**UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS**

**Facultad 3**

# Sistema integrador para el intercambio de mensajes SWIFT

Trabajo final presentado en opción al título de

Máster en Informática Aplicada

**Autor:** Ing. Adolfo Miguel Iglesias Chaviano

**Tutor:** Msc. Julio César Díaz Vera

**La Habana, Junio de 2012**

**AGRADECIMIENTOS**

Quisiera aprovechar este espacio del documento para exponer de una manera breve pero sincera el más profundo agradecimiento a aquellas personas que incidieron positivamente en el desarrollo de este trabajo. Agradecer a Yadira Calimano Meneses, Alain Lázaro Pinero, Emilio Ferrer Trujillo, Ray Williams Robinson, Yovani Royero, Analina Benítez Guzmán, Annier Mendoza, Jorge Luis Fonseca y Jorge Luis Martín. Agradecimiento también para mi tutor Julio César, pues aceptó la tutoría desinteresadamente a pesar de sus responsabilidades. Estoy agradecido igualmente de la compañera Verónica, especialista del Banco Central de Cuba y jefa de sistemas encargados de intercambiar información financiera en Cuba.

**DEDICATORIA**

El resultado de este trabajo se debe a varias personas como lo expuse en la sección de agradecimiento. También en él influyeron dos personas muy importantes para mí y a los cuales les dedico este trabajo.

A mi madre, Aurea Chaviano Rodríguez y a mi padre Adolfo Miguel Iglesias López. Sin la preocupación y el amor de ellos no hubiese tenido fuerza y motivación para culminar este trabajo.

**DECLARACION JURADA DE AUTORIA**

Declaro por este medio que yo Adolfo Miguel Iglesias Chaviano, con carné de identidad 83032711468, soy el autor principal del trabajo final de maestría Sistema integrador para la comunicación financiera, desarrollada como parte de la Maestría en Informática Aplicada y que autorizo a la Universidad de las Ciencias Informáticas a hacer uso de la misma en su beneficio, así como los derechos patrimoniales con carácter exclusivo.

Y para que así conste, firmo la presente declaración jurada de autoría en Ciudad de La Habana a los días ____ del mes _____ del año 2011.

**RESUMEN**

El presente trabajo describe las características de una solución informática que permite el intercambio de mensajes SWIFT entre entidades bancarias. El mismo se llevó a cabo por detectar que el sistema SISCOM, utilizado en el Banco Nacional de Cuba como sistema mediador para el intercambio de mensajes SWIFT, no cumplía con todas las necesidades. Se realizó un estudio de otros sistemas con características similares y se llegó a la conclusión que era necesario construir una nueva solución. En la nueva implementación, se utilizó el Proceso Unificado de Rational (RUP, en sus siglas en inglés) como metodología de desarrollo de software, Java como lenguaje de programación y Eclipse como herramienta de desarrollo. Los principales marcos de trabajos utilizados fueron: Spring, Hibernate y Dojo Toolkit. Con el desarrollo de este sistema se garantizó el principio contable de registrar el hecho económico en el momento que ocurre, ya que se automatizó la creación y procesamiento de mensajes SWIFT. Se disminuyó el esfuerzo de los operadores bancarios y el riesgo de generar errores en la conformación de estos mensajes. Por otra parte, la aplicación es flexible a los cambios que sufre anualmente el formato de los mensajes SWIFT, característica que no se lograba con el sistema informático utilizado anteriormente en el Sistema Bancario Cubano. Gracias a la arquitectura del sistema, es un componente reutilizable por otros desarrolladores que necesiten construir aplicaciones con el mismo objetivo.

**INDICE DE CONTENIDO**

# INTRODUCCIÓN

El Sistema Bancario Cubano (SBC) está compuesto por el Banco Central de Cuba (BCC), 9 bancos comerciales, 16 instituciones financieras no bancarias, 13 oficinas en representación de bancos extranjeros en Cuba y 4 oficinas en representación de instituciones financieras no bancarias.

El Banco Nacional de Cuba (BNC) es unos de los bancos pertenecientes al SBC. Su misión fundamental lo convierte en un banco muy peculiar, pues es la institución que se dedica a la compra de los alimentos, el combustible, los insumos para la zafra, las medicinas y la educación. Es el que recibe los créditos gubernamentales y bancarios para canalizar fondos a la Caja Central y emite las garantías que respaldan las operaciones comerciales [1].

Los sistemas informáticos que respaldan la ejecución de sus procesos de negocio son:

- SABIC: Sistema Automatizado para Banca Internacional de Comercio, en su versión MS-DOS.
- SISCOM: Sistema de Comunicación para la mensajería SWIFT.
- (SLBTR): Sistema de Liquidación Bruta en Tiempo Real, en su versión MS-DOS.

El SABIC es el sistema encargado de la gestión de los procesos bancarios. A través de este sistema se registran todas las operaciones contables del banco, se gestionan las cuentas bancarias, se gestionan los créditos concedido y recibidos, así como las operaciones relacionadas con las cartas de créditos. Este sistema se utiliza sobre el sistema operativo MS-DOS.

El SISCOM y el SLBTR son utilizados para enviar y recibir información financiera de otros bancos, aunque los formatos de los mensajes manejados por estos son diferentes. El SLBTR es utilizado para intercambiar mensaje SLBTR entre los bancos radicados en Cuba siempre y cuando la información transmitida este relacionadas con cuentas ubicadas en el BCC. El SISCOM por su parte, es utilizado para intercambiar mensaje SWIFT entre entidades financieras que usan del estándar SWIFT la norma ISO 15022. El sistema utilizado está desarrollado sobre Fox Pro, procedimientos almacenados en el SQL Server 2005 y un componente de comunicación sobre el lenguaje C. Este sistema es utilizado sobre el sistema operativo Windows.

En conjunto con especialistas del BNC y el departamento de informática del BCC trabajan estudiantes y profesores de la Universidad de las Ciencias Informática (UCI) con el fin de sustituir el SABIC utilizado en el BNC por otro sistema con características diferentes.

A partir de la observación de los procesos de negocio del BNC, de los encuentros realizados con especialistas de la misma entidad y el análisis de los documentos oficiales del BNC y de los sistemas informáticos instalados, se detectaron ciertas deficiencias en el proceso de intercambio de mensaje SWIFT con otros bancos. Por su importancia se describen a continuación:

- Los operadores del banco necesitan un tiempo considerable para realizar operaciones bancarias que llevan consigo la generación y recepción de mensajes. En varias ocasiones necesitan horas extras para realizar las operaciones. Todo esto se debe a que el SABIC y el SISCOM no se integran.
- Se requieren varios días de trabajo para actualizar el sistema SISCOM debido a los cambios que ocurren anualmente en el estándar de mensaje utilizado en el sistema SISCOM. Junto con esto es necesario especialistas bien capacitados para llevar a cabo esta actualización.
- Los datos que se envían y reciben en los mensajes son muy sensibles. En ellos se registran cuentas bancarias, montos de dinero, entre otras informaciones; que no deben ser consultados sin autorización. Los ficheros donde se almacenan estos datos se encuentran en las estaciones de trabajo del BNC y es posible acceder a la información contenida en los mismos sin ningún tipo de restricción.
- Los mensajes que se envían a menudo contienen información contable, la cual no es registrada cuando estos son enviados. Al día siguiente es cuando se realiza el movimiento contable de la información contenida en cada uno de los mensajes enviados.

A raíz de esto y como parte del desarrollo del proyecto se analizaron además otros tres sistemas informáticos que permiten también intercambiar mensaje SWIFT bajo la norma ISO 15022. Dos de estos sistemas, el Alliance Access (AA) y el Alliance Integrator (AI), desarrollados por la institución que creó el estándar de mensaje SWIFT, son los sistemas idóneos para utilizarlos en este caso; pero resulta que están desarrollados y utilizan componentes de software norteamericanos, por lo que está prohibido adquirirlos y utilizarlos en Cuba. El tercer sistema estudiado fue el Gari Gold

TFM, comercializado por la empresa brasileña Tas Group. Este sistema puede ser adquirido por el SBC; pero solamente está diseñado para conectarse a la red SWIFT e intercambiar mensaje. No tiene dentro de sus funcionalidades los mecanismos necesarios para la creación automática de mensaje a partir de los datos de negocio.

A raíz de esta problemática y la imposibilidad de utilizar alguno de los sistemas informáticos existente se definió como **problema científico:** *¿Cómo garantizar que se registre el hecho económico en el momento en que ocurre, se actualicen los cambios que ocurren en la norma ISO 15022 del estándar SWIFT y se restrinja el acceso a los mensaje de dicha norma en el Banco Nacional de Cuba?*

**El objeto de estudio** del presente trabajado está enmarcado en el intercambio de *información financiera* y su campo de acción es *el proceso de intercambio de mensajes SWIFT en el Banco Nacional de Cuba.*

**Como objetivo general del trabajo está:** *Desarrollar una solución informática para el Banco Nacional de Cuba que sea flexible ante los cambios que ocurren en la norma ISO 15022 del estándar SWIFT, brinde los mecanismos necesarios para registrar el hecho contable en el momento en que ocurra y controle el acceso a los datos del mensaje SWIFT .*

**Como idea defender se define que:** *Si se desarrolla la nueva solución informática que brinde las funcionalidades necesarias para gestionar el formato y las reglas semánticas, que tenga mecanismos de integración, transformación y seguridad implementados entonces se garantizará que la operación contable se registre en el momento en que ocurre, se podrán actualizar la estructura de los mensaje SWIFT cada vez que exista un cambio en la norma ISO 15022 y se controlará el acceso a los datos del mensaje en el Banco Nacional de Cuba.*

**Marco conceptual**

Se entiende como **proceso de intercambio de mensajes SWIFT:** las actividades relacionadas con la creación, el envío, la recepción, el procesamiento y la configuración de mensajes SWIFT.

