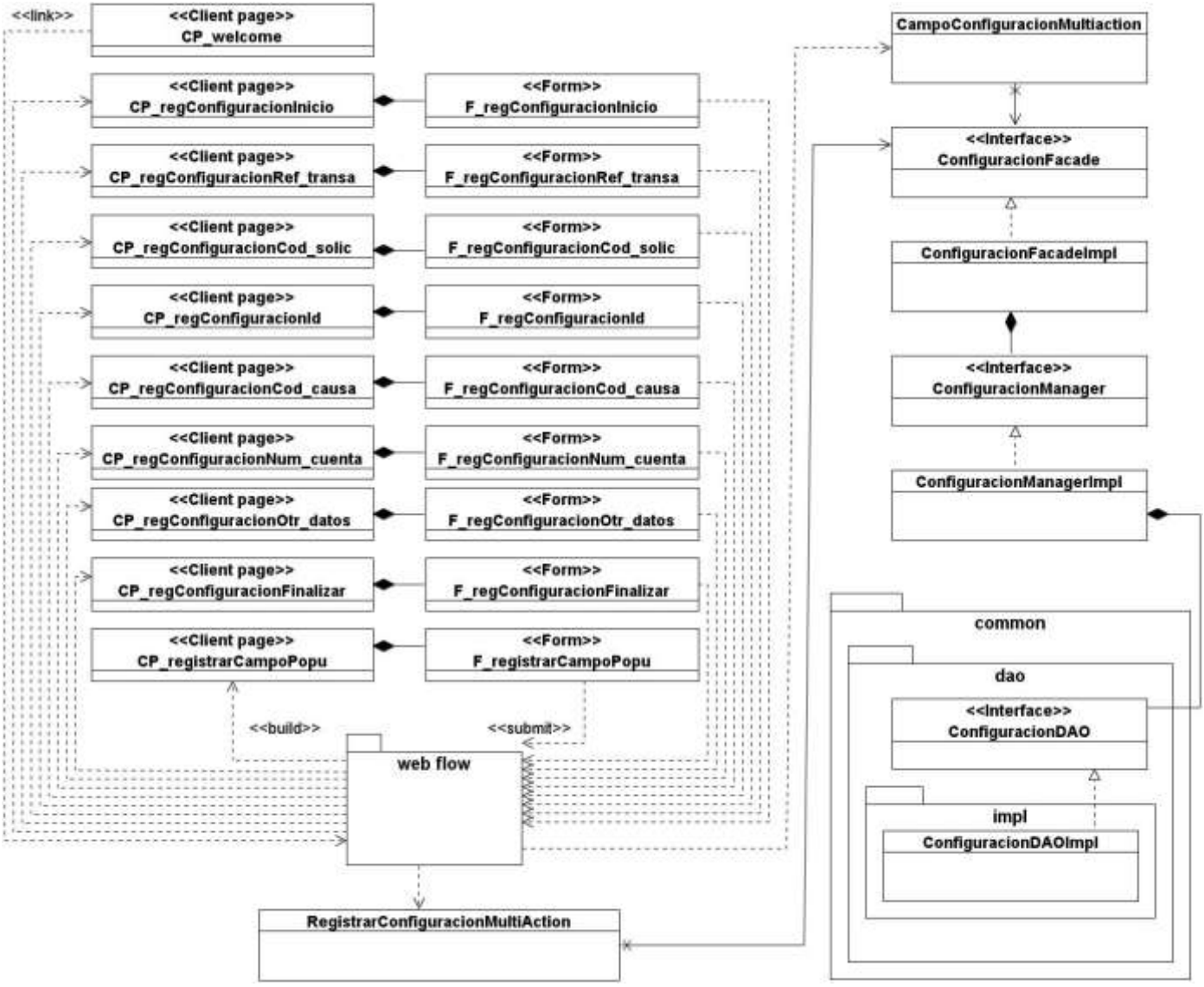



Figura 10 Diagrama de clases del diseño “CU Registrar configuración de mensaje SLBTR”

Seguidamente se muestra el diagrama de clases del diseño correspondiente a la realización del caso de uso “CU Registrar mensaje SLBTR MT199 con código de solicitud 21”.

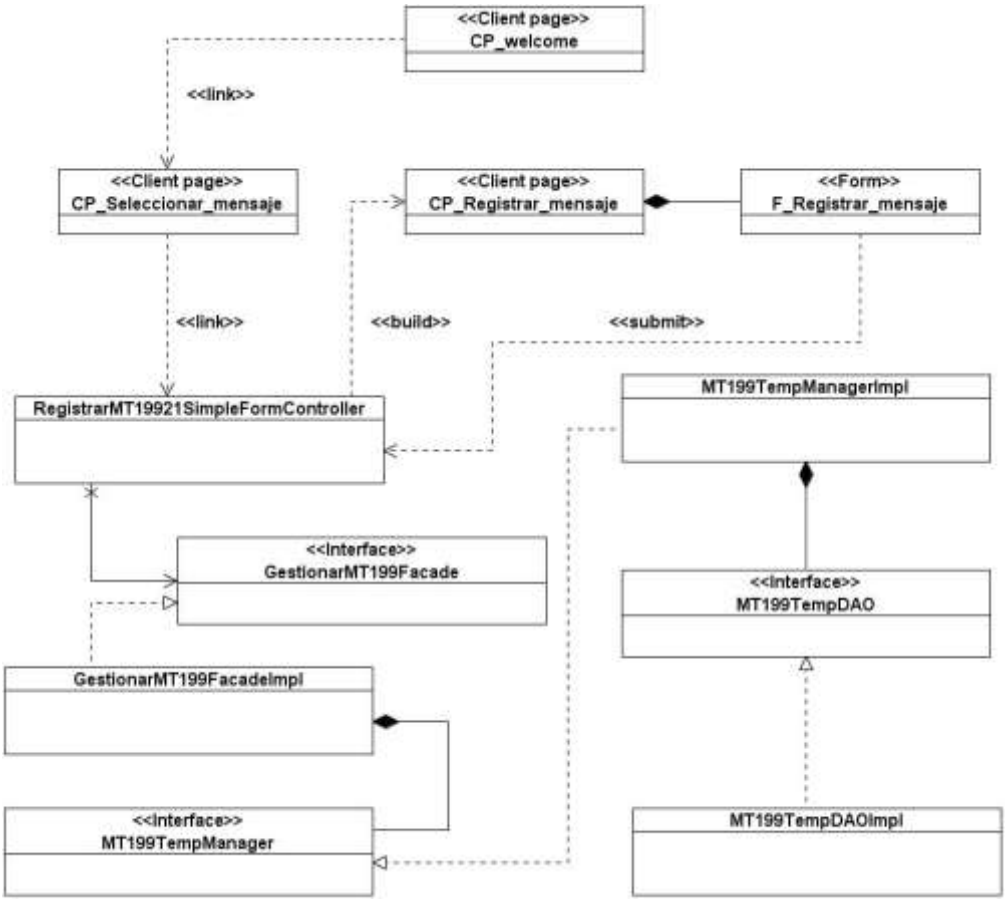



Figura 11 Diagrama de clases del diseño "CU Registrar mensaje SLBTR con código de solicitud 21"

En el Anexo # 5 se pueden consultar los diagramas mostrados anteriormente con un mayor nivel de detalle.

#### 2.4.1.3. Diagrama de secuencia

Los diagramas de interacción son utilizados para modelar los aspectos dinámicos de los sistemas, muestran la realización de un flujo o escenario concreto de un caso de uso en términos de interacción entre objetos del diseño. Se les atribuye una gran importancia debido a la compresión del sistema a la que puede llegarse a través de ellos [50].

Debido a lo anterior se realizaron diagramas de secuencia, que son un ejemplo de diagramas de interacción, que destacan principalmente el orden temporal de los mensajes para la realización de un escenario. A continuación se presenta el diagrama de secuencia asociado a la realización del caso de uso "CU Registrar mensaje SLBTR con código solicitud 21"

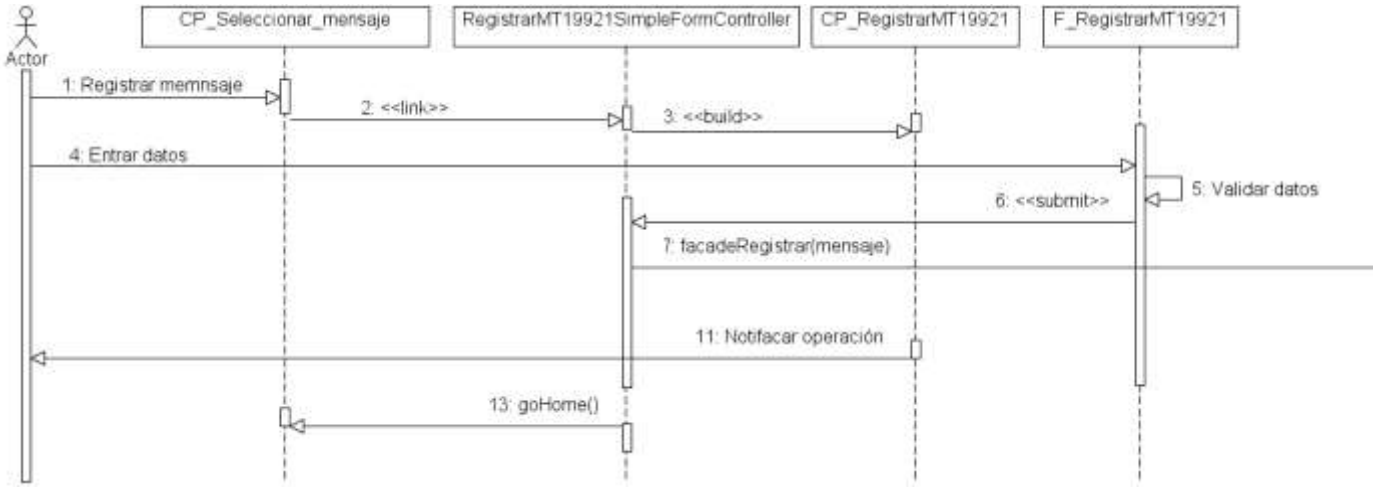



Figura 12 Diagrama de secuencia "CU Registrar mensaje SLBTR con código de solicitud 21"

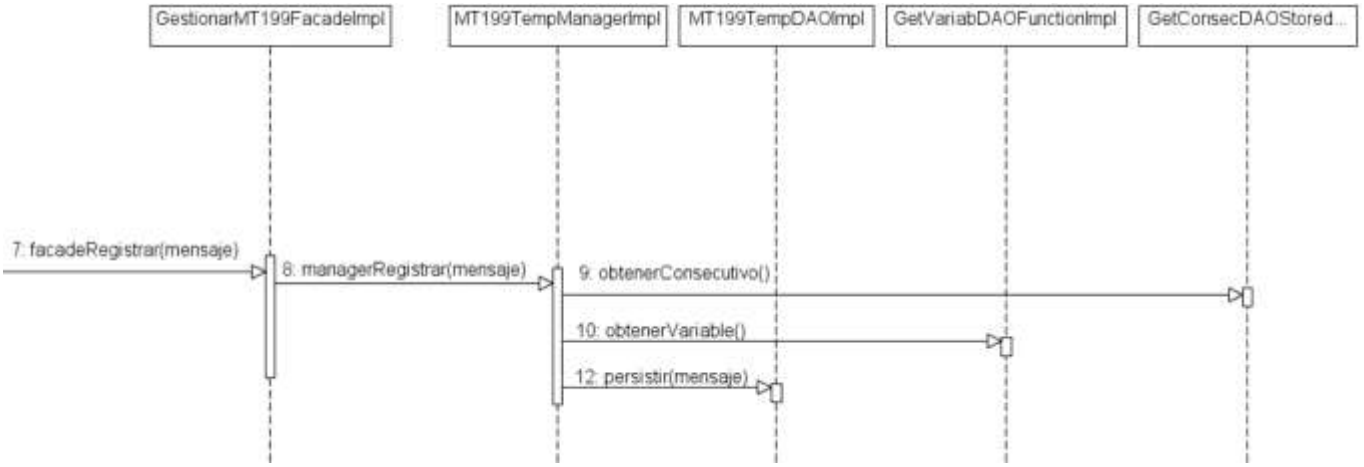



Figura 13 Continuación del diagrama de secuencia "CU Registrar mensaje SLBTR con código de solicitud 21"

### 2.4.2. Modelo de datos

El modelo de datos es un modelo abstracto que representa como se deben usar y representar los datos. Además, describe la estructura de la base de datos, las relaciones entre los datos y las restricciones que deben cumplirse entre ellos [19]. Para la solución propuesta el modelo de datos se construyó teniendo en cuenta la menor cantidad de campos nulos.

Las tablas *o_slbtr_configuracion_MT199*, *o_slbtr_elemento*, *o_slbtr_campo*, *o_slbtr_campo_editable*, *o_slbtr_campo_no_editable*, *o_slbtr_componente*, *o_slbtr_n_exp_reg*, *o_slbtr_n_componente* constituyen las encargadas de responder por el almacenamiento de la configuración de los mensajes SLBTR.

En las tablas *o_slbtr_mensaje, o_slbtr_valor, m_slbtr_mt199, m_menslbtr, o_tmpslbtr* se almacenarán los datos de los mensajes SLBTR.

El modelo de datos se muestra a continuación:

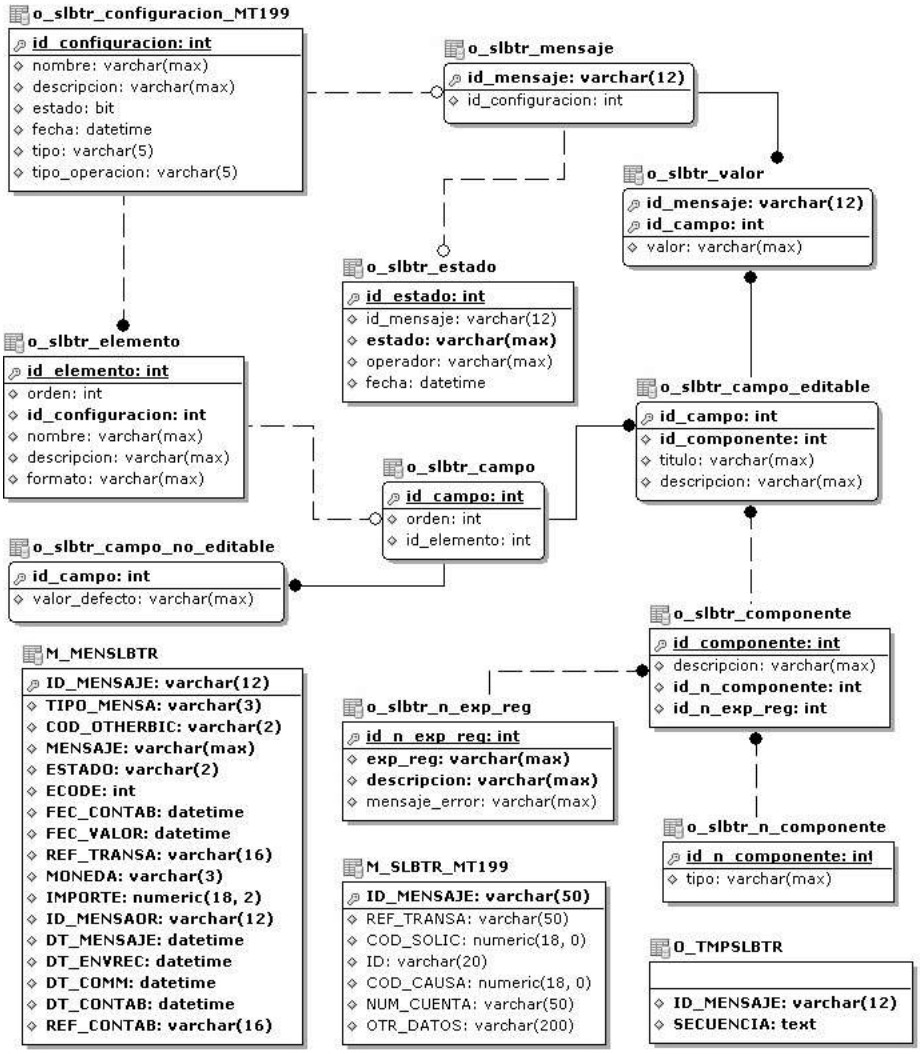



Figura 14 Diagrama del modelo datos del subsistema Mensajería SLBTR

### 2.4.3. Patrones de diseño utilizados

Generalmente, para elaborar el diseño se utilizan un grupo de patrones o modelos para lograr los objetivos esperados. Estos son considerados como procedimientos para solucionar diversos problemas del mismo tipo y son la base para la búsqueda de soluciones a dificultades comunes en el desarrollo de *software*.

Para definir el diseño de la solución propuesta se tuvieron en cuenta varios patrones, a continuación se especifican los patrones seleccionados:

**Patrón de Acceso a Datos (DAO):** Se aplica con la definición de las interfaces DAO que disminuyen la complejidad de los objetos del dominio, al librarlos de la responsabilidad de manejar la implementación de sus fuentes de datos.

**Patrones GRASP**

- **Controlador:** Las clases controladoras de los módulos, constituyen un ejemplo de la aplicación de este patrón, las mismas tendrán la responsabilidad de escuchar y responder a las peticiones realizadas por la presentación y de comunicarse con la capa de negocio.

  Ejemplo: *RegistrarMT19921SimpleFormController*.