**Las tareas realizadas para desarrollar este trabajo fueron:**

1. Caracterización de la norma ISO 15022 del estándar SWIFT.
2. Realización de un estudio para decidir la reutilización de tecnologías que permiten la creación, recepción y validación de los mensajes SWIFT bajo la norma ISO 15022.
3. Realización de un estudio para decidir la reutilización de sistemas informáticos que permiten la transmisión y recepción de mensajes SWIFT bajo la norma ISO 15022.
4. Definición de los estilos y niveles de integración que deben utilizarse en la solución.
5. Análisis de varias tecnologías de integración disponible a partir de una valoración de sus características.
6. Definición de las funcionalidades que debe brindar el nuevo sistema integrador para intercambiar mensajes SWIFT.
7. Realización del diseño y la implementación del sistema atendiendo a las funcionalidades definidas en el análisis.
8. Validación de las funcionalidades del sistema con CALISOFT y con el Banco Nacional de Cuba.

**Métodos científico utilizados en la investigación**

En el presente trabajo se utilizó el método teórico y el método empírico. Dentro del método teórico se encuentran los métodos lógicos y los históricos. En este caso se utilizó de los métodos lógicos, el hipotético deductivo y el dialéctico. El hipotético deductivo se utilizó en el análisis de la norma ISO 15022 y las teorías correspondiente a la integración de aplicaciones empresariales. Este estudio permitió inferir entre otros elementos, las características que debía tener una sistema compatible con la norma ISO 15022 y los estilos y niveles de integración que podrían utilizarse para darle solución al problema planteado. El método dialectico se utilizó en el análisis de las características de las tecnologías disponibles tanto para procesar mensaje SWIFT como para la integración de aplicaciones.

Del método empírico se utilizó la observación. A través de la observación se pudo constatar principalmente las tareas que ejecutaban las operadoras del BNC cuando procesaban mensaje SWIFT por el SISCOM.

**Aporte práctico e importancia de la solución**

Con el desarrollo de esta herramienta informática el BNC tendrá la posibilidad de respetar el principio contable de registrar el hecho económico en el momento que ocurre y mejorará la operatividad del banco porque las operaciones relacionadas con mensajes SWIFT se registrarán solo una vez. Todo esto será posible debido a las facilidades de integración que tendrá el sistema. Se obtendrá además un componente de software compatible con la norma ISO 15022 del estándar SWIFT, se tendrán las funcionalidades necesarias para actualizar la aplicación cuando cambie la norma ISO 15022 y se brindará una mayor seguridad al contenido de los mensajes. Esta herramienta constituye un producto informático reutilizable por todas aquellas entidades que intercambian mensajes SWIFT bajo la norma ISO 15022 y es un paso más hacia la independencia tecnológica del país ya que se implementará con tecnologías actuales y libres.

**Distribución del trabajo**

Capítulo 1: Fundamentación teórica: Se describe la norma ISO 15022 del estándar SWIFT y se analizan las ventajas y desventajas de las tecnologías disponibles para el estándar mencionado. También se valoran las características de los sistemas desarrollados tanto a nivel internacional como a nivel nacional que permiten intercambiar mensaje SWIFT. Luego se contextualiza la teoría vigente sobre la integración entre sistemas de información y se examinan las tecnologías disponibles para enfrentar un proyecto de este tipo.

Capítulo 2: Propuesta de la solución: Este capítulo menciona la metodología de desarrollo y las tecnologías utilizadas en el desarrollo del sistema informático. Se describen los procesos de negocios y los requerimientos de software identificados, el modelo de diseño e implementación, así como los diferentes entornos de despliegue que puede tener el sistema.

Capítulo 3: Validación de la solución: Este capítulo se enfoca en demostrar la validez de la solución desarrollada. Para esto se mencionan la pruebas realizadas y se expone un caso de prueba en representación del total realizadas, se valida la solución aplicando patrones establecidos científicamente y se describe el impacto que ha tenido la herramienta, así como los resultados obtenidos en eventos.

# 1. ESTADO DEL ARTE

En el presente capítulo se realiza una revisión bibliográfica sobre el tema en cuestión. Esencialmente se describe el estándar SWIFT de la norma ISO 15022 por ser esta la utilizada en el SBC y se analizan las ventajas y desventajas de las tecnologías disponibles para soportar dicho estándar. También se valoran las características de los sistemas desarrollados tanto a nivel internacional como a nivel nacional para soportar la comunicación a través de mensajes SWIFT. Luego se contextualiza la teoría vigente sobre la integración entre sistemas de información y se analizan algunas tecnologías que pueden utilizarse para llevar a cabo la integración entre aplicaciones.

## 1.1 Intercambio de mensajes SWIFT

Desde hace varios años, el mundo empresarial intercambia información de todo tipo y con varias finalidades. Existen alrededor de más de 100 estándares en la actualidad.

En el sector financiero se utilizan varios estándares. Algunos de estos son: el eXtensible Business Reporting Language (XBRL, por sus siglas en inglés) [2], el Financial Information eXchange (FIX, por sus siglas en inglés) [3], la United Nations/Electronic Data Interchange For Administration, Commerce and Transport (UN/EDIFACT, por sus siglas en inglés) [4], *SWIFT* [5], entre otros. Debido a que el BNC utiliza el estándar SWIFT para intercambiar información con otros bancos, no cumple objetivo para este trabajo realizar un análisis del resto de los estándares.

**SWIFT**

SWIFT se creó en Bruselas en 1973 y su desarrollo en el ámbito del intercambio de información financiero ha sido vertiginoso.

SWIFT es una cooperativa propiedad de sus miembros, a través de la cual el sector financiero lleva a cabo sus operaciones comerciales de forma rápida, segura y fiable [5]. Esta institución es únicamente un transmisor de mensajes. No posee fondos. No gestiona cuentas en nombre de los clientes, ni tampoco almacena información financiera de forma permanente. Al actuar como transmisor, sirve de vehículo para los mensajes transmitidos entre dos instituciones financieras. Esta actividad implica el intercambio seguro de datos privados, al tiempo que se garantiza su confidencialidad e integridad [5].

**Servicio FIN de SWIFT**

Las principales funcionalidades relacionadas con el envío, recepción y transmisión de mensajes están separadas por SWIFT como servicios de mensajería SWIFT. Estos servicios son FIN, InterAct, FileAct y Browse.

El presente trabajo explica solamente el servicio FIN ya que es el utilizado en el BNC.

El servicio de mensajería FIN permite el intercambio de mensajes formateados con el estándar MT de SWIFT (compatibles con la norma ISO 15022). Este estándar abarca una amplia variedad de sectores comerciales, y es utilizado y aceptado ampliamente por toda la comunidad financiera. FIN funciona en modo store-and-forward (almacenamiento y reenvío) y ofrece amplias funcionalidades de valor añadido [6].

**Norma ISO utilizada en el Sistema Bancario Cubano**

En los años 80 se creó la primera norma ISO 7775 basada en los mensajes SWIFT. Luego se creó la norma ISO 15022 como una evolución de la anterior. La última norma ISO 20022, la cual se está adoptando internacionalmente, cambia radicalmente la estructura del mensaje, pues las primeras normas se basan en mensajes de texto plano y esta última en formato XML.

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

Los mensajes MT tienen una estructura sintáctica y opcionalmente reglas semánticas. La estructura sintáctica del mensaje define los componentes que debe tener y su orden. Las reglas son las condiciones que deben cumplir los datos contenidos en los mensajes. Estos mensajes están compuestos internamente por bloques de contenido. Los tres primeros bloques contienen información de referencia para el mensaje, el cuarto bloque representa el texto del mensaje y es el más importante dentro de este ya que contiene la información financiera. El quinto y último, la información que se utiliza para garantizar la seguridad de los datos del mensaje.

Los bloques internamente están compuestos por campos o también conocidos como Tag. Los campos ubicados en el bloque 4 pueden estar o no conformados por sub-campos, los cuales tienen también un significado en el mensaje.

SWIFT desarrolló una notación que le permitió representar un formato específico para cada mensaje. Dentro de la notación se definió un alfabeto que representa los caracteres que puede contener los componentes del mensaje. Los llamados componentes son: bloque, secuencia, campos, sub-campo, función y componente. La definición de un componente tiene además de los caracteres permitidos dentro del alfabeto SWIFT, un número máximo de caracteres, los cuales deben cumplir o no según se indique, si es obligatorio su presencia y el número de veces que se puede repetir en el mensaje.

Las reglas semánticas como se mencionó anteriormente consisten en condiciones que debe cumplir el contenido de los mensajes. Un mensaje puede tener asociados más de una regla semántica y una misma regla puede ser aplicada a varios mensajes. SWIFT creó también una especie de lenguaje para representar estas reglas.

**Modificaciones en la norma ISO 15022 del estándar SWIFT**

Anualmente la institución SWIFT publica las modificaciones realizadas al estándar. Estas modificaciones se centran principalmente en cambios en la estructura de los mensajes y en las reglas semánticas.

A continuación se muestra el total de cambios realizados a las estructuras y reglas de los mensajes en el año 2008 y 2009. Los cambios abarcaron desde la creación de nuevos mensajes y reglas semánticas hasta la modificación y eliminación de estos.

| Año | Total de reglas afectadas | Total de mensajes actualizados |
|---|---|---|
| **2008** | 30 | 73 |
| **2009** | 14 | 22 |

**Tabla 1. Modificaciones en la norma 15022 del estándar SWIFT.**

## 1.2 Tecnologías compatibles con el estándar SWIFT Norma ISO 15022

Existen varias bibliotecas de clases o componentes de software que son compatibles con la segunda norma ISO del estándar SWIFT, por lo que es importante para el resultado del trabajo valorar sus características y analizar su reutilización atendiendo a cinco características fundamentales:

1. Componente gratis y accesible.
2. Componente con un modelo de clases que represente el formato del mensaje de manera genérica, es decir que el modelo pudiera utilizarse para cualquier

mensaje MT de la norma. Esto permitiría gestionar el formato del mensaje cada vez que cambie su estructura.

3. Componente con un modelo de clases que represente el formato de la regla semántica de manera genérica, es decir que el modelo pudiera utilizarse para cualquier regla según la notación definida en la norma. Esto permitiría gestionar las reglas semánticas cada vez que cambien.
4. Componente que tenga la funcionalidad de validar sintácticamente mensajes SWIFT. Este mecanismo de validación no debe afectarse en caso que cambie la estructura de algunos de los mensajes.
5. Componente que tenga la funcionalidad de validar semánticamente mensajes SWIFT independiente de la regla semántica asociada a estos. Ocurre lo mismo que en la cuarta característica. El mecanismo de validación debe ser independiente de la regla para que no se afecte si cambia la norma.

Estas cinco características permitirían entre otras cosas, que el sistema que se desarrolle pueda brindar las funcionalidades necesarias para gestionar el formato y las reglas de los mensajes SWIFT. Estas funcionalidades son necesarias para soportar los cambios que ocurren en la norma ISO 15022.

Los componentes seleccionados para analizar si es factible su reutilización son: WIFE, SWIFTDevTools, MessageObjects, C24io y SDK SWIFT.

**Componente WIFE**

El componente WIFE distingue en su jerarquía de clases los bloques 1 y 2 de los bloques 3, 4 y 5. Ver figura 1. Los tres últimos bloques se representan con una lista de Tag, a través de la clase con el mismo nombre. Esta clase representa un campo y es capaz de guardar el nombre de este y su texto. Esta jerarquía de clases no representa los sub campos u otros componentes internos de los campos.

Al decir de los propios creadores de WIFE: para WIFE todos los campos son simplemente pares de nombre y valor colocados en un objeto Tag. Si necesita procesar y obtener los componentes internos dentro de un campo específico puede utilizar las clases específicas para mensajes SWIFT y campos [7].

Esto es sin duda una característica muy importante; pues permite procesar y obtener los componentes internos; pero si aparece un nuevo campo al estándar SWIFT, entonces dejaría de ser funcional este componente porque no puede procesar el nuevo campo.

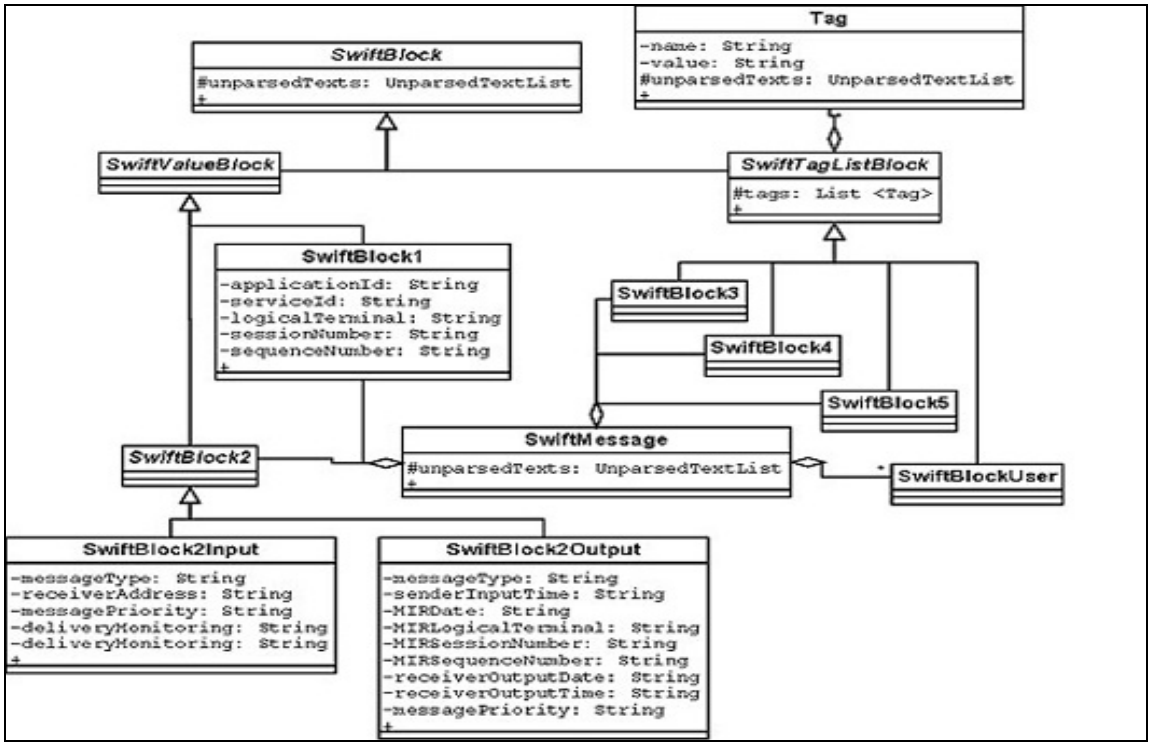



**Figura 1. Representación del mensaje SWIFT con la librería WIFE [7].**

```
String fin = "{1:F01BANKDEFMAXXX2039063581...";
IConversionService srv = new ConversionService();
SwiftMessage m = srv.getMessageFromFIN(fin);
Field32A f = new Field32A(m.getBlock4().getTagValue("32A"));
Calendar date = f.getComponent1AsCalendar();
BigDecimal amount = new BigDecimal(f.getComponent3AsNumber().doubleValue());
```

**Figura 2. Obtener elementos internos del campo con la librería WIFE.**

A partir del ejemplo mostrado en la figura 2, se puede mencionar que el componente WIFE no propone una estructura para representar los diferentes formatos de los mensajes MT.

El mecanismo utilizado por WIFE para leer un mensaje SWIFT y convertirlo a un objeto SwiftMessage no está basado en una configuración previa, en su algoritmo se encuentra la lógica necesaria para obtener el objeto mensaje. Al decir de WIFE, el componente parser[1] realiza las validaciones de sintaxis mínimas necesarias para obtener los datos del mensaje. La validación completa de mensajes (estructura, semántica, etc...) es una problemática independiente y no cubierta por el algoritmo de procesamiento del mensaje [8].

[1] Parser: Analizador

También este componente presenta una clase nombrada SwiftWriter y es utilizada para confeccionar un texto SWIFT a partir de un objeto de tipo SwiftMessage. Esta clase, SwiftWriter, no es capaz de validar la estructura del mensaje y la semántica del mismo.

**Componente SWIFTDevTools**

La compañía Datamation brinda la herramienta SWIFTDevTools, la cual da soporte a las aplicaciones relacionadas con los mensajes SWIFT. Esta herramienta según la página oficial la compañía [9], analiza, valida y construye los mensajes de la categoría 1, 2 y 9. Por otra parte la herramienta no es accesible libremente; pues se debe pagar alrededor de 6 499 EUR. Los componentes disponibles para soportar dicho estándar son: SwiftProcessor, SwiftValidator y Swift DBConn.