- **Experto:** Este patrón fue utilizado en el diseño del subsistema de manera general en la definición de las clases de acuerdo con las funcionalidades que deben realizar a partir de la información que manejan.

- **Alta cohesión:** Este patrón fue utilizado en el diseño del subsistema de manera general evidenciándose en la agrupación de las clases en dependencia de los requerimientos.

- **Bajo acoplamiento:** Este patrón se evidencia con la definición de interfaces e implementaciones, como puede ser la interfaz *GestionarMT199Facade* y su implementación *GestionarMT199FacadeImpl* permitiendo que clases como *RegistrarMT19921SimpleFormController* se relacionen únicamente con ellas para realizar sus operaciones, reduciendo el impacto de cambios posteriores en el negocio del sistema, logrando que el código sea más fácil de entender, mantener y reutilizar.

**Patrones GOF**

- **Fachada:** La utilización de este patrón se evidencia en la definición de la interfaz *GestionarMT199Facade* responsable de la comunicación entre la presentación y el grupo de clases e interfaces más complejas que se encargan de la lógica de negocio y el acceso a datos.

- **Cadena de Responsabilidad:** Cuando desde la vista se solicita información que se encuentra en la base de datos, esta es atendida primeramente por los *Controller*, luego por la *Facade*, el *Manager* y finalmente el *DAO*, evidenciándose de esta manera la utilización de dicho patrón.

### 2.4.4. Validación del diseño

Las métricas para aplicaciones informáticas, no son perfectas; muchos expertos argumentan que se necesita más experimentación hasta que se puedan emplear bien las métricas de diseño. Sin embargo, el diseño sin medición, es una alternativa inaceptable [27].

Las métricas de *software* son una medida cuantitativa del grado en que un sistema, componente o proceso posee un atributo dado. Permiten averiguar cuán bien están definidas las clases y el sistema, lo cual tiene un impacto directo en el mantenimiento del mismo, tanto por la comprensión de lo desarrollado como por la dificultad de modificarlo con éxito. Estas métricas tienen como propósito entender y mejorar la calidad del producto, evaluar la efectividad del proceso y mejorar la calidad del trabajo llevado a cabo al nivel del proyecto [50].

Las métricas orientadas a objetos, separan las métricas basadas en clases en cuatro amplias categorías: tamaño, herencia, valores internos y valores externos. Las métricas orientadas al tamaño se centran en el recuento de atributos y operaciones para cada clase individual y los valores promedio para el sistema orientado a objetos [50].

Se utilizarán las métricas: Tamaño operacional de clase (TOC) [40; 50] y Relaciones entre clases (RC) [40; 50], con el objetivo de evaluar la calidad del diseño propuesto.

Las métricas TOC y RC posibilitan medir los siguientes atributos de calidad [40; 50]:

- **Responsabilidad:** Responsabilidad que posee una clase en un marco conceptual correspondiente al modelado de la solución propuesta.
- **Complejidad del mantenimiento:** Nivel de esfuerzo necesario para sustentar, mejorar o corregir el diseño de *software* propuesto. Puede influir significativamente en los costes y la planificación del proyecto.
- **Complejidad de implementación:** Grado de dificultad que tiene implementar un diseño de clases determinado.
- **Reutilización:** Significa cuán reutilizada es una clase o estructura de clase dentro de un diseño de *software*.
- **Acoplamiento:** Dependencia o interconexión de una clase o estructura de clase respecto a otras.
- **Cantidad de pruebas:** Número o grado de esfuerzo necesario para realizar las pruebas de calidad al producto diseñado.

#### 2.4.4.1. Tamaño operacional de clase (TOC)

Para la evaluación de las clases fueron utilizados umbrales para el tamaño general, la responsabilidad, la complejidad y la reutilización de las clases.

Nota: Promedio de operaciones (PO)

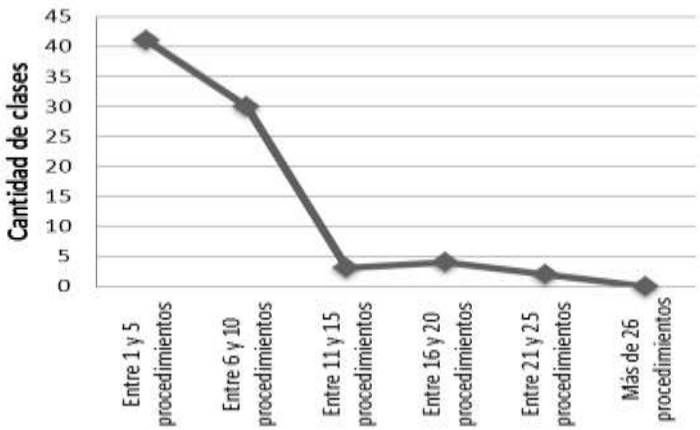
**Atributos de calidad que afecta esta prueba.**

| Atributo | Categoría | Criterio |
|---|---|---|
| Responsabilidad | Baja | < =PO |
| | Media | Entre PO y 2* PO |
| | Alta | > 2* PO |
| Complejidad implementación | Baja | < =PO |
| | Media | Entre PO y 2* PO |
| | Alta | > 2* PO |
| Reutilización | Baja | > 2* PO |
| | Media | Entre PO y 2* PO |
| | Alta | <= PO |

| Atributo de calidad | Modo en que lo afecta |
|---|---|
| **Responsabilidad** | Un aumento del TOC implica un aumento de la responsabilidad asignada a la clase. |
| **Complejidad de implementación** | Un aumento del TOC implica un aumento de la complejidad de implementación de la clase. |
| **Reutilización** | Un aumento del TOC implica una disminución del grado de reutilización de la clase. |

**Resultados de la evaluación de la métrica TOC y su influencia en los atributos de calidad.**

Como resultado de la evaluación de la métrica TOC se obtuvo lo siguiente:




**Representación en % de la incidencia de los resultados obtenidos en cada atributo de calidad:**




**Análisis de los resultados obtenidos en la evaluación de la métrica TOC**

Durante la evaluación de la métrica TOC los resultados obtenidos demuestran que los atributos de calidad de las clases se encuentran en un nivel satisfactorio; de manera que se puede confirmar la elevada reutilización con un 80 % (elemento clave en el proceso de desarrollo de *software*) y cómo se reducen la responsabilidad y la complejidad de implementación en un 85 %.

### 2.4.4.2. Relaciones entre clases (RC)

Para la evaluación de las clases fueron utilizados umbrales para el acoplamiento, complejidad de mantenimiento, la reutilización y cantidad de pruebas.

| Atributos de calidad | Categoría | Criterio |
|---|---|---|
| Acoplamiento | Ninguno | 0 |
| | Bajo | 1 |
| | Medio | 2 |
| | Alto | >2 |
| Complejidad de Mantenimiento | Baja | <= PO |
| | Media | Entre PO y 2*PO |
| | Alta | > 2*PO |
| | Alta | > 2*PO |

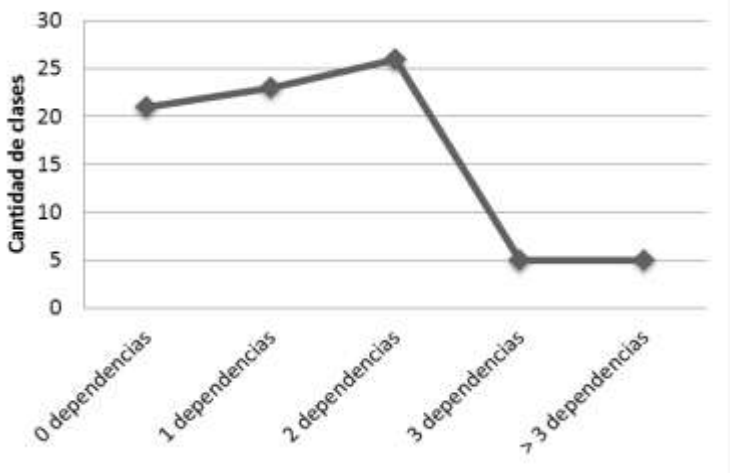
| Atributos de calidad | Categoría | Criterio |
|---|---|---|
| Reutilización | Baja | >2* PO |
| | Media | Entre PO y 2*PO |
| | Alta | <= PO |
| Cantidad de Pruebas | Baja | <= PO |
| | Media | Entre PO y 2*PO |
| | Alta | > 2*PO |

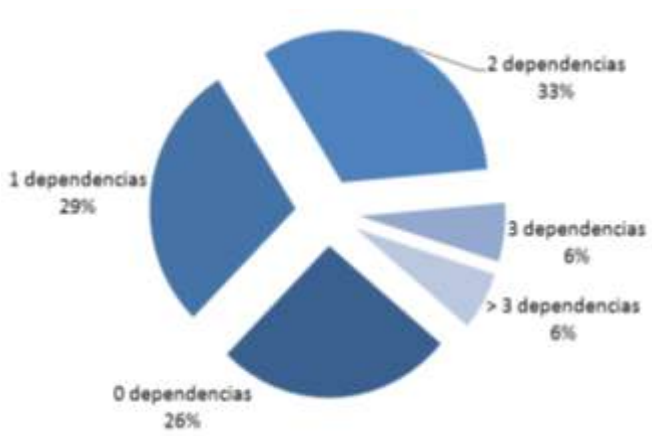
Esta métrica está dada por el número de relaciones de uso de una clase con otra y evalúa los siguientes atributos de calidad.

| Atributo de calidad | Modo en que lo afecta |
|---|---|
| **Acoplamiento** | Un aumento del RC implica un aumento del acoplamiento de la clase. |
| **Complejidad de mantenimiento** | Un aumento del RC implica un aumento de la complejidad del mantenimiento de la clase. |
| **Reutilización** | Un aumento del RC implica una disminución en el grado de reutilización de la clase. |
| **Cantidad de pruebas** | Un aumento del RC implica un aumento de la Cantidad de pruebas de unidad necesarias para probar una clase. |

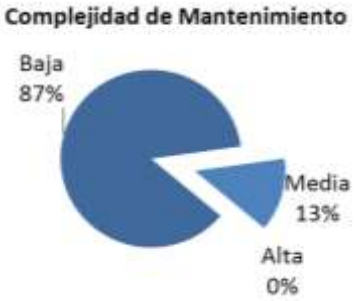
**Resultados de la evaluación de la métrica RC y su influencia en los atributos de calidad.**

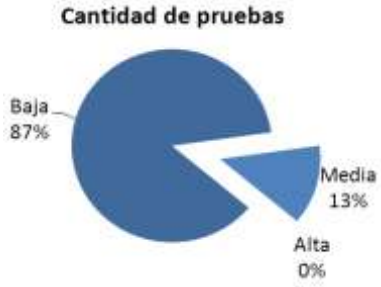
Como resultado de la evaluación de la métrica RC se obtuvo lo siguiente:




**Representación en % de la incidencia de los resultados obtenidos cada atributo de calidad**



**Análisis de los resultados obtenidos en la evaluación de la métrica RC**

Luego de aplicarse la métrica de diseño RC y obtenidos los resultados de la evaluación del instrumento de medición de la métrica, se puede concluir que el diseño propuesto tiene una calidad aceptable teniendo en cuenta que el 88 % de las clases empleadas poseen menos de 3 dependencias de otras clases lo que

lleva a evaluaciones positivas de los atributos de calidad involucrados (bajo acoplamiento con 51 %, baja complejidad de mantenimiento con un 87 %, baja cantidad de pruebas con 87 % y alta reutilización con un 73 %). Favoreciendo de esta manera la reutilización de las clases así como la modificación e implantación del diseño.

**Matriz de inferencia de indicadores de calidad**

La matriz de inferencia permite evaluar positiva o negativamente los resultados obtenidos de las relaciones atributo/métrica en una escala numérica, donde el valor 1 representa un resultado "positivo" y el valor 0 "negativo". Si la métrica no influye en el atributo de calidad la relación es considerada como nula y es representada con un guion simple (-). Al promediar por cada atributo las relaciones no nulas, se obtiene un resultado general que al ubicarse en el rango de evaluación (valor <=0.4: "Mala", valor >0.4 y valor<0.7: "Regular", valor>=0.7: "Buena") permite arribar a conclusiones sobre la calidad del diseño propuesto. Teniendo en cuenta que los resultados arrojados con la aplicación de las métricas fueron positivos para todos los atributos, la matriz de la inferencia queda elaborada de la siguiente manera:

| Atributos\Métricas | TOC | RC | Promedio |
|---|---|---|---|
| Responsabilidad | 1 | (-) | 1 |
| Complejidad de implementación | 1 | (-) | 1 |
| Reutilización | 1 | 1 | 1 |
| Acoplamiento | (-) | 1 | 1 |
| Complejidad de mantenimiento | (-) | 1 | 1 |
| Cantidad de prueba | (-) | 1 | 1 |

El promedio general es 1 y teniendo en cuenta el rango de evaluación se concluye que el subsistema Mensajería SLBTR presenta un diseño con buena calidad.

**Conclusiones parciales**

- Se desarrollaron los flujos de trabajo Requisitos, Análisis y Diseño propuestos por la metodología RUP, obteniendo el Modelo de casos de uso, Modelo de análisis y Modelo de diseño, como principales artefactos en cada caso.
- Con el objetivo de garantizar que los requerimientos fueran correctos y cumplieran con las necesidades del cliente, se utilizaron técnicas de validación mediante prototipos y escenarios; además se efectuaron revisiones técnicas de los artefactos, garantizándose la suficiente calidad como para dar paso al flujo de trabajo Análisis y Diseño.

- Se describió la arquitectura basada en el patrón arquitectónico MVC, fundamentada en la utilización de los *frameworks* Spring, Hibernate y Dojo Toolkit, y los patrones considerados para el diseño de los casos de uso.
- Se aplicaron las métricas TOC y RC propuestas por Lorenz y Kidd con el objetivo de validar los artefactos del diseño.
- Durante la evaluación de la métrica TOC los resultados obtenidos demuestran que los atributos de calidad de las clases se encuentran en un nivel satisfactorio; de manera que se puede confirmar la elevada reutilización con un 80 % (elemento clave en el proceso de desarrollo de *software*) y cómo se reducen la responsabilidad y la complejidad de implementación en un 85 %.
- Luego de aplicarse la métrica de diseño RC y obtenidos los resultados de la evaluación del instrumento de medición de la métrica, se puede concluir que el diseño propuesto tiene una calidad aceptable teniendo en cuenta que el 88 % de las clases empleadas poseen menos de 3 dependencias de otras clases lo que lleva a evaluaciones positivas de los atributos de calidad involucrados (bajo acoplamiento con 51 %, baja complejidad de mantenimiento con un 87%, baja cantidad de pruebas con 87 % y alta reutilización con un 73 %). Favoreciendo de esta manera la reutilización de las clases así como la modificación e implantación del diseño.
- La correcta aplicación de los patrones contribuyó a la buena calidad del diseño propuesto, concluyéndose a partir de los resultados de la validación mediante métricas, permitiendo dar paso a los flujos de trabajo Implementación y Pruebas.

# CAPÍTULO 3: IMPLEMENTACIÓN Y VALIDACIÓN DE LA SOLUCIÓN

**Introducción**

En el presente capítulo se realizan las actividades y se obtienen los artefactos más importantes de los flujos de trabajo Implementación y Pruebas de la metodología RUP. Se expone la nomenclatura usada tanto en el código como en los paquetes y clases del subsistema, se describe cómo los elementos del Modelo de Diseño se implementan en términos de componentes y se abordan algunos temas importantes de la implementación, como la utilización del *framework* Spring Web Flow a través de la explicación de un escenario del sistema. Se realizan pruebas de caja blanca y caja negra para evaluar la calidad del producto obtenido en la fase de implementación. Finalmente, se validan las variables de investigación con el objetivo de comprobar si la solución reduce el tiempo y el consumo de recursos cuando se requiere gestionar los mensajes SLBTR en el BNC.

## 3.1. Implementación

La implementación es el principal flujo de trabajo en la fase de construcción. En él se describe cómo los elementos del modelo de diseño se implementan en función de componentes y por ende en piezas más manejables por el lenguaje del programador. Tiene como objetivo llevar a cabo la implementación de cada una de las clases significativas del diseño [50].

### 3.1.1. Estándares de codificación

**Convenciones de nomenclatura**

Los nombres de las clases comienzan con la primera letra en mayúscula y el resto con minúscula, en caso de ser una palabra compuesta se empleará la notación *Pascal Casing*. Ejemplo: *ConfiguracionManager*

Los nombres de los métodos y los atributos de las clases, así como los nombres de ficheros de código JavaScript y sus funciones y variables internas comienzan con la primera letra en minúscula, en caso de que sea un nombre compuesto se empleará notación *Camel Casing*. Ejemplo: *idMensaje*.

**Nomenclatura según el tipo de clase**

Las clases que se encuentran dentro del paquete *controller* se nombran adicionándoles el nombre del controlador de Spring, del cual heredan al final del nombre de la clase (*MultiActionController*). Ejemplo: *CargarDatosMultiActionController*.

Las clases que se encuentran dentro del paquete *facade* se nombran adicionándoles como sufijo del nombre la palabra *Facade*. Para el paquete *impl* se les nombra igual que la interfaz que implementan y se le agrega como sufijo la palabra *Impl*. Ejemplo: *ConfiguracionFacade*, *ConfiguracionFacadeImpl*.