Se muestra en la figura 3 un ejemplo de código que valida un objeto SwiftMessage y obtiene el texto del mensaje.

```
public static void main(String[] args) throws Exception {
    // TODO Auto-generated method stub
    dat.SwiftCommon.SwiftMessage smObj = new dat.SwiftCommon.SwiftMessage();
    dat.SwiftCommon.Tag tempTag = new dat.SwiftCommon.Tag();
    java.util.Vector tempVec = new java.util.Vector();
    smObj.setArgApplid("F");
    smObj.setArgServid("01");
    smObj.setArgLTaddrBlk1("AAAAGRA0AXXX");
    smObj.setArgSesno("0057");
    smObj.setArgOsn("000289");
    // Se omiten los campos del bloque 2 y 3. Se muestran solamente un campo del bloque 4, g
    tempVec = new java.util.Vector();
    tempVec.addElement("5387354");
    // Se crea y agrega lo que seria el contenido del campo 20
    tempTag = new dat.SwiftCommon.Tag("20", tempVec);
    // Se introduce al bloque4 del mensaje.
    smObj.getBlock4().addElement(tempTag);

    // Validación y construcción del mensaje es forma de texto .
    dat.SwiftValidator.SwiftMsgValidator SMV = new dat.SwiftValidator.SwiftMsgValidator();
    dat.SwiftValidator.SwiftValidObj svObj = new dat.SwiftValidator.SwiftValidObj();

    svObj = SMV.ValidateMsg(smObj);
    if (!svObj.getErrorMessage().trim().equalsIgnoreCase("")) {
        String errorMessage = svObj.getErrorMessage();
        System.out.println(errorMessage);
        // se muestra el error
    } else {
        // No hay error
        String swiftMessage = svObj.getMessage();
        System.out.println(swiftMessage);
        // Se muestra el mensaje obtenido.
    }
```

**Figura 3. Validar y construir mensaje SWIFT con el componente SWIFTDevTools.**

El SwiftProcessor es un componente dedicado a interpretar el texto del mensaje SWIFT y obtener una representación objetual del mensaje a través de la clase SwiftMessage. También este componente tiene la capacidad de interpretar un objeto SwiftMesage y obtener el texto del mensaje.

El SwiftValidator realiza validación sintáctica y semántica del mensaje. Hasta la fecha no soporta la validación de los mensajes ubicados en las categorías 6 y 8. Este componente tiene la lógica de validación embebida en su código, algo que atenta contra la flexibilidad que debe tener para soportar los cambios que ocurren en el estándar SWIFT. Si cambia la sintaxis de algunos de los campos, sub-campos o componentes internos del mensaje entonces no podría utilizarse hasta que se modifique su implementación.

El SwiftDBConn es una clase que contiene las funcionalidades necesarias para persistir y recuperar de la base de datos los mensajes obtenidos de un texto.

**Componente MessageObjects**

La compañía AnaSys AG brinda el componente MessageObjects como tecnología compatible con SWIFT. MessageObjects está desarrollado en dos lenguajes: Java y .Net. Utilizan en su implementación plantillas XML con sus esquemas DTDs[2] y ficheros XSL[3] y soportan todas las categorías de mensajes de la norma ISO 15022. Según la documentación del sitio el componente es capaz de validar la sintaxis y la semántica de los mensajes [10]; pero no explica en detalles cómo lo realiza ni tampoco brinda la opción de descargar el código fuente para revisar su implementación. De hecho el producto es comercial; pero no indican su precio, se debe contactar con ellos a través de su sitio.

Es posible que esta biblioteca de clases sea factible para utilizarla; pero tiene el inconveniente que es comercial.

**Componente C24io**

El C24io Swift MT/MX Standards Libraries [11] es otro componente de software compatible con los mensajes SWIFT. Es una biblioteca comercial desarrollada en Java y compatible con la norma ISO 15022. Su documentación no es abundante; pero se conoce que trata al mensaje de forma objetual y en XML. En la versión demo descargada y limitada solamente a procesar y validar el mensaje MT 103, se puede constatar que brinda algunas facilidades para procesar los elementos internos de los

[2] Document Type Definition (DTD): El propósito de un fichero DTD es definir los bloques de un XML

[3] Extendible Stylesheet Language: es una familia de lenguajes basados en el estándar XML que permite describir como la información contenida en un documento XML cualquiera debe ser transformada o formateada para su presentación en un medio.

bloques del mensaje. Los mensajes, las reglas y algunos componentes como los campos, son representados por clases. Es decir, el mensaje MT 103 está representado por la clase MT103Element, algunos de sus campos son las clases Field20Reference, Field32aDateAndAmount, entre otras y algunas de sus reglas están representadas en las clases MT103C10_ACodeE16Rule y MT103C3_BCodeE02Rule.

```
    public static void runExample(String filename) throws IOException {
//        println("Reading SWIFT Message from '" + filename + "'...");
        SwiftPreParser preParser = new SwiftPreParser(new FileInputStream(filename));
        SwiftMessage msg = preParser.readSwiftMessage();
//        println("...done");

        System.out.println("Found SWIFT MT" + msg.getMessageType());
        System.out.println("Block 1: " + msg.getBlock1().toString());
        System.out.println("Block 2: " + msg.getBlock2().toString());
        System.out.println("Block 4: " + msg.getBlock4().toString());
        System.out.println("Incoming to SWIFT network? " + msg.isIncoming());
        System.out.println("Outgoing from SWIFT network? " + msg.isOutgoing());
        System.out.println("Service message? " + msg.isServiceMessage());
        System.out.println("System message? " + msg.isSystemMessage());
        System.out.println("Sequence number: " + msg.getSequenceNumber());
        System.out.println("Session number: " + msg.getSessionNumber());
        if (msg.getSourceLTAddress() != null)
            System.out.println("Source LT Address: " + msg.getSourceLTAddress());
        if (msg.getTargetLTAddress() != null)
            System.out.println("Target LT Address: " + msg.getTargetLTAddress());
        System.out.println("ACK? " + msg.isAck());
        System.out.println("NAK? " + msg.isNak());
    }
```

**Figura 4. Obtener y validar un mensaje con C24io Swift MT/MX Standards Libraries.**

Tal y como se muestra en la figura 4, la clase SwiftPreParser es la encargada de recibir el texto del mensaje y obtener el mensaje en forma de objeto a través de la clase SwiftMessage. Esta clase contiene entre otros, los métodos para acceder a los bloques del mensaje. Estos bloques a su vez tienen los métodos para interactuar con sus componentes internos.

La clase ValidationManager se encarga de validar el mensaje. Se le indica el mensaje y ejecuta la validación. Esta clase no se nutre de una representación del formato del mensaje para ejecutar las comprobaciones correspondientes.

### Standard Developer Kit de Swift (SDK)

El SDK es un componente de software desarrollado en Java que brinda la propia institución SWIFT para facilitar el desarrollo de aplicaciones compatible con sus mensajes. Tiene soporte tanto para mensajes MT como para los mensajes de la norma ISO 20022 (mensajes MX). Cada mensaje MT está representado por un esquema XML (XSD), y cada campo y sub-campo del cuarto bloque se definen como elementos independientes. Un ejemplo de esto se muestra en la siguiente figura:

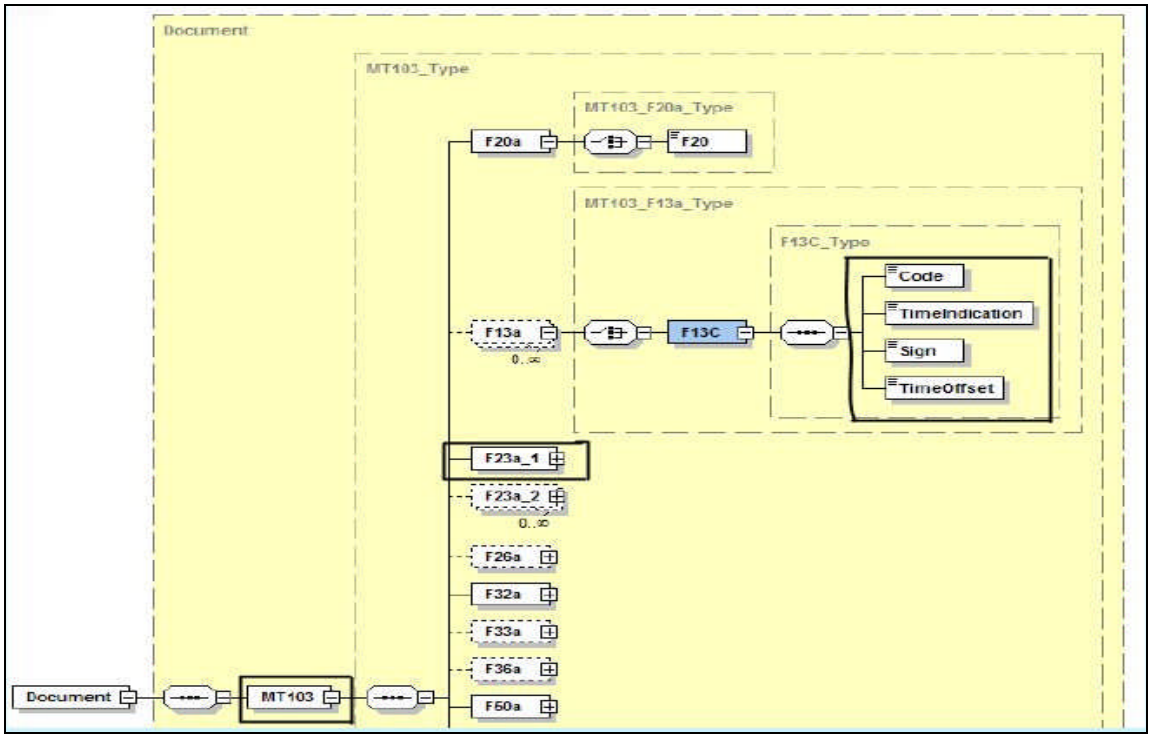



**Figura 5. Estructura utilizada por el SDK para representar un MT 103.**

Los rectángulos F20a, F13a, F23_1 y los otros que comienzan con F en la imagen anterior representan los campos del mensaje. El rectángulo F13C está compuesto por los sub-campos Code, Timeindication, Sign y TimeOffset.

```
private void convert(Reader pReader, Node pMessageNode) throws ErrorReportException {
    final PushbackReader pbReader = new PushbackReader(pReader, MT2XmlProcessor.MT_MAX_MSGSIZE);
    final LineNumberPushbackReader lineReader = new LineNumberPushbackReader(pbReader);
    final MTFieldReader fieldReader = new MTFieldReader(lineReader);
    final ReportErrorHandler errorHandler = new ReportErrorHandler();
    final MT2XmlProcessor processor = new MT2XmlProcessor(mSchemaDoc, fieldReader, pMessageNode, errorHandler);

    MatchResult matchResult = null;
    try {
        matchResult = processor.process();
    } catch (final MT2XmlException e) {
        errorHandler.add(e);
    }
    errorHandler.checkEnd(pMessageNode);
    MTInputProcessor.appendChild(pMessageNode, matchResult);
}
```

**Figura 6. Método para procesar un mensaje en texto y llevarlo a formato XML.**

A partir del estudio de un demo descargado de este componente se puede observar que la validación sintáctica del mensaje se garantiza a través de los XSD. Se obtiene, después de procesar un texto de mensaje, un objeto que representa el mensaje en formato XML. Para darle un tratamiento objetual es necesario tener clases por cada elemento definido en los XSD del mensaje. En la figura 6 se muestra el código para obtener la representación XML del mensaje.

De la validación semántica no muestran ejemplos y en la documentación disponible tampoco hacen mención a ella. Sin duda esta tecnología desarrollada por SWIFT soporta este tipo de validación; pero es imposible conocer cómo lo hacen a no ser que se adquiera el producto.

## 1.3 Aplicaciones informáticas para intercambiar mensajes SWIFT

Existen varias aplicaciones informáticas desarrolladas por varios proveedores que permiten intercambiar mensajes SWIFT. En la presente sección de este capítulo se analizan algunas de estas aplicaciones.

Para la selección de dichos sistemas se definieron algunos criterios:

- Sistemas desarrollados por los propios creadores del estándar SWIFT y por uno de sus socios comerciales, TAS Group.
- Sistemas desarrollados en Cuba y utilizado en el BNC.

**Sistemas informáticos desarrollados por SWIFT y por TAS Group**

La organización SWIFT ha sido puntera en el desarrollo de sistemas que permiten comunicarse con la red SWIFT. El AA y AI son algunos de sus principales productos.

El software AA es la interfaz de mensajería principal de SWIFT que permite conectar las aplicaciones a los servicios de mensajería de SWIFT mediante su amplia gama de adaptadores[4] disponibles [12]. Soporta los estándares de mensajería MT, MX, FpML y AnyXML.

A partir del AA se establece la comunicación con la red SWIFT. Dentro de sus funcionalidades según [12], se agrupan en Administración de mensajes, Validación, Encaminamiento avanzado, Auditoría y Supervisión de los mensajes.

Existen dos formas para interactuar con la aplicación. Una a través de una aplicación de escritorio y otra a través de una aplicación web. En dependencia de la versión que se utilice está disponible una o las dos interfaces gráficas.

El AA provee además varios adaptadores opcionales que facilitan la conexión con las aplicaciones de negocio. Estos son:

- Lectura y escritura de ficheros mediante el adaptador AFT (Adaptador de transmisión automática de archivos para el procesamiento por lotes).

[4] Componentes de software que facilitan la integración entre aplicaciones.

- Adaptador central de MQ incrustado (MQHA), que ofrece conectividad con WebSphere MQ independiente de la plataforma.
- Adaptador central de SOAP incrustado, que ofrece una alternativa a los vínculos interactivos legados.
- Un nivel genérico de servicios web, que permite que las aplicaciones comerciales puedan activar servicios específicos desde Access por medio de protocolos web estándar.
- MQSA, un adaptador legado, que ofrece conectividad con WebSphere MQ a través del Juego de herramientas de desarrollador de Alliance.

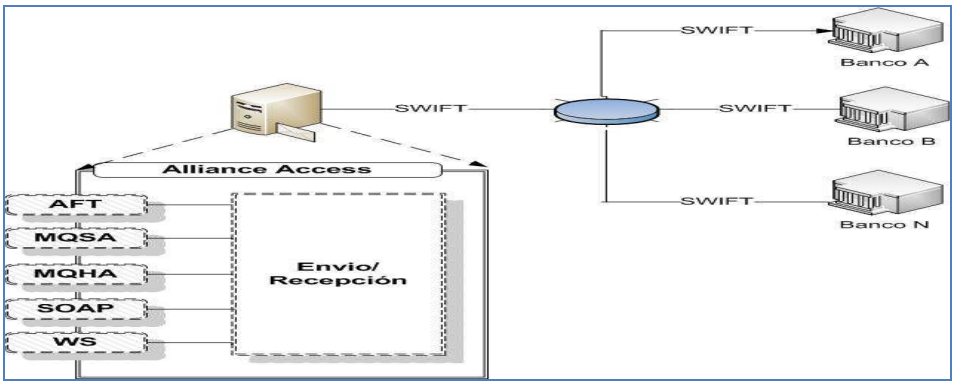



**Figura 7. Modelo de despliegue del Alliance Access.**

Al igual que las formas de interacción con el AA, la disponibilidad de los adaptadores está en dependencia de la versión que se utilice.

Este sistema es comercializable y el precio de adquirir este software es de 43 800 EUR y por mantenimiento anual de 14 400 EUR. Debido al bloqueo económico que Estados Unidos le impone a Cuba no se podrá seguir utilizando este sistema a partir del 2012 por tener componentes norteamericanos. Se está valorando la factibilidad de adquirir el sistema brasileño Gari Gold TFM.

La institución SWIFT también está asociada a diferentes proveedores de software y servicios financieros y los certifica como socios siempre y cuando cumplan con los requisitos que ellos establecen. En este caso está la compañía TAS Group, la cual comercializa el Gari Gold TFM, sistema mencionado anteriormente.

Este sistema cumple la misma función del AA. Se encarga de enviar y recibir los mensajes de la red SWIFT. Es compatible con la norma ISO 15022 y posee tres adaptadores para integrarse con aplicaciones empresariales que lo requieran. Estos adaptadores son:

1. Cola de mensajes

2. Escritura y lectura de ficheros
3. Servicios Web

La arquitectura del sistema se divide en 4 capas, las cuales corresponden a funcionalidades y la ubicación física [13]. En la imagen siguiente se puede observar estas capas.

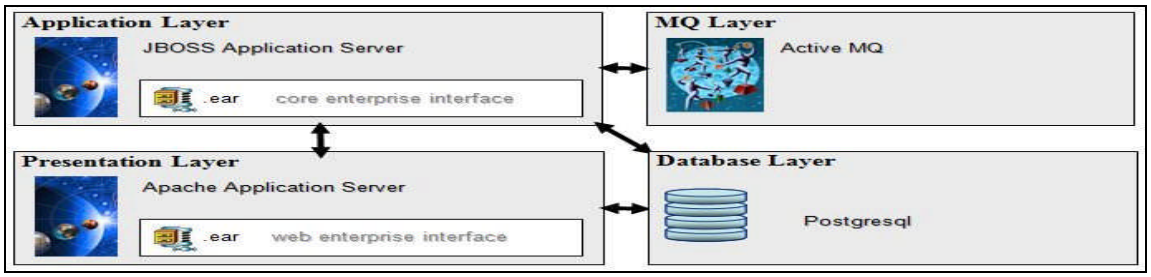



**Figura 8. Capas arquitectónicas del Gari Gold FTM.**

El AI es un complemento de AA que ayuda a conectar las aplicaciones empresariales a SWIFT con un trabajo mínimo [6].

Permite disminuir al mínimo las modificaciones necesarias en las aplicaciones de back-office, reducir el tiempo y los gastos relacionados con la implementación de las soluciones empresariales a través de SWIFTNet [6].

El AI soporta la mayoría de los formatos de registro o archivo de las aplicaciones empresariales, por ejemplo, valores separados por comas (CSV), libros de copias COBOL, mensajes XML o mensajes SWIFT. También soporta en su funcionamiento los estándares MT, MX [6].

Posee además los distintos adaptadores propuestos por AA para comunicarse con éste e incluye el paquete de integración comercial de Oracle para la conexión con aplicaciones empresariales. En la figura 9 se muestra el entorno de despliegue y sus adaptadores implementados.