Las clases que se encuentran dentro del paquete *manager* se nombran adicionándoles como sufijo del nombre la palabra *Manager*. Para el paquete *impl* se les nombra igual que la interfaz que implementan y se le agrega como sufijo la palabra *Impl*. Ejemplo: *ConfiguracionManager*, *ConfiguracionManagerImpl*.

Las clases que se encuentran dentro del paquete *dao* se nombran adicionándoles como sufijo del nombre las siglas *DAO* o *DAOStoreProcedure* dependiendo de la fuente de datos origen. Para el paquete *impl* se les nombra igual que la interfaz que implementan y se le agrega como sufijo la palabra *Impl*. Ejemplo: *MT199DAO, MT199DAOImpl, GetConsecDAOStoreProcedure, GetConsecDAOStoreProcedureImpl*.

### 3.1.2. Modelo de componentes

Los diagramas de componentes describen los elementos físicos del sistema y sus relaciones. Muestran las opciones de realización incluyendo código fuente, binario y ejecutable. Los componentes representan todos los tipos de elementos del *software* que entran en la fabricación de aplicaciones informáticas [35].
El siguiente diagrama de componentes muestra las relaciones existentes entre los módulos del subsistema SLBTR con el resto de los componentes del sistema Quarxo y una breve descripción de las funciones que realizan cada uno de ellos:

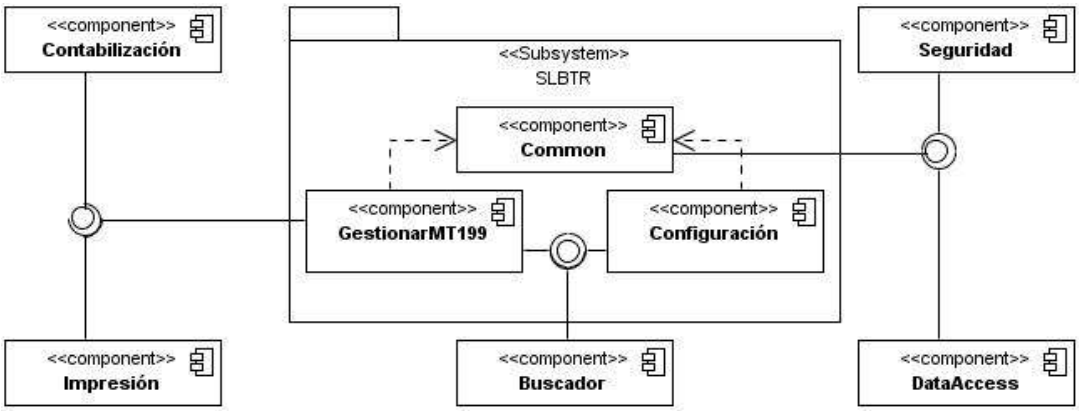



Figura 15 Diagrama de componentes del subsistema Mensajería SLBTR

Mediante el componente **Buscador** se ofrecen un grupo funcionalidades y clases que forman en su conjunto el motor de búsqueda del sistema, mediante una interfaz gráfica se solicitan los diferentes conceptos y criterios en dependencia del módulo donde se emplee. Para la correcta gestión de la información en el subsistema SLBTR se hace necesaria la búsqueda de:

- Mensajes para gestionar y enviar.
- Configuraciones para gestionar y habilitar.

Utilizando el componente **DataAccess** se acceden a las funciones necesarias para llevar a cabo la conexión a la base de datos.

**Impresión** consiste en un componente que tiene como función brindar las clases que contienen las funcionalidades mediante las cuales es posible imprimir determinada información. Específicamente el módulo GestionarMT199.

El componente **Seguridad** como su nombre lo indica es el encargado, entre otros aspectos, de mantener la seguridad en todo el sistema a través de la comprobación de las peticiones y sus permisos en dependencia del usuario que la realice, gestiona la información de los usuarios, roles y permisos necesarios para lograr la confiabilidad requerida en el sistema. Por lo que se hace necesaria la comunicación con este componente para garantizar la seguridad del subsistema SLBTR.

En el caso del componente **Contabilización** se proponen un grupo de clases necesarias para obtener determinados datos que son necesarios para la confección de los id de los mensajes, como son los consecutivos y la variable *WBAN_SLBTR,* que son ofrecidos por el nomenclador de este componente; por lo que se hace necesaria la integración con el mismo.

### 3.1.3. Diagrama de despliegue

En la siguiente figura se muestra el diagrama de despliegue del subsistema de Quarxo desarrollado.

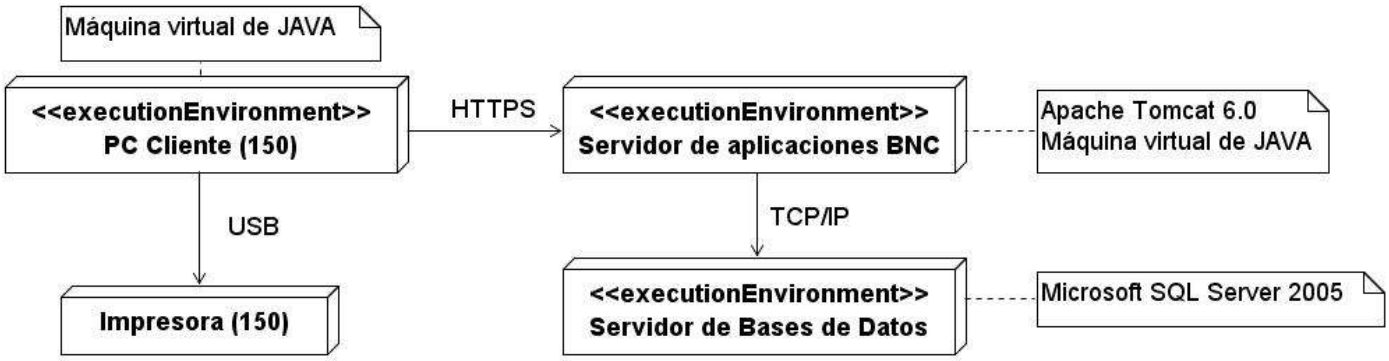



Figura 16 Diagrama de despliegue

### 3.1.4. Descripción de las clases y las funcionalidades

Para un mayor entendimiento de la implementación, en el Anexo # 6 se describen las clases más significativas de los módulos **GestionarMT199** y **Configuración** además de las principales clases dominio que se encargan de manejar el flujo de cada módulo.

### 3.1.5. Aspectos principales de la implementación

**Utilización del framework Spring Web Flow**

El *framework* Spring Web Flow (SWF) permite definir y gestionar los flujos de páginas dentro de una aplicación web. Para llevar a cabo la implementación de los casos de uso registrar y actualizar configuración, en el módulo Configuración, se utilizó dicho *framework*, dada la complejidad de los mismos y por las ventajas que este ofrece, ya que mantiene un flujo común que permite navegar a diferentes estados de vista (*view-state*) utilizando el mismo objeto o *command*, simplificando considerablemente la implementación.

Teniendo en cuenta que una configuración está formada por seis elementos y estos a su vez están compuestos por campos, se diseña un flujo general de vistas para conformar los datos de cada elemento y un subflujo que será utilizado para agregar campos a dichos elementos.

A continuación se hace una descripción de las operaciones de registro de una configuración de mensaje utilizando SWF (Se recomienda para un mejor entendimiento consultar primero el diagrama de colaboración correspondiente, en el Anexo # 4).

```
<?xml version="1.0" encoding="UTF-8"?>
<flow xmlns="http://www.springframework.org/schema/webflow"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/webflow
    http://www.springframework.org/schema/webflow/spring-webflow-2.0.xsd">

    <!-- Variables -->

    <var name="configuracionCommand" class="cu.uci.finixubnc.slbtr.common.domain.ConfiguracionMt199" />
    <var name="elementoCommand" class="cu.uci.finixubnc.slbtr.common.domain.Elemento" />
    <var name="campoCommand" class="cu.uci.finixubnc.slbtr.common.domain.Campo" />
```

Figura 17 Declaración de los *command* a utilizar en el flujo general para registrar una configuración

El flujo inicia con la declaración de las variables, que conformarán los *command* que serán actualizados a medida que se vayan avanzando por los distintos estados del flujo general. (Ver Figura 17). En este caso se tienen las siguientes variables:

- *configuracionCommand*: es el *command* que se utilizará por el flujo general y se va actualizando a medida que se recorran los estados de la configuración que se desea registrar o actualizar.
- *elementoCommand*: este *command* cambiará su referencia a medida que cambie de estado de vista, esta variable se encarga de ir actualizando cada *elemento* de la lista de elementos de la configuración. Siempre que se cambie de estado de vista, este *command* se cargará con el *elemento* que se corresponda a la mencionada vista.

Por otro lado, el *campoCommand* solo se referenciará como variable de entrada en el subflujo *registrarCampoSubflow-flow* y cargará en el momento que se salga de este.

```
<!-- Iniciar flujo -->

<on-start>
    <evaluate expression="registrarConfiguracionMultiAction.iniciarFlujo" />
</on-start>
```

Figura 18 Definición de la inicialización del flujo general para registrar una configuración

Al comenzar el flujo (Ver Figura 18) se ejecuta el método *iniciarFlujo(),* el cual colocará el objeto *configuracionCommand* dentro del objeto principal del flujo *(flowScope)* , para poder acceder a él desde todos los estados del flujo general.

```
<!-- Inicio de estados del flujo -->

<view-state id="registrarConfiguracion" view="regConfiguracionInicio"
    model="configuracionCommand">
    <transition on="siguiente" to="registrarRef_transa">
        <evaluate expression="registrarConfiguracionMultiAction.setElemento" />
    </transition>
    <transition on="cancelar" to="fin" />
</view-state>
```

Figura 19 Definición del estado inicial del flujo general: muestra la primera vista de secuencia de páginas

Luego, del estado *on-start* (Ver Figura 19)**,** el flujo pasa a su siguiente estado (Ver Figura 20), que en este caso es el primer estado de vista (*view-state* con id igual a "registrarConfiguración"), donde se muestra la primera página de la secuencia de páginas que procesarán para llevar a cabo el registro de la configuración. Este estado se va a orientar a la vista, y el modelo que va a recoger los datos de la configuración, llenados en la misma, será *configuracionCommand*. Cuando se recojan los datos de la primera parte de la configuración, se debe hacer clic en el botón siguiente, activando un evento con identificador *"*siguiente*"***.** A partir de ahí, ocurrirá una transición hacia diferentes vistas, no sin antes ejecutar el método *setElemento(),* el cual carga en el *elementoCommand* el e*lemento* que corresponde actualizar en la próxima vista del flujo.

La vista (*view*) de este estado (view-state con id igual a "registrarRef_transa") (Ver Figura 20**)** corresponde a un elemento de la configuración, por lo que a continuación el flujo recorrerá un estado de vista con el mismo diseño de flujo explicado anteriormente para cada elemento restante de la configuración. Estos estados son: *registrarRef_transa, registrarCod_solic, registrarId, registrarCod_causa, registrarNum_cuenta y registrarOtr_datos*. Solo van a variar el *id* del estado anterior y del siguiente de cada uno de ellos, además, los *command* solo se cargarán con los datos que corresponden actualizar en cada vista. El atributo *on-render* se ejecutará siempre antes de que se cree la vista, por lo que se hace necesario ejecutar los métodos *getNombreEstado()* para almacenar en el *flowScope* el *id* del estado en que se encuentra el flujo y *crearJSON()* para enviar desde el controlador un objeto *JSON* con el objetivo de actualizar un componente *DataGrid (tabla de Dojo Toolkit)* en la vista y mostrar los campos correspondiente al *elemento* que se está editando. El resto de las transiciones son de carácter organizativo y de gestión del propio *DataGrid***.**

```
<view-state id="registrarRef_transa" view="regConfiguracionRef_transa"
    model="elementoCommand">
    <on-render>
        <evaluate expression="registrarConfiguracionMultiAction.getNombreEstado" />
        <evaluate expression="registrarConfiguracionMultiAction.crearJSON" />
    </on-render>
    <transition on="siguiente" to="registrarCod_solic">
        <evaluate expression="registrarConfiguracionMultiAction.setElemento" />
    </transition>
    <transition on="agregar" to="registrarcampoSubFlow" />
    <transition on="eliminar" to="registrarRef_transa">
        <evaluate expression="registrarConfiguracionMultiAction.eliminar" />
    </transition>
    <transition on="subir" to="registrarRef_transa">
        <evaluate expression="registrarConfiguracionMultiAction.subir" />
    </transition>
    <transition on="bajar" to="registrarRef_transa">
        <evaluate expression="registrarConfiguracionMultiAction.bajar" />
    </transition>
    <transition on="anterior" to="registrarConfiguracion" />
    <transition on="cancelar" to="fin" />
</view-state>
```

Figura 20 Definición del estado del flujo donde registran los datos del primer elemento de la configuración.

Una vez que se ejecute la transición *"*agregar*"*, se llama al subflujo encargado de registrar los campos en un *elemento***.** Dicho subflujo se describe a continuación (Ver Figura 21):

```
<!-- subflujo para campos-->
<subflow-state id="registrarcampoSubFlow" subflow="registrarCampoSubflow-flow">
    <input name="campoCommand" />
    <transition on="end" to="${flowScope.idEstadoFlow}">
        <set name="flowScope.campoCommand" value="currentEvent.attributes.campoCommand" />
    </transition>
    <on-exit>
        <evaluate expression="registrarConfiguracionMultiAction.adicionarCampo" />
    </on-exit>
</subflow-state>
```

Figura 21 Definición del subflujo utilizado para registrar campos en un elemento.

En este estado (Ver Figura 21) antes de ejecutar el subflujo, se le agrega como parámetro de entrada *campoCommand*, luego que se llegue a la transición final dentro del subflujo, se regresará al estado anterior que invocó al subflujo almacenado en la variable de flujo *idEstadoFlow***,** que se crea o actualiza en el método *getNombreEstado(),* el cual se ejecuta antes de procesar una vista. Luego se actualiza el objeto *campoCommand* del flujo general obteniendo como resultado el objeto *campoCommand* que se editó dentro del subflujo. El atributo *on-exit* se ejecuta cuando se haya finalizado la llamada al subflujo adicionando el *campo* obtenido al *elemento* que se está registrando.

```
<view-state id="registrarFinalizar" view="regConfiguracionFinalizar"
    model="configuracionCommand">
    <transition on="aceptar" to="persistirConfiguracion" />
    <transition on="anterior" to="registrarOtr_datos">
        <evaluate expression="registrarConfiguracionMultiAction.setElemento" />
    </transition>
    <transition on="cancelar" to="fin" />
</view-state>
```

Figura 22 Definición del último estado correspondiente al registro de una configuración

En este estado (Ver Figura 22) se finaliza la recogida de los datos que conformarán la configuración y se mostrará una vista previa del XML de la configuración mensaje SLBTR MT199. Este XML se forma mediante la evaluación del método *vistaPrevia()* cuando se pasa a este estado, con la transición del estado de vista anterior (*view-state* con id igual a "registrarOtr_datos") (Ver Figura 23).

```
<transition on="siguiente" to="registrarFinalizar">
    <evaluate expression="registrarConfiguracionMultiAction.vistaPrevia" />
</transition>
```

Figura 23 Transición contenida dentro del estado correspondiente al registro del último elemento

La transición al estado "*aceptar*" (Ver Figura 22) evalúa el método *persistir()* mediante el cual que guarda la configuración en la base datos, ejecuta una transición hacia el estado final del flujo (Ver Figura 24) , dirigiendo el proceso a la página principal del subsistema.

```
<action-state id="persistirConfiguracion">
    <evaluate expression="registrarConfiguracionMultiAction.persistir" />
    <transition on="success" to="fin" />
</action-state>
<!-- finalizar flujo -->
<end-state id="fin"
    view="externalRedirect:servletRelative:../../common/home.htm">
</end-state>
```

Figura 24 Estado final del flujo de registro de una configuración

### 3.2. Pruebas

El principal objetivo de esta disciplina es evaluar la calidad del producto que se desarrolló. Se realiza a través de diferentes fases mediante la aplicación de pruebas concretas para validar que las suposiciones hechas en el diseño y que los requerimientos se estén cumpliendo satisfactoriamente [35].

Esto significa que se verifica el funcionamiento del producto como se diseñó y que los requerimientos han sido cumplidos satisfactoriamente.

#### 3.2.1. Pruebas unitarias de *software*

La prueba unitaria es la prueba enfocada a los elementos más pequeños que se puedan probar del *software* compuesto por un conjunto de técnicas experimentales para la búsqueda de fallos en el código fuente que determinan en cierto grado la calidad del producto. Es aplicable a los componentes representados en el modelo de implementación para verificar que los flujos de control y de datos están cubiertos, y que ellos funcionen como se espera [35].

Se aplicarán las técnicas de pruebas **caja blanca** para comprobar los métodos internos del subsistema y **caja negra** para probar que las funcionalidades del subsistema son operativas, que la entrada de datos se acepta educadamente y que se produce un resultado correcto, verificando que la integridad de la información externa no sufre ninguna variación.

#### 3.2.1.1. Aplicación de pruebas de caja blanca

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 [50].

Se identificaron dos formas de realizar las pruebas de caja blanca [47]:

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

Se seleccionó **pruebas de caja blanca dinámica**. Para su realización se utilizó el *framework* JUnit, el cual brinda un conjunto de bibliotecas que se integran fácilmente al IDE seleccionado: Eclipse. JUnit permite la automatización de las pruebas de aplicaciones en Java, tiene un conjunto de clases que el desarrollador puede utilizar para construir sus casos de prueba y ejecutarlos automáticamente. Además, mejora el diseño de implementación, haciéndolo flexible y probable [63].

Para la realización de la prueba se siguieron los siguientes pasos [63]:

- Crear un caso de prueba por cada clase implementada.
- Configurar en el caso de prueba, los ficheros que permiten comunicar las clases de las capas de presentación y lógica de negocio.
- Definir los métodos a probar dentro de cada caso de prueba, incluyendo los parámetros de entrada.
- Realizar pruebas a los métodos.

A continuación se evidencia como se le realizaron las pruebas de caja blanca a la clase controladora **CargarDatosMultiActionController**, localizada en el módulo GestionarMT199**,** demostrándose mediante la aplicación de la prueba al método **consultarConfiguracion**.

```
public ModelAndView consultarConfiguracion(HttpServletRequest request,
        HttpServletResponse response) {
    String id = request.getParameter("id");
    Map<String, String> model = new HashMap<String, String>();
    ConfiguracionMt199 c = configuracionFacade
            .cargarConfiguracion(new Integer(id));
    String xml = configuracionFacade.armarXml(c);
    model.put("xml_store", xml);
    return new ModelAndView("consultarConfiguracion", model);
```

Para realizar la prueba a dicho método primeramente se crea una clase que herede de la clase TestCase de JUnit, en este caso dicha clase se nombra Prueba, en la cual inicialmente, se declara un objeto de la clase que se desea probar, mediante el cual se invoca al método que se desea verificar; y dos objetos *mock*, que simularán los parámetros de tipo HttpServlet (*request* y *response*) que recibe dicho método.

```
public class Prueba extends TestCase {

    public static final String KEY = "slbtr";
    private CargarDatosMultiActionController cargarDatosMultiActionController;

    private MockControl controlHttpServlet;
    private HttpServletRequest mockhttpServletRequest;

    private MockControl controlHttpResponse;
    private HttpServletResponse mockhttpServletResponse;

    private MockControl controlHttpSession;
    private HttpSession mockHttpSession;
```

En el método **setUp()** se define la configuración que va a tener el entorno de prueba, para este caso mediante el archivo *slbtr-contex.xml* se accederá al contexto de la aplicación para poder hacer la llamada a los métodos de la clase que se quiere probar. Este contexto permitirá en la prueba utilizar las referencias a las clases que se encuentran en otras capas de la aplicación. También se inicializan las variables *mocks* para poder simular las variables de *HttpServletRequest* y *HttpServletResponse* que constituye los parámetros de entrada del método que se desea probar.

```
public void setUp() throws Exception {
    cargarDatosMultiActionController = new CargarDatosMultiActionController();
    ApplicationContext context = new ClassPathXmlApplicationContext(
            "classpath:cu/uci/finixubnc/slbtr/common/web/mvc/controller/slbtr-context.xml");
    // Set componente facade
    ComponenteManagerImpl componenteManagerImpl = (ComponenteManagerImpl) context
            .getBean("componenteManager");
    ComponenteFacadeImpl componenteFacadeImpl = new ComponenteFacadeImpl();
    componenteFacadeImpl.setComponenteManager(componenteManagerImpl);
    cargarDatosMultiActionController.setComponenteFacade(componenteFacadeImpl);
    // Set configuracion facade
    ConfiguracionManagerImpl configuracionManagerImpl = (ConfiguracionManagerImpl) context
            .getBean("configuracionManager");
    ConfiguracionFacadeImpl configuracionFacadeImpl = new ConfiguracionFacadeImpl();
    configuracionFacadeImpl.setConfiguracionManager(configuracionManagerImpl);
    cargarDatosMultiActionController.setConfiguracionFacade(configuracionFacadeImpl);

    controlHttpServlet = MockControl.createControl(HttpServletRequest.class);
    mockhttpServletRequest = (HttpServletRequest) controlHttpServlet.getMock();
    controlHttpResponse = MockControl.createControl(HttpServletResponse.class);
    mockhttpServletResponse = (HttpServletResponse) controlHttpResponse.getMock();
    controlHttpSession = MockControl.createControl(HttpSession.class);
    mockHttpSession = (HttpSession) controlHttpSession.getMock();
    super.setUp();
```

Durante la realización de la prueba, se preparan los parámetros de pruebas que le son necesarios al método **consultarConfiguracion**. Estos parámetros serán pasados por el *mockHttpServletRequest* y posteriormente se verifica la condición de prueba a través del método **assertNotNull()** el cual comprueba si el resultado de la llamada al método devuelve un valor distinto de **null**

```
public final void testConsultarConfiguracion() {
    mockhttpServletRequest.getParameter("id");
    controlHttpServlet.setReturnValue("91");
    controlHttpServlet.replay();
    ModelAndView pag = cargarDatosMultiActionController
            .consultarConfiguracion(mockhttpServletRequest,
                    mockhttpServletResponse);
    System.out.println(pag);
    assertNotNull(pag);
}
```

#### 3.2.1.1.1. Resultado de las pruebas de caja blanca

El resultado de la prueba realizada a todos los métodos de la clase **CargarDatosMultiActionController** fue satisfactorio, como se muestra en la Figura 25. Evidenciándose por el color verde mostrado en la barra de la parte superior izquierda (en caso de existir fallos la barra tendría color rojo). En dicho escenario se ejecutaron 19 casos de pruebas, donde el 100 % tuvo 0 errores (*errors* en inglés) y 0 fallos (*failures* en inglés).

Es importante aclarar, que durante toda la fase de implementación se utilizó JUnit para evaluar si el funcionamiento de cada uno de los métodos de las clases se comportaba como se esperaba, lo permitió que se detectaran errores tempranamente y que fueran corregidos por los desarrolladores.

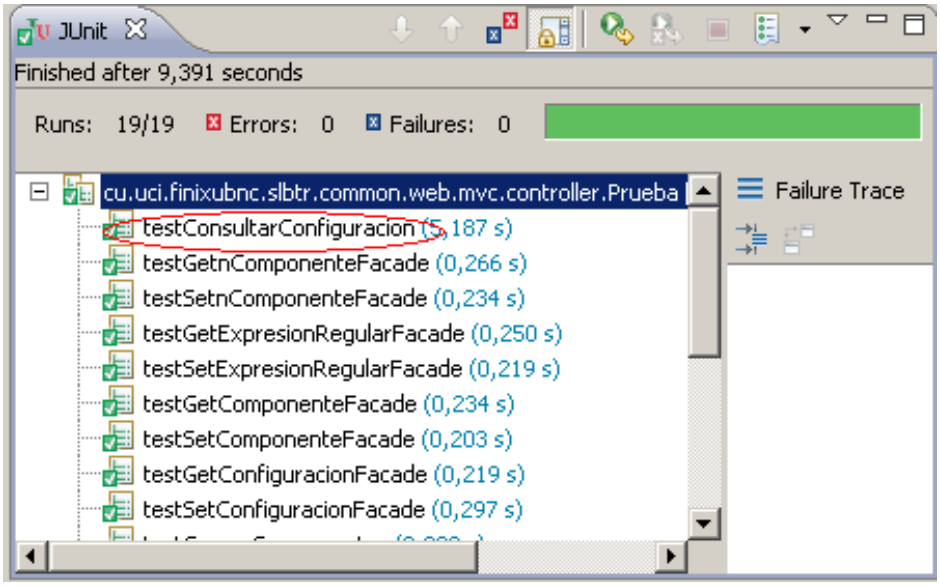



Figura 25 Paleta de resultados JUnit

#### 3.2.1.2. Aplicación de pruebas de caja negra

Las pruebas de caja negra, también denominadas pruebas de comportamiento, se centran en los requisitos funcionales del *software*. Permiten obtener conjuntos de condiciones de entrada que ejerciten completamente todos los requisitos funcionales de un programa. La prueba de caja negra no es una alternativa a las técnicas de prueba de caja blanca, se trata de un enfoque complementario que intenta descubrir diferentes tipos de errores que los métodos de caja blanca [50]. Con este tipo de pruebas se intenta encontrar: funciones incorrectas o ausentes, errores de interfaz, errores en estructuras de datos o en accesos a las bases de datos externas, errores de rendimiento y errores de inicialización y terminación. Algunas técnicas en este tipo pruebas son las que se muestran a continuación [50]:

- **Partición de equivalencia:** divide el campo de entrada en clases de datos que tienden a ejercitar determinadas funciones del *software*.
- **Análisis de valores límites:** prueba la habilidad del programa para manejar datos que se encuentran en los límites aceptables.
- **Grafos de causa-efecto:** permite al encargado de la prueba validar complejos conjuntos de acciones y condiciones.

Para la realización de las pruebas que se realizaron a los módulos se utilizó la técnica **partición de equivalencia** pues esta es una de las técnicas más efectiva ya que permite comprobar los valores válidos e inválidos de todas las entradas existentes en el *software*, además permite la reducción del número total de casos de prueba que hay que desarrollar.

Para verificar que la aplicación se comporta según los requerimientos establecidos por el cliente, se diseñan casos de pruebas usando el método de caja negra, los cuales pueden ser consultados en el Anexo # 7.

#### 3.2.1.2.1. Resultado de las pruebas de caja negra

Los resultados obtenidos a través de la realización de las pruebas expuestas anteriormente, fueron satisfactorios desde el punto de vista interno y funcional, dado que los módulos mantuvieron un correcto comportamiento ante las diferentes situaciones en que se probaron.

El subsistema fue probado por el Centro Nacional de Calidad para Soluciones Informáticas (CALISOFT) del MIC, donde fue sometido a 3 iteraciones de pruebas funcionales, de carga y estrés. Como constancia de los satisfactorios resultados obtenidos, dicha entidad emitió un acta de liberación de *software* certificando la calidad del producto desarrollado (consultar Anexo #1)

Una vez liberada la aplicación fue presentada ante el cliente (BNC), el cual luego de haberla probado durante aproximadamente un año y tres meses estuvo satisfecho con el producto entregado. Concluyéndose, con la puesta funcionamiento del subsistema Mensajería el día 11 de Junio de 2012 en el BNC.

## 3.3. Validación de las variables de investigación

La investigación realizada plantea como idea a defender que: “Si se desarrolla el subsistema Mensajería SLBTR para el sistema Quarxo, se podrá reducir el tiempo y el consumo de recursos materiales cuando se requiere gestionar los mensajes SLBTR, que intercambia el Banco Nacional de Cuba con el sistema bancario nacional”. A continuación se evalúan las variables tiempo y recursos materiales con el uso del subsistema Mensajería SLBTR de Quarxo en el BNC.

### 3.3.1. Reducción de tiempo

La variable tiempo se analiza haciendo referencia a la capacidad de adaptación a los cambios que ocurren en el estándar de los mensajes SLBTR por parte del BNC, tomando como indicador el tiempo invertido para registrar o actualizar los cambios en la configuración de un tipo de mensaje.

La disminución de esta variable implicará necesariamente mayor eficiencia y productividad, a partir de que se dedicará menos tiempo en solucionar estas operaciones, y se agilizan los procesos que esperan por su realización.

Se consideraron los siguientes pasos para registrar o actualizar la configuración de un tipo de mensaje SLBTR, utilizando SABIC y el Quarxo. En la siguiente tabla se resumen:

| SABIC (SLBTR) | Quarxo (Subsistema Mensajería SLBTR) |
|---|---|
| **Recursos necesarios** | **Recursos necesarios** |
| 1. **Programador con conocimientos en FoxPro**.<br>2. **Código fuente** de la aplicación en FoxPro<br>3. **Editor para programar** en FoxPro.<br>4. **Servidor de pruebas**. | **1. Usuario de Quarxo con conocimientos sobre la configuración de un mensaje SLBTR.** |
| | **Observaciones** |
| | No es necesario el acceso al código fuente de la aplicación, ni conocimientos de programación, además de que no son necesarias herramientas de desarrollo. |
| **Pasos** | **Pasos** |
| 1. **Abrir código fuente** del proyecto en **editor FoxPro.**<br>2. **Diseñar interfaz visual** del formulario en **FoxPro**.<br>**3. Programar restricciones** de **componentes visuales.**<br>4. **Programar** acciones del **negocio** y **conformar estructura** del mensaje SLBTR **dentro del código fuente.**<br>5. **Compilar** proyecto.<br>6. **Probar nueva versión** de la aplicación **en servidor de pruebas**.<br>7. **Detener aplicación en uso y sustituir aplicación por nueva versión**. | 1. **Acceder al módulo configuración** del subsistema Mensajería SLBTR de la aplicación web Quarxo.<br>2. **Registrar o actualizar datos** de la estructura de la configuración.<br>3. **Activar nueva configuración**. |
| | **Observaciones** |
| | El usuario del sistema podrá en tiempo de ejecución gestionar la configuración de los mensajes SLBTR, utilizando el subsistema sin necesidad de programar. Los cambios realizados se hacen efectivos en las funcionalidades correspondientes a la gestión de los mensajes SLBTR una vez activada la configuración. |

**1. Tiempo invertido para registrar o actualizar los cambios en la configuración de un tipo de mensaje SLBTR**

Para medir este indicador se seleccionó el 100 % de los tipos de mensajes SLBTR que se gestionan en el departamento Préstamos y depósitos del BNC.

Para realizar los pasos de registro o actualización de configuraciones de mensajes SLBTR gestionados por SABIC, se utilizó un trabajador con experiencia en programación FoxPro y con conocimientos de gestión de mensajes SLBTR del departamento de Sistema Automatizados del BNC, en lo adelante programador SABIC. Y para realizar las acciones mencionadas anteriormente, utilizando el subsistema Mensajería SLBTR de Quarxo, se seleccionó un operador del departamento de Préstamos y depósitos con conocimientos nulos en programación y con habilidades básicas para interactuar con una aplicación web, en lo adelante usuario Quarxo.

A continuación se presenta el escenario a analizar y los resultados.

**Escenario 01. Registrar configuración de tipo de mensaje SLBTR MT199**

| Escenario | 01.01 | Registrar configuración de tipo de mensaje SLBTR MT199 con código de solicitud 06. | | | |
|---|---|---|---|---|---|
| Paso | Programador SABIC Tiempo invertido | | Paso | Usuario Quarxo Tiempo invertido | |
| 1 | 1 | | 1 | 1 | |
| 2 | 35 | | 2 | 16 | |
| 3 | 90 | | 3 | 1 | |
| 4 | 245 | | Total | 18 | |
| 5 | 5 | | En los dos casos el tiempo se mide en **minutos** | | |
| 6 | 1440 | | | | |
| 7 | 60 | | | | |
| Total | 1876 | | | | |

| Escenario | 01.02 | Registrar configuración de tipo de mensaje SLBTR MT199 con código de solicitud 10. | | | |
|---|---|---|---|---|---|
| Paso | Programador SABIC Tiempo invertido | | Paso | Usuario Quarxo Tiempo invertido | |
| 1 | 1 | | 1 | 1 | |
| 2 | 38 | | 2 | 17 | |
| 3 | 97 | | 3 | 1 | |
| 4 | 255 | | Total | 19 | |
| 5 | 5 | | En los dos casos el tiempo se mide en **minutos** | | |
| 6 | 1440 | | | | |
| 7 | 60 | | | | |
| Total | 1896 | | | | |

| Escenario | 01.03 | Registrar configuración de tipo de mensaje SLBTR MT199 con código de solicitud 13. | | | |
|---|---|---|---|---|---|
| Paso | Programador SABIC Tiempo invertido | | Paso | Usuario Quarxo Tiempo invertido | |
| 1 | 1 | | 1 | 1 | |
| 2 | 38 | | 2 | 16 | |
| 3 | 97 | | 3 | 1 | |
| 4 | 255 | | Total | 18 | |
| 5 | 5 | | En los dos casos el tiempo se mide en **minutos** | | |
| 6 | 1440 | | | | |
| 7 | 60 | | | | |
| Total | 1896 | | | | |

| Escenario | 01.04 | Registrar configuración de tipo de mensaje SLBTR MT199 con código de solicitud 21. | | | |
|---|---|---|---|---|---|
| Paso | Programador SABIC Tiempo invertido | | Paso | Usuario Quarxo Tiempo invertido | |
| 1 | 1 | | 1 | 1 | |
| 2 | 52 | | 2 | 22 | |
| 3 | 130 | | 3 | 1 | |
| 4 | 302 | | Total | 24 | |
| 5 | 5 | | En los dos casos el tiempo se mide en **minutos** | | |
| 6 | 1440 | | | | |
| 7 | 60 | | | | |
| Total | 1990 | | | | |

| Escenario | 01.05 | Registrar configuración de tipo de mensaje SLBTR MT199 con código de solicitud 22. | | | |
|---|---|---|---|---|---|
| Paso | Programador SABIC Tiempo invertido | | | Paso | Usuario Quarxo Tiempo invertido |
| 1 | 1 | | | 1 | 1 |
| 2 | 55 | | | 2 | 22 |
| 3 | 137 | | | 3 | 1 |
| 4 | 312 | | | **Total** | 24 |
| 5 | 5 | | | En los dos casos el tiempo se mide en **minutos** | |
| 6 | 1440 | | | | |
| 7 | 60 | | | | |
| **Total** | 2010 | | | | |

| Escenario | 01.06 | Registrar configuración de tipo de mensaje SLBTR MT199 con código de solicitud 23. | | | |
|---|---|---|---|---|---|
| Paso | Programador SABIC Tiempo invertido | | | Paso | Usuario Quarxo Tiempo invertido |
| 1 | 1 | | | 1 | 1 |
| 2 | 30 | | | 2 | 15 |
| 3 | 79 | | | 3 | 1 |
| 4 | 229 | | | **Total** | 17 |
| 5 | 5 | | | En los dos casos el tiempo se mide en **minutos** | |
| 6 | 1440 | | | | |
| 7 | 60 | | | | |
| **Total** | 1844 | | | | |

| Escenario | 01.07 | Registrar configuración de tipo de mensaje SLBTR MT199 con código de solicitud 24. | | | |
|---|---|---|---|---|---|
| Paso | Programador SABIC Tiempo invertido | | | Paso | Usuario Quarxo Tiempo invertido |
| 1 | 1 | | | 1 | 1 |
| 2 | 63 | | | 2 | 25 |
| 3 | 155 | | | 3 | 1 |
| 4 | 338 | | | **Total** | 27 |
| 5 | 5 | | | En los dos casos el tiempo se mide en **minutos** | |
| 6 | 1440 | | | | |
| 7 | 60 | | | | |
| **Total** | 2062 | | | | |

| Escenario | 01.08 | Registrar configuración de tipo de mensaje SLBTR MT199 con código de solicitud 27. | | | |
|---|---|---|---|---|---|
| Paso | Programador SABIC Tiempo invertido | | | Paso | Usuario Quarxo Tiempo invertido |
| 1 | 1 | | | 1 | 1 |
| 2 | 30 | | | 2 | 15 |
| 3 | 79 | | | 3 | 1 |
| 4 | 229 | | | **Total** | 17 |
| 5 | 5 | | | En los dos casos el tiempo se mide en **minutos** | |
| 6 | 1440 | | | | |
| 7 | 60 | | | | |
| **Total** | 1844 | | | | |