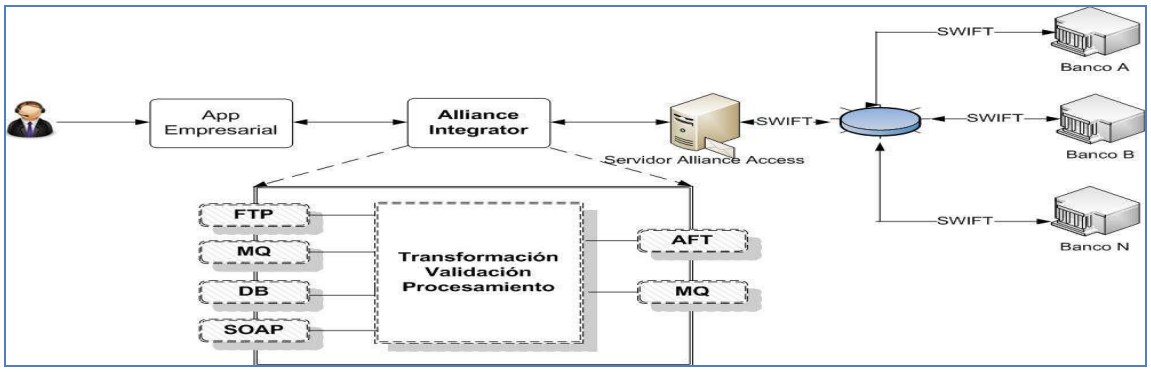



**Figura 9. Esquema general del Alliance Integrator.**

Otra característica importante del software es la interoperabilidad que brinda entre los mensajes MT y MX a partir de reglas definidas por SWIFT para la conversión automática y la compatibilidad con varias versiones de los estándares de mensajes.
Este sistema también es comerciable. El precio de adquisición es de 22 600 EUR y por el mantenimiento se debe derogar 10 200 EUR. La utilización de este sistema también está prohibida debido a las restricciones que el gobierno norteamericano le impone a Cuba.

**Sistemas desarrollados en Cuba**

En Cuba se han desarrollado hasta el momento dos aplicaciones diferentes para intercambiar información entre entidades bancarias. El primero fue el SISCOM y el segundo fue el SLBTR. Se analiza a continuación el SISCOM porque es la aplicación utilizada para enviar mensajes SWIFT.
El SISCOM es un producto para el procesamiento y enrutamiento de mensajería SWIFT a través de la red SWIFT FIN. El sistema utiliza como interfaz los servicios de un servidor SWIFT con el AA instalado [14].
Los módulos del sistema son:

- SACIM: Sistema de Archivo, Captación e Impresión de Mensajes.
- Middleware: Intermediario entre SISCOM y Alliance Access (AA).
- SwAdmin: Funciones administrativas.
- SwMPServer Módulo que interactúa directamente con el AA.

El SACIM tiene la función de confeccionar, archivar e imprimir los distintos tipos de mensajes que se intercambian entre las instituciones financieras a través de la red SWIFT [14].
El Middleware lo conforman dos sub-aplicaciones, SwMPLink y SwMPClient, cuya función es distribuir en ambos sentidos la mensajería entre la estación de comunicaciones del SISCOM ubicada en cada banco y el servidor AA, ubicado este en el BCC [14]. Es el que garantiza el flujo de mensajería en ambos sentidos.
Por su parte, el SwAdmin garantiza una salva del tráfico diario de mensajes y la recuperación de los mismos, así como la generación de reportes por diferentes criterios [14].
El SwMPServer es el que interactúa directamente con el AA. El mismo se encarga de escribir dichos lotes al AA luego de recibir los lotes de mensajes vía SwMPLink. En el

proceso inverso, lee los lotes de mensajes y se los envía al SwMPLink. La integración con el AA la realiza a través de la escritura y lectura de ficheros.

Los módulos SACIM, SwMPClient y SwAdmin y SwMPServer están desarrollados sobre el lenguaje FoxPro y el módulo SwMPLink en C. La información procesada por la aplicación es persistida en ficheros FoxPro y en el gestor de base de datos SQL Server 2005.

Los módulos del SISCOM no se comunican de manera directa. El SACIM una vez creado y firmado un mensaje lo coloca en una carpeta de salida. Por su parte, el módulo SwMPClient revisa cada cierto tiempo esa carpeta, una vez detectado algún mensaje, confirma los lotes de mensajes de salida y los coloca en otra carpeta. Aquí es donde comienza la función del SwMPLink, revisando cada cierto tiempo la presencia de lotes de mensajes por enviar. Una vez detectado los envía por sockets a través de una conexión que existe con el SwMPLink instalado en un servidor del BCC. Una vez llegado al BCC, son recibidos por el SwMPServer y los coloca en el AA. La comunicación en el proceso de recepción de un mensaje es muy parecida. Una vez recibido un lote de mensaje por el SwMPLink del BNC, realiza algunas verificaciones de integridad en los datos y lo escribe en una carpeta de entrada. Una vez detectada la llegada de un lote de mensajes por el SwMPClient, este obtiene los mensajes y los distribuye para las estaciones de trabajo según un fichero de distribución.

El despliegue del SISCOM, ver figura 10, utiliza el AA y los módulos SwMPServer y SwMPLink en un servidor ubicado en el BCC. Se instalan también los módulos SwMPLink, SwMPClient, SACIM, SwAdmin y una instancia del AA en el BNC. Los tres primeros módulos están en un servidor y los otros dos en otro servidor (variante utilizada en el BNC).

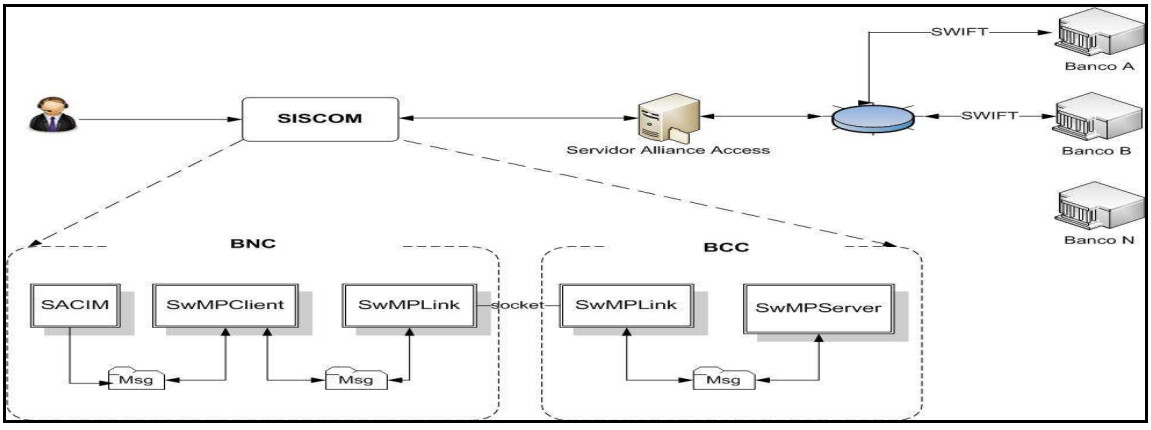



**Figura 10. Modelo de despliegue del SISCOM.**

Para que los usuarios puedan ejecutar el SACIM se debe crear en cada estación de trabajo un acceso directo al ejecutable del SACIM y copiarle los ficheros FoxPro necesarios para su funcionamiento.

De acuerdo a la problemática de la presente investigación, la integración entre sistemas de información constituye un punto importante para mejorar el proceso de comunicación a través del estándar SWIFT. Por tal razón en la sección siguiente se aborda este tema.

## 1.4 Integración entre aplicaciones empresariales

Existen diversas ideas en torno a qué es, en fin, la integración de sistemas de información [15] [16], ideas que, sin embargo, giran en torno a un consenso generalizado.

Algunas definiciones dadas por diferentes autores son:

… el compartimiento no restringido de información y procesos de negocio entre cualesquiera aplicaciones conectadas y fuentes de datos dentro de la empresa [15].

La integración es la tarea de hacer trabajar juntas a aplicaciones dispares para producir un conjunto unificado de funcionalidad [16].

Independientemente de que la naturaleza y objetivo de las definiciones anteriores es distinto, lo cual da lugar a que tengan algunas diferencias, dejan ver con claridad que la integración de sistemas involucra igualmente a información y procesos del negocio en función de proveer funcionalidad común.

**Niveles de Integración**

Los niveles de integración, definen o conceptualizan de alguna manera, las posibles dimensiones que este proceso pudiera abarcar. Todas las soluciones de integración pueden ser circunscritas, en uno o varios de los siguientes niveles de integración [15]: Nivel de Datos, Nivel de Interfaz de Aplicación, Nivel de Método y Nivel de Interfaz de Usuario.

La integración a nivel de datos comprende el proceso, y por consiguiente, las técnicas y tecnologías, de mover información entre almacenes de datos. En simples palabras se puede describir como extraer información de una base de datos, posiblemente realizar algún tipo de procesamiento sobre ella, y actualizarla en otra base de datos.