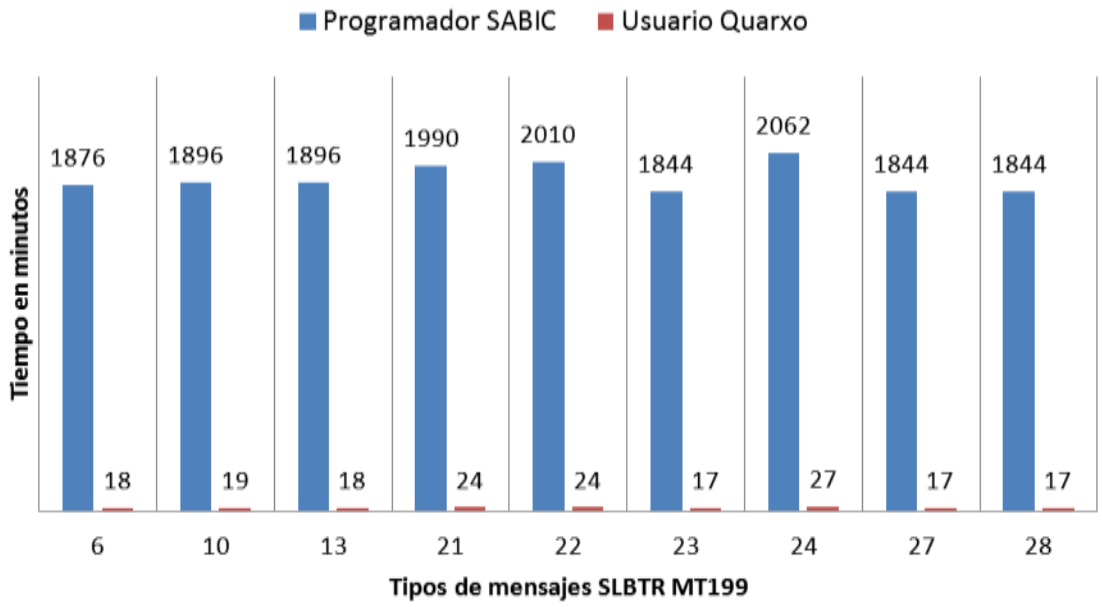
| Escenario | 01.09 | Registrar configuración de tipo de mensaje SLBTR MT199 con código de solicitud 28. |
|---|---|---|

| Paso | Programador SABIC<br>Tiempo invertido |
|---|---|
| 1 | 1 |
| 2 | 30 |
| 3 | 79 |
| 4 | 229 |
| 5 | 5 |
| 6 | 1440 |
| 7 | 60 |
| **Total** | 1844 |

| Paso | Usuario Quarxo<br>Tiempo invertido |
|---|---|
| 1 | 1 |
| 2 | 15 |
| 3 | 1 |
| **Total** | 17 |

En los dos casos el tiempo se mide en **minutos**

| Tiempo invertido para registrar configuración de un tipo de mensaje SLBTR MT199. (en minutos) | | |
|---|---|---|
| **Mensaje SLBTR MT199 con código de solicitud:** | **Programador SABIC** | **Usuario Quarxo** |
| 06 | 1876 | 18 |
| 10 | 1896 | 19 |
| 13 | 1896 | 18 |
| 21 | 1990 | 24 |
| 22 | 2010 | 24 |
| 24 | 1844 | 17 |
| 27 | 2062 | 27 |
| 28 | 1844 | 17 |
| Tiempo promedio (minutos) | 1844 | 17 |




A partir de las mediciones anteriormente realizadas se demuestra que con el uso del Subsistema SLBTR se reduce el tiempo invertido para registrar o actualizar los cambios en la configuración de un tipo de mensaje SLBTR, a partir de la reducción del número de actividades y la disminución de la complejidad de

las mismas. Actividades, que cuando se usaba SABIC demoraban aproximadamente un promedio de 30 horas, se reducen a pocos minutos usando el subsistema Mensajería SLBTR.

### 3.3.2. Reducción de consumo de recursos materiales

Uno de los inconvenientes que posee el SABIC instalado en el BNC es que no provee funcionalidades para consultar y controlar los mensajes registrados o enviados por cada operador, provocando que se tenga que incurrir en gasto de papel por concepto de impresión de los datos como resultado de la operación, para que un supervisor revise si los datos están correctos y mantenga un histórico de los mensajes enviados en formato papel.

Con la utilización del subsistema Mensajería SLBTR el gasto de papel provocado por SABIC se reduce totalmente, pues los supervisores desde la aplicación pueden revisar que los datos de los mensajes a enviar estén correctos, hacer correcciones en caso de que sea necesario, emitir reportes en formato digital (PDF), que pueden ser enviados por correo electrónico a los interesados en el BNC o guardados en la base de datos para constancia y respaldo de la información manejada.

Por lo expuesto anteriormente, quedan validadas las variables consideradas en la investigación, demostrándose que la solución desarrollada reduce el tiempo y el consumo de recursos materiales cuando se requieren gestionar los mensajes SLBTR, que intercambia el BNC con el sistema bancario nacional.

**Conclusiones parciales**

- Se obtuvo el subsistema Mensajería SLBTR de Quarxo, desarrollado con tecnologías libres, provisto de elementos de seguridad y cumpliendo estándares. La reutilización de componentes desarrollados por el grupo de arquitectura del proyecto Quarxo posibilitó reducir el tiempo en la fase implementación.
- La solución exhibe valor técnico a partir de la utilización del *framework* Spring como elemento fundamental, el cual brinda recursos potentes que permitieron un manejo más fácil de los procesos de gestión de la configuración de los mensajes debido la cantidad de flujos complejos y reutilizables que se requieren para ejecutar estas operaciones.
- El subsistema fue probado mediante pruebas de caja blanca y caja negra, y validado utilizando pruebas de aceptación con el cliente.
- Se evaluaron las variables consideradas en la investigación, obteniéndose como resultado que la solución desarrollada reduce el tiempo y el consumo de recursos materiales cuando se requiere gestionar los mensajes SLBTR que intercambia el BNC con el sistema bancario nacional.

# CONCLUSIONES GENERALES

Una vez culminado el trabajo y como resultado del mismo se logró alcanzar el objetivo general propuesto, tras cumplir todas las tareas trazadas a inicio de la investigación:

- A partir del estudio de los sistemas para la gestión de mensajes SLBTR se concluyó que no es factible costear cualquiera de los sistemas informáticos ya existentes a nivel internacional, pues no implementan el estándar cubano de mensaje SLBTR, además, de que no cumplen con la política bancaria cubana.
- Se descartó el sistema informático desarrollado en Cuba para tales fines, por las limitadas funcionalidades de gestión que posee. Evidenciándose la necesidad de desarrollar un sistema informático web multiplataforma para la gestión de la mensajería SLBTR y la administración de los formatos asociados a dichos mensajes.
- El proceso de desarrollo de *software* fue guiado por la metodología RUP, cumpliendo con los elementos que dicha metodología propone y logrando una efectiva ingeniería de *software*.
- Las tecnologías seleccionadas para el desarrollo de la solución están acordes a los requerimientos del cliente, a las políticas del centro y a los estándares internacionales.
- Se obtuvo el subsistema Mensajería SLBTR de Quarxo, probado mediante pruebas de caja blanca y caja negra, y validado utilizando pruebas de aceptación con el cliente. La reutilización de componentes desarrollados por el grupo de arquitectura del proyecto Quarxo posibilitó reducir el tiempo en la fase implementación.
- Se evaluaron las variables consideradas en la investigación, obteniéndose como resultado que la solución desarrollada reduce el tiempo y el consumo de recursos materiales cuando se requiere gestionar los mensajes SLBTR que intercambia el BNC con el sistema bancario nacional.

# RECOMENDACIONES

- Desarrollar un mecanismo que permita adicionar configuraciones al subsistema en un fichero XML.

# REFERENCIAS BIBLIOGRÁFICAS


[1] ALUR, D.; J. CRUPI*, y otros. Core J2EE patterns: best practices and design strategies*. Prentice Hall PTR, 2003. p. 0131422464

[2] BANCO CENTRAL DE CUBA. *El sistema bancario cubano*, Banco Central de Cuba, 2012. [2012]. Disponible en: http://www.bc.gov.cu/Espanol/sist_bancario.asp]

[3] BANCO CENTRAL DE CUBA. *Resolución 48 de Junio 2009. Banco Central de Cuba. Sistema de Liquidación Bruta en Tiempo Real.*, Banco Central de Cuba, 2009. [2012]. Disponible en: http://www.bc.gov.cu/Manual/Cap%C3%ADtulo%2004.%20Regulaciones%20de%20Cobros%20y%20pagos/4.08%20-%20Resoluci%C3%B3n%20No.%2048%20de%204%20de%20junio%20de%202009.%20Sistema%20de%20Liquidaci%C3%B3n%20Bruta%20en%20Tiempo%20Real%20(SLBTR)..pdf]

[4] BANCO CENTRAL DE CHILE. *Sistemas Sistemas de Pagos en Chile. Avances y desafíos pendiente*, 2011.

[5] BANCO DE MEXICO. *SIPEI*, 2012. [2012]. Disponible en: http://www.banxico.org.mx/sistemas-de-pago/informacion-general/sistemas-de-pago-de-alto-valor/sistema-pagos-electronicos-in.html]

[6] BANK FOR INTERNATIONAL SETTLEMENTS. *Real-time gross settlement systems*, Committee on Payment and Settlement Systems. Bank for International Settlements, 1997. [Disponible en: http://www.bis.org/publ/cpss22.pdf]

[7] BANKSERV. *Global Funds Exchange (GFX)*, 2011. [Disponible en: http://www.bankserv.com/fedwire.html]

[8] BASS, L.; P. CLEMENTS*, y otros. Software architecture in practice*, Addison-Wesley Longman Publishing Co., Inc., 2003. [Disponible en: http://dl.acm.org/citation.cfm?id=773239]

[9] BHASIN, N. K. *Innovations in E Banking - Real Time Gross Settlement. in Ninth AIMS International Conference on Management* 2012. [Disponible en: http://www.aims-international.org/aims9/aims9cd/pdf/P9402-Final.pdf]

[10] CABALLERO, P. M. y M. E. SIMÓN. *Sistema de Liquidación interbancaria del Sistema Bancario Cubano*. *COFIN Habana. Revista de la Facultad de Contabilidad y Finanzas de la Universidad de la Habana*, 1997.

[11] CEREZAL, L. *La contabilidad en una nueva tecnología.*, DISAIC, 2002. [Disponible en: http://www.betsime.disaic.cu/secciones/tec_feb_02.htm]

[12] CEREZAL, L. y J. TORRES. *Enriqueciendo el sistema contable en el Banco Central de Cuba*, Centro de Información Bancaria y Económica (CIBE), 2006. [1]. Disponible en: http://www.bc.gov.cu/anteriores/RevistaBCC/2006/No1-2006/Documentos/Enriqueciendo_el_sistema_contable_en_el_banco_Central_de_Cuba-RevBCC-No1-2006.pdf]

[13] CONSEJO DE ESTADO DE LA REPÚBLICA DE CUBA. *Decreto Ley No. 172 de 1997*, Consejo de Estado de la República de Cuba, 1997. [Disponible en: http://www.bc.gov.cu/Espanol/Leyes/LEY172.pdf]

[14] CHOPRA, V.; S. LI*, y otros. Professional Apache Tomcat 6*. Wrox, 2011. p. 1118058771

[15] DHINGRA, P. y T. SWANSON. *Microsoft SQL Server 2005*. Sams, 2007. p. 0672329220

[16] DIXON, M. *"A single CASE environment for teaching and learning". Proceedings of the 9th annual SIGCSE conference on Innovation and technology in computer science education. Software Engineering Institute*. Leeds, UK, 2004.

[17] DOJO TOOLKIT. *Dojo Toolkit 1.3*, 2012. [Disponible en: http://dojotoolkit.org]

[18] ECLIPSE FOUNDATION. *Eclipse - The Eclipse Foundation open source community website.*, 2012. [Disponible en: http://www.eclipse.org]

[19] ELMASRI, R.; S. B. NAVATHE*, y otros. Fundamentos de sistemas de bases de datos*. Addison-Wesley, 2002. p. 8478290516

[20] ENCICLOPEDIA DEL INVERSOR. *Trans-european Automated Real-time Gross settlement Express Transfer system (TARGET)*, 2012. 2012.

[21] ERIKSSON, P. *Design Patterns and Code Structures using JEE*, 2011. [Disponible en: http://www.nada.kth.se/utbildning/grukth/exjobb/rapportlistor/2011/rapporter11/eriksson_per_11045.pdf]

[22] FLANAGAN, D. *JavaScript: the definitive guide*. O'Reilly Media, 2006. p. 0596101996

[23] FUNDTECH. *PAYplus RTGS*, 2012. [2012]. Disponible en: http://www.fundtech.com/products/payments/global-payments/global-payplus-services-platform/high-care-rtgs/]

[24] FURST, K.; W. LANG*, y otros. Technological Innovation in Banking and Payments: Industry Trends and Implications for Banks*, 2012. [Disponible en: http://ro.uow.edu.au/aabfj/vol5/iss4/4/]

[25] GAMMA, E.; R. HELM*, y otros. Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Pub Co, 1995. p.

[26] GÓMEZ, O. T.; P. P. R. LÓPEZ*, y otros.* Criterios de selección de metodologías de desarrollo de software *Ind. data*, 2010, 13(2).

[27] GONZALEZ DORIA, H. *Las Metricas de Software y su Uso en la Region*. Departamento de Ingeniería en Sistemas Computacionales, Escuela de Ingeniería. Ecuador, Universidad de las Américas Puebla, 2010. p.

[28] GUPTA, U. G. y W. COLLINS The impact of information systems on the efficiency of banks: an empirical investigation *Industrial Management & Data Systems*, 1997, 97(1): 10-16.

[29] HIBERNATE. *Hibernate Framework*, 2012. [Disponible en: http://www.hibernate.org/]

[30] HO, C. *Spring Web Flow and JSF*, 2012. [Disponible en: http://www.springerlink.com/content/t855057101053423/]

[31] IBM. *Montran Real Time Gross Settlement System 3.0*, 2012. [2012]. Disponible en: http://www-304.ibm.com/partnerworld/gsd/solutiondetails.do?solution=45264&expand=true&lc=en]

[32] IEEE. *IEEE Standard Glossary of Software Engineering Terminology*, 1990. [2012]. Disponible en: http://standards.ieee.org/findstds/standard/610.12-1990.html]

[33] IEEE ARCHITECTURE WORKING GROUP IEEE Recommended Practice for Architectural Description of Software-Intensive Systems. IEEE Standard IEEE Std 1471-2000 *IEEE Standards Association*, 2000.

[34] ISO 15022. *ISO 15022*, 2012. [Disponible en: http://www.iso15022.org/ISO15022XML/General/MessageStandardsEvolution.pdf]]

[35] JACOBSON, I.; G. BOOCH*, y otros. El proceso unificado de desarrollo de software*, Madrid, España : Pearson, 2000. [Disponible en: http://en.scientificcommons.org/6960494]

[36] LADD, S. *Expert Spring MVC and Web Flow*. Dreamtech Press, 2006. p. 8181284437

[37] LARMAN, C. *UML y Patrones*. Segunda Edición. Prentice Hall, 1999. p. 970-17-0261-1

[38] LARRÁN JORGE, M.; M. DE LOS REYES*, y otros.* La banca por Internet como innovación tecnológica en el sector bancario *Investigaciones europeas de dirección y economía de la empresa*, 2007, 13(2): 145-153.

[39] LÓPEZ, J. Control de versiones con Subversion *Todo linux: la revista mensual para entusiastas de GNU/LINUX*, 2008, (90): 42-45.

[40] LORENZ, M. y J. KIDD Object-oriented software metrics *Journal of Systems and Software*, 1994, 44(2): 147-154.

[41] MICROSOFT CORPORATION. *VFP Team. A Message to the Community. Microsoft announced that there would be no VFP 10.*, 2009. [2012]. Disponible en: http://msdn.microsoft.com/en-us/vfoxpro/bb308952.aspx]

[42] MONTRAN. *Montran RTGS*, 2011. [Disponible en: http://www.montran.com/products/rtgs/index.html]

[43] MURPHY, G. C.; M. KERSTEN*, y otros. How are Java software developers using the Elipse IDE?*, 2006. [23]. Disponible en: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1657944]

[44] ORACLE. *Core J2EE Patterns - Composite View*, Oracle, 2012. [2012]. Disponible en: http://java.sun.com/blueprints/corej2eepatterns/Patterns/CompositeView.html]

[45] ORACLE. *Core J2EE Patterns - Data Access Object*, 2012. [2012]. Disponible en: http://java.sun.com/blueprints/corej2eepatterns/Patterns/DataAccessObject.html]
[46] OUB. *Selects Fundtech's PAYplus RTGS Solution for Payments Processing; Contract Marks Fundtech's First Deal in Singapore*, Overseas Union Bank 2011.
[47] PATTON, R. *Software testing*, 2. Sams Publishing, 2005. [2]. Disponible en: http://dl.acm.org/citation.cfm?id=1051270]
[48] PERAGO. *Perago RTGS*, 2010. [Disponible en: http://www.sia.eu/Perago/Engine/RAServePG.php/P/251210070409]
[49] PILATO, C. M.; B. COLLINS-SUSSMAN*, y otros. Version control with subversion*. O'Reilly Media, 2008. p. 0596510330
[50] PRESSMAN, R. Ingeniería del Software: Un Enfoque Práctico *Mc Graw*, 2005.
[51] PROYECTO QUARXO Modernización Sistema del Banco Nacional de Cuba *Documento de Arquitectura*, 2009.
[52] REYES, R. A. M. *Los Patrones como un Medio del Diseño Orientado a Objetos*, 2008. [Disponible en: http://www.revistaupiicsa.20m.com/Emilia/RevMayAgo04/Machorro1.pdf]
[53] RUMBAUGH, J.; I. JACOBSON*, y otros. UML. El lenguaje de modelado unificado. Manual de referencia*, Addison Wesley, 2000.
[54] RUSSELL, M. *Dojo: the definitive guide*. O'Reilly Media, Inc., 2008. p. 0596516487
[55] S. SHAKEEL y V. N. SASTRY. *Basics of Structured Financial Messaging System (SFMS)*, Institute for Development & Research in Banking Tecnology (IDRBT), 2012. [2012]. Disponible en: http://www.mpf.org.in/pdf/Basics%20of%20SFMS%20Standards.pdf]
[56] SHALLOWAY, A. y J. TROTT. *Design Patterns Explained: A New Perspective on Object-Oriented Design (Software Patterns Series)*, Addison-Wesley Professional, 2004.
[57] SIBANC. *Manual SLBTR-Bancos*, 2012.
[58] SOFTWARE INTEGRATORS. *STAR Liquidity Management*, 2011. [Disponible en: http://www.software-integrators.co.uk/star-liquidity-management]
[59] SOMMERVILLE, I. *Ingeniería del software*. Pearson Educación, 2005. p. 8478290745
[60] SPRING SOURCE. *Spring Framework*, 2012. [Disponible en: http://www.springsource.org/spring-framework]
[61] SUN MICROSYSTEMS. *JavaEE*, 2011. [Disponible en: http://java.sun.com/javaee/5/docs/tutorial/doc/]
[62] SWIFT CORPORATION. *Society for Worldwide Interbank Financial Telecommunication*, 2012. [Disponible en: http://www.swift.com/about_swift/company_information/index.page?lang=es]
[63] TAHCHIEV, P.; F. LEME*, y otros. JUnit in action*. Manning Publications Co., 2010. p. 1935182021
[64] VANETTI, R. Strategic innovation in payments systems: What are the next big things? *Journal of Payments Strategy & Systems*, 2010, 4(1): 17-25.
[65] VISUAL PARADIGM. *Visual Paradigm 6.4*, 2012. [Disponible en: http://www.visual-paradigm.com/product/vpuml/]
[66] W3C. *XHTML 1.0 The Extensible HyperText Markup Language (Second Edition)*, 2011. [Disponible en: http://www.w3.org/TR/xhtml1/]
[67] XBRL CORPORATION. *Extensible Business Reporting Language (XBRL) 2.1*, 2008. [Disponible en: http://www.xbrl.org/Specification/XBRL-RECOMMENDATION-2003-12-31+Corrected-Errata-2008-07-02.htm#_1]

# GLOSARIO DE TÉRMINOS

**Campo:** Se refiere campo como al atributo que forma parte del valor de un elemento dentro de un mensaje XML, describe generalmente una propiedad o atributo de un objeto del dominio del negocio.

**Caso de prueba:** Especifica una forma de probar el sistema, incluyendo la entrada o resultado con la que se ha de probar y las condiciones bajo las que hay que probarse.

**Command:** Es la representación específica de la información con la cual el sistema opera. Se limita a lo relativo de la vista y su controlador facilitando las presentaciones visuales complejas. Los *command* también pueden operar con más datos no relativos a la presentación, haciendo uso integrado de otras lógicas de negocio y de datos afines con el sistema modelado.

**Elemento:** Se refiere elemento a un bloque o atributo que forma parte de un mensaje XML.

**JSP:** (*Java Server Page* en inglés) Tecnología orientada a crear páginas web con programación en Java que permite mezclar HTML estático con HTML generado dinámicamente.

**Módulo:** cada módulo es una parte del sistema, que se instala y funciona por separado, entrelazándose con otros módulos con los que intercambia información.

**MS-DOS:** Microsoft Disk Operating System, Sistema operativo de disco de Microsoft.

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

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

**ORM:** (*Object Relacional Mapping* en inglés) Persistencia automática y transparente de objetos de una aplicación en una base de datos relacional utilizando metadatos que describen la correspondencia entre el objeto y las tablas de la base de datos.

**Plugins:** Un *plugin* es un módulo de *hardware* o *software* que adiciona una característica o un servicio específico a un sistema existente.

**Servlet:** Su función principal es proveer páginas web dinámicas y personalizadas, utilizando para este objetivo el acceso a bases de datos, flujos de trabajo y otros recursos.

**Sistema bancario:** Conjunto de instituciones que permiten el desarrollo de todas aquellas transacciones (entre personas, empresas y organizaciones) que impliquen el uso de recursos financieros.

# ANEXOS

**Anexo # 1. Acta de liberación de los módulos y subsistemas emitida por CALISOFT**

UCI Universidad de las Ciencias Informáticas

## Acta de Liberación de Productos Software

**Fecha de liberación:** 20 de diciembre de 2010.

**Emitida a favor de:** SAGEB.

**1. Datos del producto**

| Artefacto | Versión | Estado final | Cantidad Iteraciones | Tipos de pruebas realizadas |
|---|---|---|---|---|
| Gestionar Acuerdo | V1.0 | | 3 | Funcionales, Carga y Estrés |
| Documentos de Embarque | V1.0 | | 3 | Funcionales, Carga y Estrés |
| Clientes Jurídicos | V1.0 | | 3 | Funcionales, Carga y Estrés |
| Mensajería SLBTR | V1.0 | | 3 | Funcionales, Carga y Estrés |
| Préstamos | V1.0 | | 3 | Funcionales, Carga y Estrés |
| Gestionar Transferencias | V1.0 | | 3 | Funcionales, Carga y Estrés |
| Vencimientos | V1.0 | | 3 | Funcionales, Carga y Estrés |
| Ejercicio Contable. | V1.0 | | 3 | Funcionales, Carga y Estrés |
| Carta de Remesa | V1.0 | | 3 | Funcionales, Carga y Estrés |
| Emisión de CC | V1.0 | | 3 | Funcionales, Carga y Estrés |
| Negociación de CC | V1.0 | | 3 | Funcionales, Carga y Estrés |
| Seguridad | V1.0 | | 3 | Funcionales, Carga y Estrés |
| Gestionar Chequeras | V1.0 | | 3 | Funcionales, Carga y Estrés |
| Cheque | V1.0 | | 3 | Funcionales, Carga y Estrés |

1

Figura 26 Parte 1 Acta de liberación CALISOFT

Universidad de las Ciencias Informáticas

## Acta de Liberación de Productos Software

| | | | | |
|---|---|---|---|---|
| Depósitos | V1.0 | | 3 | Funcionales, Carga y Estrés |
| Discrepancias de DE | V1.0 | | 3 | Funcionales, Carga y Estrés |
| Personas Autorizadas | V1.0 | | 3 | Funcionales, Carga y Estrés |
| Plan de Cuentas | V1.0 | | 3 | Funcionales, Carga y Estrés |
| Gestionar Banco | V1.0 | | 3 | Funcionales, Carga y Estrés |
| Medios de Comunicación | V1.0 | | 3 | Funcionales, Carga y Estrés |
| Permisos Sobre Cuentas | V1.0 | | 3 | Funcionales, Carga y Estrés |
| Libros Contables | V1.0 | | 3 | Funcionales, Carga y Estrés |
| Gestionar Cta. de Clientes | V1.0 | | 3 | Funcionales, Carga y Estrés |
| Tasa de Cambio | V1.0 | | 3 | Funcionales, Carga y Estrés |
| Cuentas de Banco | V1.0 | | 3 | Funcionales, Carga y Estrés |
| Gestionar Cta. Concepto | V1.0 | | 3 | Funcionales, Carga y Estrés |
| Transacciones Generales | V1.0 | | 3 | Funcionales, Carga y Estrés |
| Sobregiro Autorizado | V1.0 | | 3 | Funcionales, Carga y Estrés |
| Reservación de Fondos | V1.0 | | 3 | Funcionales, Carga y Estrés |
| Capacidad Financiera | V1.0 | | 3 | Funcionales, Carga y Estrés |

Delvis Echeverría Perez
Nombre y Apellidos
Responsable Calisoft

Lissett Díaz Mesa
Nombre y Apellidos
Responsable Proyecto

Figura 27 Parte 2 Acta de liberación CALISOFT

## Anexo # 2. Requisitos no funcionales

### Requisitos no funcionales del subsistema

***Funcionalidad***

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***

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.
3. El menú de navegación estará disponible en todas las páginas.
4. En caso de que los resultados de las consultas tengan más de 10 coincidencias, estos se mostrarán de forma paginada 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***

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

***Seguridad***

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

8. El sistema se implementará usando la plataforma JEE.
9. El sistema estará basado en un estilo arquitectónico en capas.

***Interfaz de usuario***

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

## Anexo # 3. Descripción de los casos de usos del sistema

### Descripción del CUS "Gestionar configuración de mensaje SLBTR"

| **Caso de uso:** | **Gestionar configuración de mensaje SLBTR** |
|---|---|
| **Actores:** | Usuario |
| **Resumen:** | El caso de uso se inicia cuando el usuario selecciona la opción “Gestionar configuración de mensaje SLBTR”. El sistema registra, actualiza, consulta, elimina cambia de estado y permite realizar búsquedas de configuraciones de mensajes SLBTR. El caso de uso termina cuando el usuario acepta realizar la operación. |
| **Precondiciones:** | • El usuario debe estar autenticado con los permisos necesarios. |
| **Referencias** | RF 14.1, RF 14.2, RF 14.3, RF 14.4, RF 14.5, RF 14.6 |
| **Prioridad** | Alta |

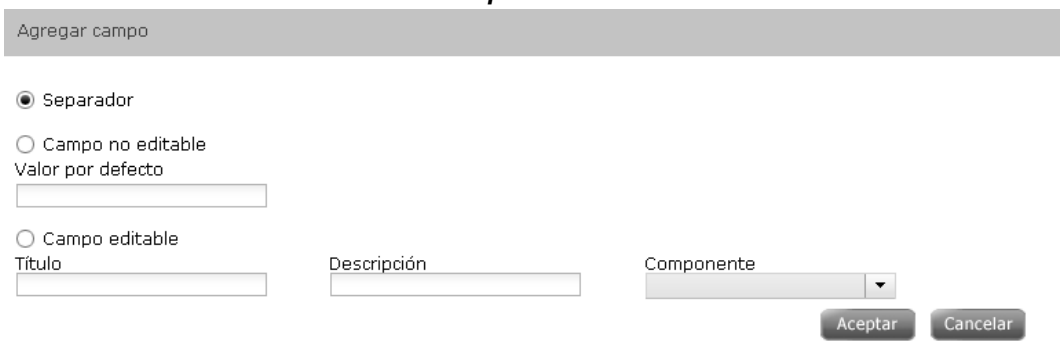
| **Flujo normal de eventos** | |
|---|---|
| **Sección “Gestionar configuración de mensaje SLBTR”** | |
| **Acción del actor** | **Respuesta del sistema** |
| 1. El usuario selecciona la opción “Gestionar configuración de mensaje SLBTR”. | 2. El sistema muestra las opciones de registrar, actualizar, consultar, cancelar y buscar una configuración de mensajes SLBTR |
| 3. El usuario puede seleccionar una de las opciones siguientes:<br>a. Registrar configuración de mensaje SLBTR.<br>Ver sección “Registrar configuración de mensaje SLBTR”.<br>b. Actualizar configuración de mensaje SLBTR.<br>Ver sección “Actualizar configuración de mensaje SLBTR”. (Anexo # 3)<br>c. Consultar configuración de mensaje SLBTR.<br>Ver sección “Consultar configuración de mensaje SLBTR”. (Anexo # 3)<br>d. Eliminar configuración de mensaje SLBTR.<br>Ver sección “Eliminar configuración de mensaje SLBTR”. (Anexo # 3)<br>e. Buscar configuración de mensaje SLBTR.<br>Ver sección “Buscar configuración de mensaje SLBTR”. (Anexo # 3)<br>e. Cambiar estado de configuración de mensaje SLBTR.<br>Ver sección “Cambiar estado de configuración de mensaje SLBTR”. (Anexo # 3) | |
| **Sección “Registrar configuración de mensaje SLBTR”** | |
| **Acción del actor** | **Respuesta del sistema** |
| | 1. El sistema muestra la interfaz para ***registrar el inicio de la configuración.*** |
| 2. El usuario introduce los datos en los campos de la interfaz y presiona el botón “siguiente”.<br>a. En caso de presionar el botón “cancelar”<br>Ver sección “Cancelar operación” | 3. El sistema valida los datos.<br>a. En caso de datos incorrectos.<br>Ver sección “Datos incorrectos y/o campos vacíos”<br>4. El sistema muestra la interfaz para ***registrar referencia de transacción (REF_TRANSA) de la configuración.*** |

| | |
|---|---|
| 5. El usuario gestiona la lista de campos que aparecen en la interfaz y presiona el botón “siguiente”.<br>a. En caso seleccionar un campo en la lista de campos y presionar el botón “agregar campo”<br>Ver sección “Agregar campo”<br>b. En caso seleccionar un campo en la lista de campos y presionar el botón “eliminar campo”<br>Ver sección “Eliminar campo”<br>c. En caso seleccionar un campo en la lista de campos y presionar el botón “subir campo”<br>Ver sección “Subir campo”<br>d. En caso seleccionar un campo en la lista de campos y presionar el botón “bajar campo”<br>Ver sección “Bajar campo”<br>e. En caso seleccionar un campo en la lista de campos y presionar el botón “cancelar”<br>Ver sección “Cancelar operación”<br>f. En caso de presionar el botón “Anterior”<br>Ver sección “Anterior” | 6. Ídem al paso 3.<br>7. El sistema muestra la interfaz para ***registrar el código de solicitud (COD_SOLIC) de la configuración.*** |
| 8. Ídem al paso 5 | 9. Ídem al paso 3.<br>10. El sistema valida los datos y muestra la interfaz para ***registrar el identificador (ID) de la configuración.*** |
| 11. Ídem al paso 5 | 12. Ídem al paso 3.<br>13. El sistema valida los datos y muestra la interfaz para ***registrar el código de causa (COD_CAUSA) de la configuración.*** |
| 14. Ídem al paso 5 | 15. Ídem al paso 3.<br>16. El sistema valida los datos y muestra la interfaz para ***registrar el número de cuenta (NUM_CUENTA) de la configuración.*** |
| 16. Ídem al paso 5 | 17. Ídem al paso 3.<br>18. El sistema valida los datos y muestra la interfaz para ***registrar elemento “otros datos” (OTR_DATOS) de la configuración***. |
| 19. Ídem al paso 5 | 20. Ídem al paso 3.<br>21. El sistema genera una ***vista previa de la estructura de la configuración.*** |
| 22. El usuario presiona el botón “aceptar”.<br>a. En caso seleccionar un campo en la lista de campos y presionar el botón “cancelar”<br>Ver sección “Cancelar operación”<br>b. En caso de presionar el botón “Anterior”<br>Ver sección “Anterior” | 23. El sistema registra la configuración y finaliza la sección del CU. |