La integración a nivel de interfaz de aplicación se refiere al uso de las interfaces de las aplicaciones para llevar a cabo el proceso de integración. Estas interfaces no son más

que conjuntos de servicios expuestos por los desarrolladores dentro de las aplicaciones, usualmente en forma de API[5], con el objetivo de brindar acceso a información o procesos del negocio.

La integración a nivel de método se refiere al intercambio de lógica de negocio existente dentro de las aplicaciones. Aunque los mecanismos para su realización son varios, existen dos soluciones básicas: o bien se crea un conjunto compartido de servidores de aplicaciones que existen dentro de un mismo servidor físico compartido, o bien se usan tecnologías de intercambio de métodos como los objetos distribuidos.

La integración al nivel de interfaz de usuario es la más antigua de las formas de integración, pero aún continúa siendo necesaria en algunos ambientes. La idea es usar la interfaz de usuario de las aplicaciones como punto de integración.

**Estilos de integración**

Los estilos de integración describen las técnicas básicas que constituyen el fundamento de cualquier solución de integración. De acuerdo con lo anterior, cada estilo de integración es entonces un patrón, aunque todos, en primer lugar están orientados a resolver el mismo problema. En sentido general dichos estilos son [16]: Transferencia de Archivos, Bases de Datos Compartidas, Invocación de Procedimientos Remotos y Mensajería.

La transferencia de archivos, figura 11, consiste en colocar o leer archivos ubicados en una dirección de la máquina. Los archivos son un mecanismo de almacenamiento universal, integrado nativamente a cualquier sistema operativo y disponible en cualquier lenguaje de programación empresarial. Por tanto, constituye la manera más simple en la que, de alguna forma, dos o más sistemas pueden integrarse.

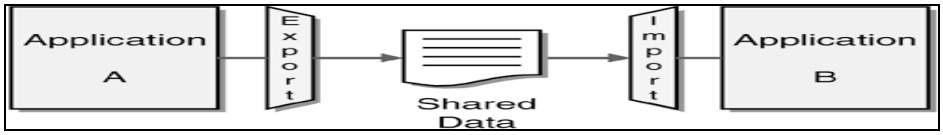



**Figura 11. Representación gráfica de la transferencia de archivos [16].**

Un punto importante aquí es el formato de los datos a utilizarse. Dado que rara vez, la salida que produce cada uno de los sistemas involucrados, es exactamente igual a la que espera el resto. Como resultado, los formatos estándares han evolucionado gradualmente, siendo hoy los basados en XML los más utilizados.

[5] Application Programming Interface

La Base de Datos compartida, figura 12, se refiere a que una base de datos es utilizada por varios sistemas a la vez. Con este estilo se elimina los problemas de consistencia, y desaparece la necesidad de crear un mecanismo de bloqueo, pues existen sistemas de administración de transacciones que evitarán múltiples operaciones contraproducentes entre sí.

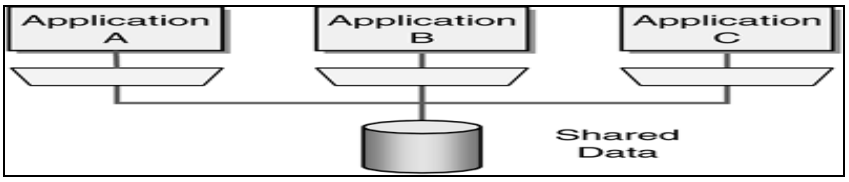



**Figura 12. Aplicaciones usando una base de datos compartida [16].**

Tener múltiples aplicaciones que intentan frecuentemente leer o modificar datos en ella, puede resultar un "cuello de botella" y por ende, largas esperas para realizar las operaciones a causa de los bloqueos. Una posible alternativa al respecto, sería una base de datos distribuida, aunque también presenta sus inconvenientes.

La Invocación de Procedimientos Remotos, consiste en que; si una aplicación necesita ejecutar alguna operación contenida dentro de otra, se lo solicita en la misma forma en la que se realizaría una llamada a una función local, aunque existen diferencias fundamentales. Este estilo, ver figura 13, provee una forma elegante de tratar las disonancias semánticas, puesto que como la información está encapsulada en métodos, se pudieran exponer varias interfaces para los mismos datos, aunque cada sistema tendría que negociar estas interfaces con sus vecinos.

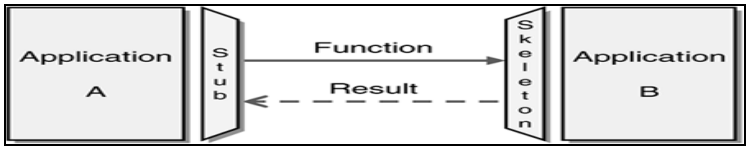



**Figura 13. Comunicación mediante invocación de procedimientos remotos [16].**

El estilo de Mensajería, figura 14, y en particular la mensajería asincrónica, es una reacción sumamente pragmática a los problemas de los sistemas distribuidos.

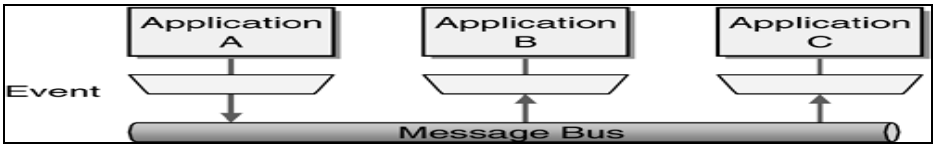



**Figura 14. Comunicación mediante mensajería [16].**

Aquí la principal observación es que el envío de un mensaje no requiere que los sistemas integrados trabajen al mismo tiempo. Los mensajes una vez enviados,

pueden ser transformados, incluso si es necesario, sin que las aplicaciones involucradas tengan conocimiento de esto en la práctica.

**Tecnologías para integrar sistemas de información**

La especificación Java Enterprise Edition (JEE[6]) ha demostrado ser de las más prometedoras y poderosas plataformas para la computación empresarial. Posee una gama de tecnologías que pueden utilizarse en los estilos de Invocación de Procedimientos Remotos y Mensajería, Base de datos compartidos y Transferencias de ficheros. A continuación se ofrecen de manera sintetizada, descripciones de aquellas que son más conocidas y difundidas internacionalmente.

Java Message Service (JMS): JMS es una API creada por Sun Microsystems, ahora de Oracle, para el uso de colas de mensajes, por lo que constituye el estándar sobre la plataforma JEE para crear, enviar, recibir y leer mensajes. JMS propone dos modelos para la comunicación. Está el modelo punto a punto y el Publicador/Suscriptor. En el modelo punto a punto el remitente envía los mensajes a una cola en particular y el receptor lee los mensajes de la cola. Por su parte el modelo Publicador/Suscriptor permite la publicación de mensajes para un tema de mensajes en particular y son los suscriptores los que deben registrar su interés en recibir mensajes de este tópico particular. Ambos modelos pueden ser sincrónicos o no; pero usualmente la variante asincrónica es preferida. Para usar JMS, se debe contar con algún proveedor que gestione tanto las sesiones como las colas. Existe actualmente una amplia variedad de estos proveedores, entre los que destacan Apache ActiveMQ, WebSphere MQ, y Oracle AQ, por solo citar algunos. En [17] se encuentra disponible una matriz de comparación histórica de los proveedores de JMS desde 2005.

Remote Method Invocation (RMI): RMI es un mecanismo que brinda Java como plataforma para invocar un método de manera remota. Dicho de otra forma, es la propuesta de dicho entorno para la invocación de procesos remotos anteriormente discutida. RMI es parte del entorno estándar de ejecución de Java y proporciona un mecanismo simple para la comunicación de servidores en aplicaciones distribuidas basadas exclusivamente en el mencionado lenguaje. El principal rasgo distintivo de RMI es su facilidad de uso, dado que está específicamente diseñado para Java. A través de RMI, un programa Java puede exportar un objeto, con lo que dicho objeto

[6] JEE es la especificación del lenguaje Java para implementar componentes empresariales.

estará accesible a través de la red y el programa permanece a la espera de peticiones en un puerto TCP. A partir de ese momento, un cliente puede conectarse e invocar los métodos proporcionados por el objeto.

Spring Framework[7] (SF): es una de las bibliotecas de clases más populares para el desarrollo de aplicaciones empresariales de los últimos tiempos. Brinda soporte para las tecnologías JMS y RMI. Como es de suponer, Spring no modifica la filosofía de trabajo con dichas tecnologías, pero sí representa una mejora sustancial en la manera de hacerlo, permitiendo al desarrollador explotar con mayor facilidad y en todas sus extensiones, las potencialidades de las mencionadas tecnologías. Para una descripción más detallada al respecto, ver [18], Parte IV.

JAX – RPC/JAX – WS: es un API que permite la invocación remota mediante XML, orientado a servicios Web. Esta API hace sencillo el uso y el desarrollo de servicios web. Implementa una llamada a procedimiento remoto como una petición y una respuesta de mensaje SOAP[8]. Esta tecnología es una forma especializada de mensajería SOAP. Con el advenimiento de su versión 2.0 JAX – RPC fue renombrada a JAX – WS para reflejar el movimiento desde el estilo RPC hacia el de los servicios web.