***Prototipo de interfaz***


| **Flujos alternos** | |
|---|---|
| **Sección "Datos incorrectos"** | |
| **Acción del actor** | **Respuesta del sistema** |
| | 1. El sistema señala los datos incorrectos y muestra el mensaje "Datos incorrectos"<br>2. Regresa al paso del flujo normal de eventos. |
| **Sección "Cancelar operación"** | |
| **Acción del Actor** | **Respuesta del sistema** |
| | 1. El sistema cancela la operación y finaliza el caso de uso. |
| **Sección "Anterior"** | |
| **Acción del actor** | **Respuesta del sistema** |
| | 1. El sistema regresa al paso anterior en el flujo normal de eventos. |
| **Sección "Subir campo"** | |
| **Acción del actor** | **Respuesta del sistema** |
| | 1. El sistema sube el campo seleccionado en la lista de campos.<br>2. El sistema actualiza la lista de campos<br>3. El sistema regresa al flujo normal de eventos. |

| Sección “Bajar campo” | |
|---|---|
| **Acción del actor** | **Respuesta del sistema** |
| | 1. El sistema sube el campo seleccionado en la lista de campos.<br>2. El sistema actualiza la lista de campos<br>3. El sistema regresa al flujo normal de eventos. |
| **Sección “Eliminar campo”** | |
| **Acción del actor** | **Respuesta del sistema** |
| | 1. El sistema elimina el campo seleccionado en la lista de campos.<br>2. El sistema actualiza la lista de campos<br>3. El sistema regresa el flujo normal de eventos. |
| **Sección “Agregar campo”** | |
| **Acción del actor** | **Respuesta del sistema** |
| | 1. El sistema muestra la interfaz para registrar campo correspondiente al elemento de la configuración seleccionado. |
| 2. El usuario selecciona el tipo de campo y llena los datos y presiona el botón “aceptar”<br>a. En caso de presionar el botón “cancelar” regresa al flujo normal de eventos. | 3. El sistema valida los datos.<br>a. En caso de datos incorrectos.<br>Ver sección “Datos incorrectos”<br>4. El sistema agrega el campo en la lista de campos.<br>5. El sistema actualiza la lista de campos<br>6. El sistema regresa al flujo normal de eventos. |
| ***Prototipo de Interfaz*** | |



**Continuación de las secciones de la descripción del caso de uso CU Gestionar configuración de mensaje SLBTR**

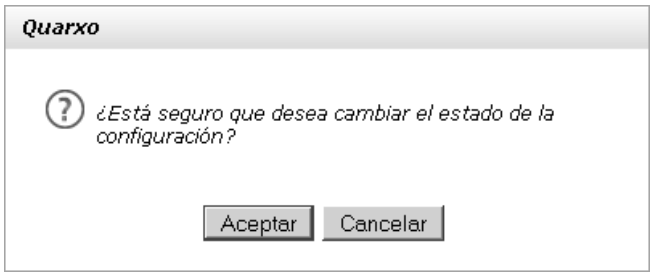
| | |
|---|---|
| **Caso de Uso:** | CU Gestionar configuración de mensaje SLBTR |
| **Actores:** | Usuario |
| **Resumen:** | El caso de uso se inicia cuando el usuario selecciona la opción “Gestionar configuración de mensaje SLBTR”. El sistema registra, actualiza, consulta, elimina cambia de estado y permite realizar búsquedas de configuraciones de mensajes SLBTR. El caso de uso termina cuando el usuario acepta realizar la operación. |
| **Precondiciones:** | • El usuario debe estar autenticado con los permisos necesarios. |
| **Referencias** | RF 14.1, RF 14.2, RF 14.3, RF 14.4, RF 14.5, RF 14.6 |
| **Prioridad** | Alta |

| **Sección “Actualizar configuración de mensaje SLBTR”** | |
|---|---|
| **Acción del actor** | **Respuesta del sistema** |
| | 1. El sistema muestra la interfaz para ***actualizar el inicio de la configuración.*** |
| 2. El usuario introduce los datos en los campos de la interfaz y presiona el botón “siguiente”.<br>a. En caso de presionar el botón “cancelar”<br>Ver sección “Cancelar operación” | 3. El sistema valida los datos.<br>a. En caso de datos incorrectos.<br>Ver sección “Datos incorrectos y/o campos vacíos”<br>4. El sistema muestra la interfaz para ***actualizar referencia de transacción (REF_TRANSA) de la configuración.*** |
| 5. El usuario gestiona la lista de campos que aparecen en la interfaz y presiona el botón “siguiente”.<br>a. En caso seleccionar un campo en la lista de campos y presionar el botón “agregar campo”<br>Ver sección “Agregar campo”<br>b. En caso seleccionar un campo en la lista de campos y presionar el botón “eliminar campo”<br>Ver sección “Eliminar campo”<br>c. En caso seleccionar un campo en la lista de campos y presionar el botón “subir campo”<br>Ver sección “Subir campo”<br>d. En caso seleccionar un campo en la lista de campos y presionar el botón “bajar campo”<br>Ver sección “Bajar campo”<br>e. En caso seleccionar un campo en la lista de campos y presionar el botón “cancelar”<br>Ver sección “Cancelar operación”<br>f. En caso de presionar el botón “Anterior”<br>Ver sección “Anterior” | 6. Ídem al paso 3.<br>7. El sistema muestra la interfaz para ***actualizar el código de solicitud (COD_SOLIC) de la configuración.*** |
| 8. Ídem al paso 5 | 9. Ídem al paso 3.<br>10. El sistema valida los datos y muestra la interfaz para ***actualizar el identificador (ID) de la configuración.*** |
| 11. Ídem al paso 5 | 12. Ídem al paso 3.<br>13. El sistema valida los datos y muestra la interfaz para ***registrar el código de causa (COD_CAUSA) de la configuración.*** |
| 14. Ídem al paso 5 | 15. Ídem al paso 3.<br>16. El sistema valida los datos y muestra la interfaz para ***actualizar el número de cuenta (NUM_CUENTA) de la configuración.*** |
| 16. Ídem al paso 5 | 17. Ídem al paso 3.<br>18. El sistema valida los datos y muestra la interfaz para ***actualizar el elemento “otros datos” (OTR_DATOS) de la configuración***. |
| 19. Ídem al paso 5 | 20. Ídem al paso 3.<br>21. El sistema genera una ***vista previa de la estructura de la configuración.*** |

| | |
|---|---|
| 22. El usuario presiona el botón “aceptar”.<br>a. En caso seleccionar un campo en la lista de campos y presionar el botón “cancelar”<br>Ver sección “Cancelar operación”<br>b. En caso de presionar el botón “Anterior”<br>Ver sección “Anterior” | 23. El sistema actualiza la configuración y finaliza la sección del CU. |

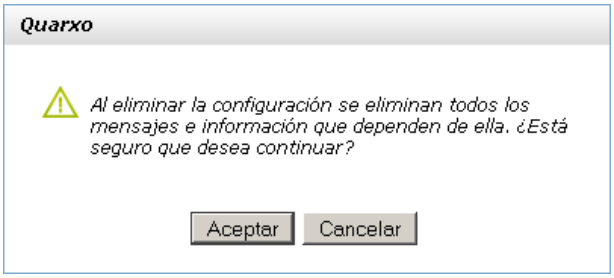
| ***Sección “Cambiar estado de configuración de mensaje SLBTR”*** | |
|---|---|
| **Acción del actor** | **Respuesta del sistema** |
| 1. El usuario selecciona la configuración y presiona el botón “Cambiar estado”. | 2. El sistema muestra el mensaje de confirmación |
| 3. El usuario presiona el botón “aceptar”.<br>a. En caso presionar el botón “cancelar”<br>Ver sección “Cancelar operación” | 4. El sistema cambia el estado de la configuración.<br>6. El sistema muestra un mensaje donde se notifica el éxito de la operación y el estado en que queda la configuración<br>5. Finaliza la sección del CU. |

***Prototipo de Interfaz***




| ***Sección “Cambiar estado de configuración de mensaje SLBTR”*** | |
|---|---|
| **Acción del actor** | **Respuesta del sistema** |
| 1. El usuario selecciona la configuración y presiona el botón “eliminar”. | 2. El sistema muestra el mensaje de confirmación |
| 3. El usuario presiona el botón “aceptar”.<br>a. En caso presionar el botón “cancelar”<br>Ver sección “Cancelar operación” | 4. El sistema elimina la configuración.<br>6. El sistema muestra un mensaje donde se notifica el éxito de la operación y el estado en que queda la configuración<br>5. Finaliza la sección del CU. |

***Prototipo de Interfaz***



## Anexo # 4. Diagramas de colaboración.

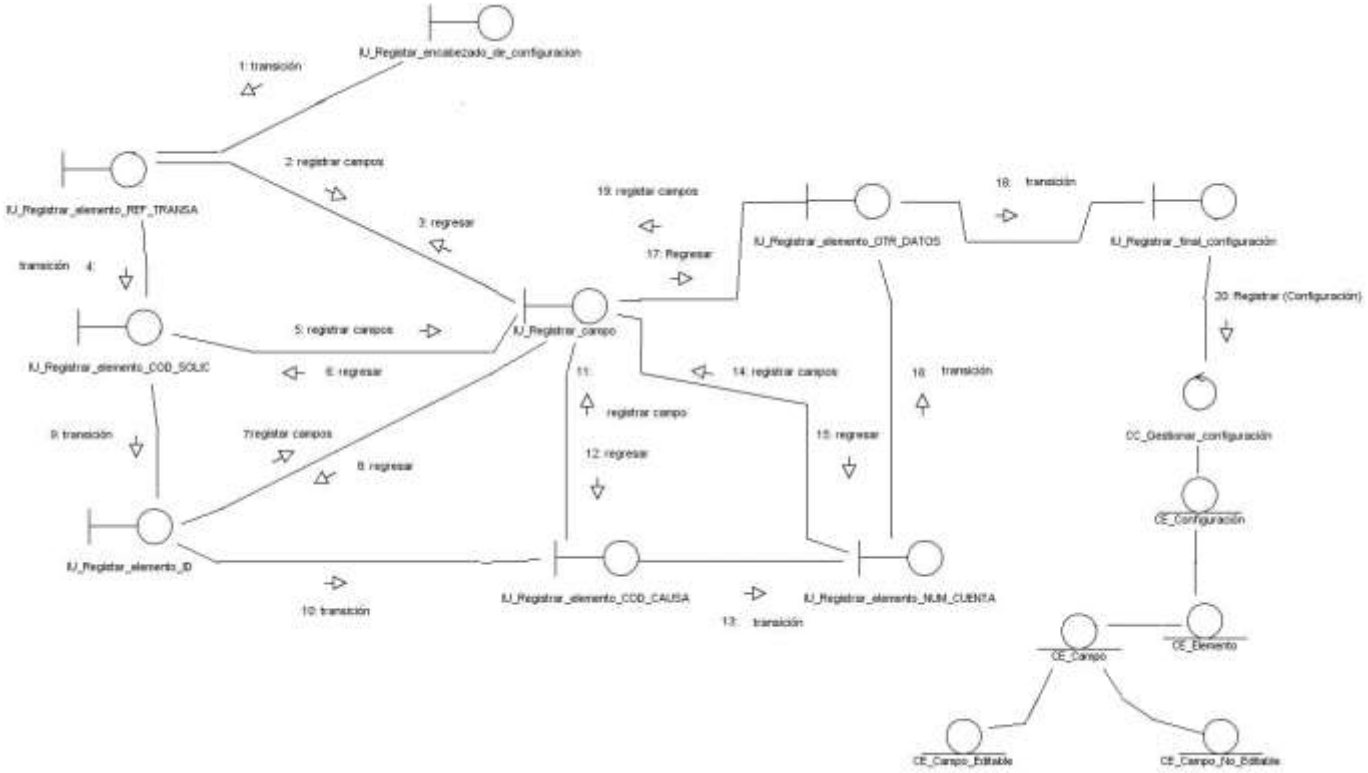



Figura 28 Diagrama de colaboración “CU Registrar configuración de mensaje SLBTR”

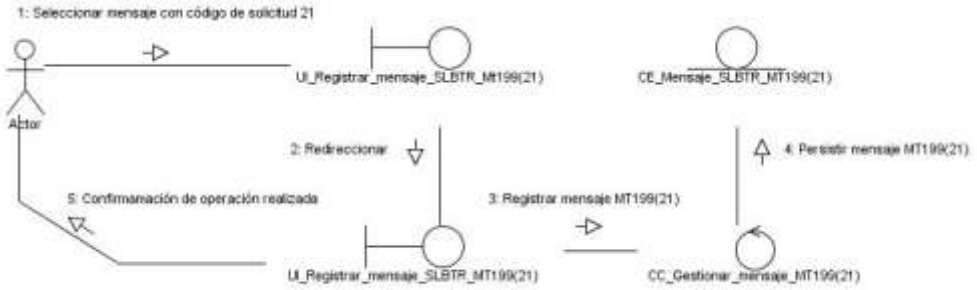



Figura 29 Diagrama de colaboración “CU Registrar mensaje SLBTR con código de solicitud 21”

## Anexo # 5. Diagramas de clases del diseño.

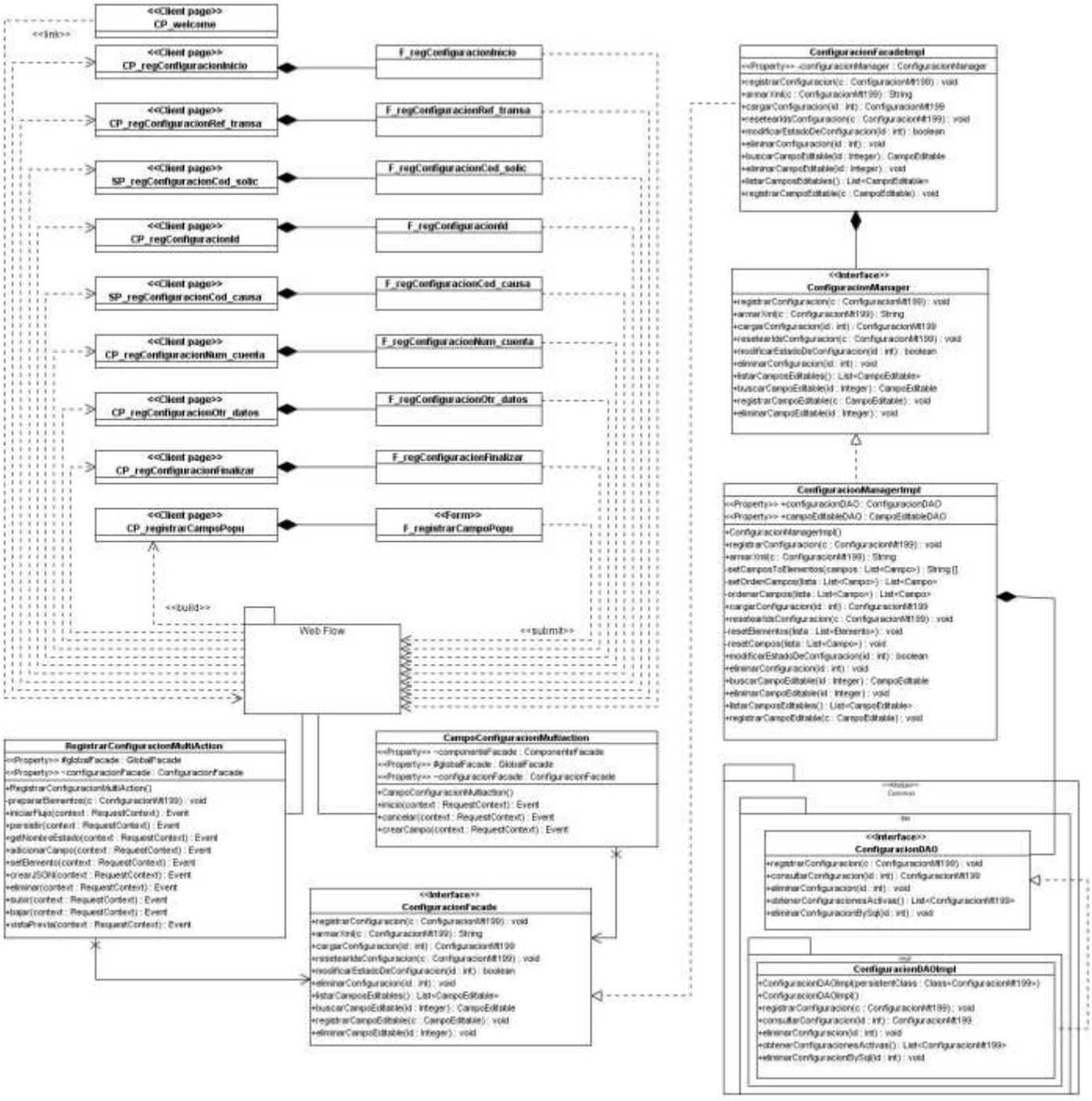



Figura 30 Diagrama de clases del diseño “CU Registrar configuración de mensaje SLBTR”

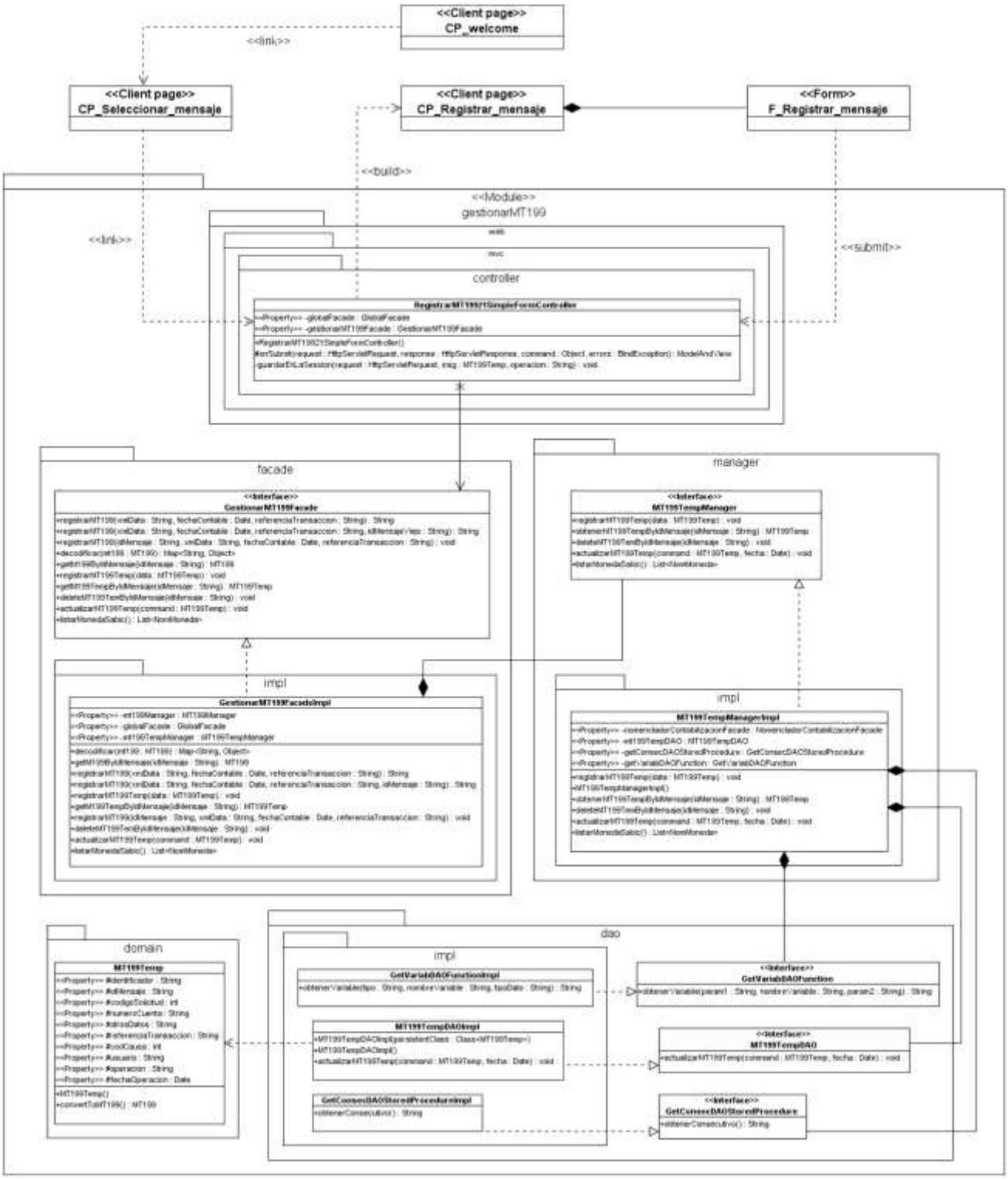



Figura 31 Diagrama de clases del diseño “CU Registrar mensaje SLBTR con código de solicitud 21”

## Anexo # 6. Descripción de las clases y las funcionalidades.

Tabla 2 Clase RegistrarConfiguracionMultiAction del módulo Configuración

| **Nombre**: RegistrarConfiguracionMultiAction | |
|---|---|
| **Tipo de clase**: Controladora | |
| **Atributo** | **Tipo** |
| configuracionFacade | ConfiguracionFacade |
| globalFacade | GlobalFacade |
| **Para cada responsabilidad**: | |
| **Nombre**: | **Descripción**: |
| RegistrarConfiguracionMultiAction() | Método constructor de la clase. |
| getConfiguracionFacade() | Método para acceder al atributo privado configuracionFacade. |
| setConfiguracionFacade(ConfiguracionFacade configuracionFacade) | Método para actualizar el atributo privado configuracionFacade de la clase. |
| getGlobalFacade() | Método para acceder al atributo privado globalFacade. |
| setGlobalFacade(GlobalFacade globalFacade) | Método para actualizar el atributo privado globalFacade de la clase. |
| prepararElementos(ConfiguracionMt199 c) | Método que prepara un objeto configuración adicionándole los 6 elementos del mensaje MT199 ya que el usuario no modifica directamente estos datos. |
| iniciarFlujo(RequestContext context) | Método mediante se inicializa el flujo y se crea el objeto configuración y se pone dentro del contexto del flujo. |
| persistir(RequestContext context) | Mediante este método se persiste el objeto configuración que ha recogido sus datos a través del flujo. |
| getNombreEstado(RequestContext context) | Método que almacenará el id del estado en el que el flujo se encuentre en la variable de flujo idEstadoFlow, con el objetivo de poder regresar a este estado una vez que se salga del subflujo registrarCampoSubflow-flow. |
| adicionarCampo(RequestContext context) | Método que adiciona al Elemento que esté en ese momento almacenado en la variable de flujo elementoCommand el campo que se recoja dentro del subflujo registrarCampoSubflow-flow. |
| setElemento(RequestContext context) | Mediante este método se le asigna a la variable de flujo elementoCommand el Elemento que se desea actualizar en el flujo. |
| crearJSON(RequestContext context) | Compone un objeto de JSON para aplicar a un DataGrid que representará los campos que van componiendo un Elemento determinado. |
| eliminar(RequestContext context) | Método que elimina un campo determinado del Elemento que se esté editando en ese momento dentro del flujo. |
| subir(RequestContext context) | Sube un nivel el campo seleccionado dentro de la lista de campos del Elemento. |
| bajar(RequestContext context) | Baja un nivel el campo seleccionado dentro de la lista de campos del Elemento. |
| vistaPrevia(RequestContext context) | Método mediante se crea una vista previa del mensaje XML el cual se almacena en la variable de flujo xml_store. |

Tabla 3 Clase de dominio ConfiguracionMt199 del módulo Configuración

| **Nombre**: ConfiguracionMt199 | |
|---|---|
| **Tipo clase**: Modelo | |
| **Atributo** | **Tipo** |
| idConfiguracion | Integer |
| nombre | String |
| descripcion | String |
| estado | Boolean |
| tipo | String |
| tipo_operación | String |
| fecha | Date |

Tabla 4 Clase de dominio MT199 del módulo GestionarMT199

| **Nombre**: MT199 | |
|---|---|
| **Tipo clase**: Modelo | |
| **Atributo** | **Tipo** |
| identificador | String |
| idMensaje | String |
| codigoSolicitud | Integer |
| numeroCuenta | String |
| otrosDatos | String |
| referenciaTransaccion | String |
| codCausa | Integer |

Tabla 5 Clase CargarDatosMultiActionController del módulo GestionarMT199

| **Nombre**: CargarDatosMultiActionController | |
|---|---|
| **Tipo de clase**: Controladora | |
| **Atributo** | **Tipo** |
| gestionarMT199Facade | GestionarMT199Facade |
| globalFacade | GlobalFacade |
| reporteFacade | ReporteFacade |
| **Para cada responsabilidad**: | |
| **Nombre**: | **Descripción**: |
| cargarCodONE(HttpServletRequest request, HttpServletResponse response) | Crea un objeto JSON listando los códigos ONE de las instituciones. |
| cargarPaises(HttpServletRequest request, HttpServletResponse response) | Devuelve un JSON con el listado de países disponibles en el sistema. |
| eliminarMT199Temp(HttpServletRequest request, HttpServletResponse response) | Elimina un mensaje determinado tomando como parámetro el id del mismo. |
| obtenerTipoMensajeMT199Temp(HttpServletRequest request, HttpServletResponse response) | Devuelve el código de solicitud de un mensaje determinado mediante el parámetro idMensaje. |
| resolverVistaConsultarMT199Temp(HttpServletRequest request, HttpServletResponse response) | Obtiene el mensaje temporal mediante el parámetro idMensaje el cual se decodifica y se muestra en su vista correspondiente para consultar. |
| resolverVistaConsultarMT199(HttpServletRequest request, HttpServletResponse response) | Obtiene el mensaje general mediante el parámetro idMensaje el cual se decodifica y se muestra en su vista correspondiente para consultar. |
| cargarMonedas(HttpServletRequest request, HttpServletResponse response) | Crea un objeto JSON que se forma mediante una lista de la clase MonedaSABIC de la cual se utiliza el dato sigMoneda(). |
| imprimirAutomatico(HttpServletRequest request, HttpServletResponse response) | Genera un arreglo de bytes que conformarán el PDF que se mostrará en un *applet*. |

## Anexo # 7. Casos de prueba

### Caso de prueba requisito registrar mensaje SLBTR MT199 con código de solicitud 21.

**Condiciones generales**:

- Se registra el mensaje.

**Condiciones de Ejecución**:

- El usuario debe estar autenticado con los permisos necesarios para realizar la acción.
- Debe existir conexión con la Base de Datos.

| Nombre de la sección | Escenario de la sección | Descripción de la funcionalidad |
|---|---|---|
| SC 1: Gestionar mensaje SLBTR MT199 con código solicitud 21. | EC 1.1: Registrar mensaje | El usuario llena los campos necesarios del mensaje que desea registrar, se verifican que los datos sean correctos y se persiste el mensaje. |
| | EC 1.2: Datos incorrectos | El sistema señala los datos que se insertaron con errores y muestra un mensaje de error en cada caso en correspondencia con el campo. |
| | EC 1.3: Cancelar operación | Se cancela la operación de registrar mensaje. |

### Secciones a probar en el caso de uso:

### Descripción de variables

| Nombre del campo | Clasificación | Valor nulo | Descripción |
|---|---|---|---|
| Referencia original | Campo de texto | No | Se introduce el dato |
| Fecha apertura | Campo de fecha | No | Se selecciona la fecha |
| Cuenta cheque | Campo de texto | No | Se introduce el dato |
| Cuenta CF | Campo de texto | No | Se introduce el dato |
| Código organ. CF | Campo de texto | No | Se introduce el dato |
| Código organ. Importador | Campo de texto | No | Se introduce el dato |
| Código one. CF | Campo de texto | No | Se introduce el dato |
| Código one. Importador | Campo de texto | No | Se introduce el dato |
| Moneda 1 | Lista desplegable | No | Se carga automáticamente |
| Importe 1 | Campo de texto | No | Se introduce el dato |
| Moneda 2 | Lista desplegable | No | Se carga automáticamente |
| Importe 2 | Campo de texto | No | Se introduce el dato |
| Tipo de cambio | Campo de texto | No | Se introduce el dato |

### Matriz de datos

| **Escenario: Registrar mensaje** | | | |
|---|---|---|---|
| **Resultado de la prueba** | **Satisfactoria** | | |
| **Respuesta del sistema** | El sistema registra el mensaje | | |
| | **Valor** | **T** | **Flujo central** |
| **Referencia original** | **CC204580** | V | 1. El usuario escribe la URL en el navegador.<br>2. Escribe las credenciales y selecciona el botón Aceptar.<br>3. Selecciona el subsistema SLBTR.<br>4. El usuario selecciona la opción Registrar Mensaje.<br>5. El sistema muestra la interfaz para seleccionar el tipo de mensaje y el usuario selecciona 21.<br>6. El usuario introduce los datos del mensaje y selecciona Aceptar.<br>7. El sistema valida los datos de entrada.<br>8. El sistema registra el mensaje y lo notifica al usuario. |
| **Fecha de apertura** | 12/05/2012 | V | |
| **Cuenta cheque** | 098787665 | V | |
| **Cuenta CF** | 897867564 | V | |
| **Código organismo CF** | 09087 | V | |
| **Código organismo importador** | 56418 | V | |
| **Código ONE CF** | 65214 | V | |
| **Código ONE Importador** | 65214 | V | |
| **Moneda 1** | CUC | V | |
| **Importe 1** | 594.36 | V | |
| **Moneda 2** | EUR | V | |
| **Importe 2** | 147.23 | V | |
| **Tipo de cambio** | 1.2301 | V | |

| Escenario: Datos incorrectos | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Resultado de la prueba** | **Satisfactoria** | | | | | | | | | | | | | | | | | | | | | | | | |
| **Respuesta del sistema** | El sistema alerta la existencia de datos incorrectos. | | | | | | | | | | | | | | | | | | | | | | | | |
| | Valor | T | Valor | T | Valor | T | Valor | T | Valor | T | Valor | T | Valor | T | Valor | T | Valor | T | Valor | T | Valor | T | Valor | T | Valor | T |
| **Referencia original** | **CC20 FF80** | **I** | CC20 4580 | V | CC20 4580 | V | CC20 4580 | V | CC20 4580 | V | CC20 4580 | V | CC20 4580 | V | CC20 4580 | V | CC20 4580 | V | CC20 4580 | V | CC20 4580 | V | CC20 4580 | V | CC20 4580 | V |
| **Fecha de apertura** | 12/05/ 2012 | V | 12/05/ 2012 | V | 12/05/ 2012 | V | 12/05/ 2012 | V | 12/05/ 2012 | V | 12/05/ 2012 | V | 12/05/ 2012 | V | 12/05/ 2012 | V | 12/05/ 2012 | V | 12/05/ 2012 | V | 12/05/ 2012 | V | 12/05/ 2012 | V | 12/05/ 2012 | V |
| **Cuenta cheque** | 09878 7665 | V | **Vacío** | **I** | 09878 7665 | V | 09878 7665 | V | 09878 7665 | V | 09878 7665 | V | 09878 7665 | V | 09878 7665 | V | 09878 7665 | V | 09878 7665 | V | 09878 7665 | V | 09878 7665 | V | 09878 7665 | V |
| **Cuenta CF** | 89786 7564 | V | 89786 7564 | V | **Vacío** | **I** | 89786 7564 | V | 89786 7564 | V | 89786 7564 | V | 89786 7564 | V | 89786 7564 | V | 89786 7564 | V | 89786 7564 | V | 89786 7564 | V | 89786 7564 | V | 89786 7564 | V |
| **Código organismo CF** | 09087 | V | 09087 | V | 09087 | V | **AA09** | **I** | 09087 | V | 09087 | V | 09087 | V | 09087 | V | 09087 | V | 09087 | V | 09087 | V | 09087 | V | 09087 | V |
| **Código organismo importador** | 56418 | V | 56418 | V | 56418 | V | 56418 | V | **ZZ56** | **I** | 56418 | V | 56418 | V | 56418 | V | 56418 | V | 56418 | V | 56418 | V | 56418 | V | 56418 | V |
| **Código ONE CF** | 65214 | V | 65214 | V | 65214 | V | 65214 | V | 65214 | V | **YY652** | **I** | 65214 | V | 65214 | V | 65214 | V | 65214 | V | 65214 | V | 65214 | V | 65214 | V |
| **Código ONE Importador** | 65214 | V | 65214 | V | 65214 | V | 65214 | V | 65214 | V | 65214 | V | **XX654** | **I** | 65214 | V | 65214 | V | 65214 | V | 65214 | V | 65214 | V | 65214 | V |
| **Moneda 1** | CUC | V | CUC | V | CUC | V | CUC | V | CUC | V | CUC | V | CUC | V | **12CU** | **I** | CUC | V | CUC | V | CUC | V | CUC | V | CUC | V |
| **Importe 1** | 594.36 | V | 594.36 | V | 594.36 | V | 594.36 | V | 594.36 | V | 594.36 | V | 594.36 | V | 594.36 | V | **Vacío** | **V** | 594.36 | V | 594.36 | V | 594.36 | V | 594.36 | V |
| **Moneda 2** | EUR | V | EUR | V | EUR | V | EUR | V | EUR | V | EUR | V | EUR | V | EUR | V | EUR | V | **12EU** | **I** | EUR | V | EUR | V | EUR | V |
| **Importe 2** | 147.23 | V | 147.23 | V | 147.23 | V | 147.23 | V | 147.23 | V | 147.23 | V | 147.23 | V | 147.23 | V | 147.23 | V | 147.23 | V | **W47.2** | **I** | 147.23 | V | 147.23 | V |
| **Tipo de cambio** | 1.2301 | V | 1.2301 | V | 1.2301 | V | 1.2301 | V | 1.2301 | V | 1.2301 | V | 1.2301 | V | 1.2301 | V | 1.2301 | V | 1.2301 | V | 1.2301 | V | **L1212** | **I** | | |

| | | |
|---|---|---|
| **Flujo central** | 1. El usuario escribe la URL en el navegador.<br>2. Escribe las credenciales y selecciona el botón Aceptar.<br>3. Selecciona el subsistema SLBTR.<br>4. El usuario selecciona la opción Registrar Mensaje.<br>5. El sistema muestra la interfaz para seleccionar el tipo de mensaje y el usuario selecciona 21.<br>6. El usuario introduce los datos del mensaje y selecciona Aceptar.<br>7. El sistema valida los datos de entrada y muestra el mensaje: “El valor especificado no es válido”.<br>8. El usuario corrige los datos de entrada y selecciona “Aceptar”.<br>9. El sistema valida los datos de entrada.<br>10. El sistema registra el mensaje y lo notifica al usuario. | *Las celdas de la tabla contienen V, I, o N/A. V indica válido, I indica inválido, y N/A que no es necesario proporcionar un valor del dato en este caso, ya que es irrelevante* |