Apache Axis 2: es un completo rediseño y reescritura del ampliamente usado motor de servicios web Apache Axis. No solamente brinda la posibilidad de agregar interfaces de servicios web a las aplicaciones, sino que puede funcionar igualmente como un servidor de aplicaciones. Entre sus principales características se encuentran la velocidad de ejecución, pues usa su propio modelo de objetos y su propia API para el tratamiento de XML. Otra característica importante es que no consume mucha memoria, tiene un modelo de objeto ligero y propio para procesar mensaje. Soporta tanto la creación de servicios web como las invocaciones a éstos de manera asincrónica. Brinda una sencilla y limpia abstracción para integrar y usar transportes, al tiempo que el núcleo es independiente del transporte utilizado. Soporta además, las versiones 1.1 y 2.0 del WSDL e incluye la especificación de seguridad para los servicios web: WSS4J10 [19].

Spring WS: es un producto de la comunidad Spring centrado en la creación de servicios web manejados por documentos. Tiene como objetivo facilitar el desarrollo de

[7] Framework: Terminología utilizada para agrupar una jerarquía de clases que facilitan el desarrollo de aplicaciones informáticas. Constituye una tecnología.

[8] SOAP: Simple Object Access Protocol. Es un protocolo para compartir información estructurada en formato XML en un ambiente descentralizados y distribuido ,

servicios Contract-First[9] SOAP, permitiendo la creación de servicios web flexibles usando una de las tantas formas que existen para manipular contenidos XML. Está basado en Spring, lo cual implica que conceptos como Inyección de Dependencias pueden ser usados como parte integral de los servicios web implementados [20]. Spring WS tiene soporte para las dos ediciones desarrolladas del WS-Addressing, la cual consiste en cómo identificar los servicios web y mensajes de servicios web independientemente del protocolo de transporte utilizado. La WS-Addressing fue desarrollada por la W3C y pertenece al grupo de especificaciones de los servicios web. Los servicios web desarrollados con Spring WS son interoperable con AXIS1 y 2, Jax-WS, Windows Communication Foundation y Windows Services Enhancements (WSE) 3.0. Con relación al manejo de XML se puede realizar a través de varios API disponibles, así como tecnologías que permiten convertir de XML a Objeto y viceversa. En este caso están el JAXB 1 y 2, Castor, XMLBeans, entre otros. También soporta la especificación de WS –Security, la cual establece cómo brindarle seguridad a los servicios web. Esto lo logra utilizando su propio componente de seguridad, Spring Security. Soporta la versión 1.1 y 2.0 de SOAP. Propone además una jerarquía de clases para manejar las excepciones que se pueden lanzar ejecutando un servicio.

Spring Integration: es un nuevo miembro en el portafolio de Spring, es decir, es un nuevo sub – proyecto de Spring motivado por las mismas metas y principios que el resto de los existentes sub – proyectos. Extiende el modelo de programación de Spring hacia el dominio de la mensajería y se erige sobre el soporte que este brinda a la integración empresarial, para proveer un nivel de abstracción mayor.

Soporta enrutamiento y transformación de mensajes lo cual implica que diferentes transportes y formatos de datos pueden ser integrados sin impacto alguno en la funcionalidad de la solución. El diseño de Spring Integration está inspirado por el reconocimiento de una fuerte afinidad entre patrones comunes dentro del modelo de Spring y bien conocidos patrones de integración.

Apache Camel: es un framework de integración dirigido a hacer los proyectos de integración productivos, focalizado en simplificar la integración. El proyecto Camel comienza a principios de 2007 y pese a su relativa juventud, es ya un proyecto de código abierto maduro, disponible bajo la licencia Apache 2.

[9] Contract First: Es un método utilizado para definir un servicio web. Se define el WSDL y luego se implementa el contrato.

Camel puede ser visto como un motor de enrutamiento, o más precisamente un motor constructor de enrutamiento. Permite definir reglas propias de ruteo, decidir de cuáles fuentes aceptar mensajes y determinar cómo procesar y enviar esos mensajes hacia otros destinos. Uno de los principios fundamentales de Camel es que no asume nada al respecto de los tipos de datos que se necesitan procesar. Este es un punto importante, en tanto brinda al desarrollador la posibilidad de integrar cualquier tipo de sistema sin la necesidad de convertir los datos a un formato canónico, es decir, universalmente válido. Su arquitectura extensible y modular permite implementar y transparentemente conectar el soporte para protocolos propios, sean propietarios o no.
Otros proyectos de código abierto como Apache ServiceMix y Apache ActiveMQ, ya están usando Camel para llevar a cabo la integración empresarial.
De manera similar a Spring Integration, Camel es una implementación directa de los patrones de integración descritos en [16].

## 1.5 Conclusiones del capítulo

La siguiente tabla refleja cómo se cumplen las cinco características que deben cumplirse en los componentes analizados para que puedan ser reutilizados en este trabajo. Por las filas aparecen las características deseadas y por las columnas los componentes analizados.

| **Características** | **WIFE** | **SWIFTDev Tools** | **Message Objects** | **C24io** | **SDK SWIFT** |
|---|---|---|---|---|---|
| **Gratis y accesible libremente** | Si | No | No | No | No |
| **Modelo de clases que represente el formato de los mensajes de manera genérica.** | No | No | Se desconoce | No | No |
| **Modelo de clases que represente las reglas semánticas de manera genérica.** | No | No | Se desconoce | No | Se desconoce |
| **Validar sintácticamente el mensaje independiente del mensaje que esté validando** | No | No | Se desconoce | No | Si |
| **Validar semánticamente el mensaje independiente de la regla que esté validando** | No | No | Se desconoce | No | Si |

**Tabla 2. Características de los componentes de software compatibles con SWIFT.**

Luego de haber estudiado y analizado las características de los componentes seleccionados se puede concluir que ninguno cumple con todas las características deseadas. El SDK de SWIFT y WIFE parecen ser las mejores opciones; sin embargo el

SDK de SWIFT es un producto comercial y no cumple con la segunda característica indicada en la tabla, la cual es una de las más importantes. Por su parte el componente WIFE es gratis, pero no cumple con el resto de las características.

Del estudio realizado se desprende que: es necesario crear un componente propio, tomando las características positivas de los estudiados y que cumpla con todas las características identificadas.

Con relación a los sistemas informáticos, la institución SWIFT posee varios sistemas compatibles con la norma ISO utilizada en Cuba. Estos sistemas son idóneos para conectarse a la red SWIFT; pero al país le resulta prácticamente imposible adquirirlos por cuestiones de financiamiento y por el bloqueo económico impuesto por Estados Unidos. Por tal motivo, el SISCOM se utiliza en sustitución del AI. El lenguaje utilizado en el desarrollo del sistema SISCOM trae consigo que cada estación de trabajo su base de datos. Esto representa una vulnerabilidad del sistema; pues el usuario que tenga acceso a la estación de trabajo puede acceder a la información persistida.

Es justo señalar que la aplicación funciona correctamente y permite la transmisión de los mensajes SWIFT definidos. Con relación a los cambios que ocurren anualmente en la norma ISO 15022 se puede plantear que el sistema no posee las funcionalidades requeridas para soportarlos. Los cambios en las reglas semánticas no pueden ser modificados desde una interfaz gráfica de la aplicación; ya que no existen ficheros que guarden la expresión que representa cada regla, ni la asociación de éstas con los mensajes. Para modificar las reglas semánticas se necesita modificar el código fuente del programa. Con relación a las variaciones en las estructuras de los mensajes SWIFT, se puede plantear que tampoco son tolerados ya que la aplicación no cuenta con las funcionalidades necesarias. Los cambios se realizan mediante la interacción directa con la base de datos y en algunos casos es necesario modificar el código fuente.

La integración entre aplicaciones brinda la posibilidad de comunicarse entre ellos sin importar las herramientas utilizadas en su desarrollo, ni la forma en que gestiona su información. Para lograr una integración entre sistemas se crearon los llamados adaptadores, componentes de software que permiten el intercambio de información utilizando protocolos establecidos. Cualquier sistema que al menos soporte un tipo de adaptador entonces cuenta con un punto de integración.

El SISCOM como aplicación intermedia entre AA y la aplicación empresarial brinda solamente un punto de integración, utilizando el estilo de base de datos compartida.

Los mensajes creados por SWIFT están organizados de acuerdo a su propósito y todos menos los informativos pueden indicar la ejecución de una operación financiera. El SISCOM no tiene en cuenta este principio, por lo que no soporta la asociación de una operación de la aplicación empresarial con uno o varios mensajes; pues solamente se distribuyen los mensajes a la estación de trabajo indicada; a partir de un fichero de la base de datos que contiene la información sobre cómo distribuirlo.

Unas de las debilidades del SISCOM es la poca integración que tiene con la aplicación empresarial. Funciona como mediador entre dos sistemas, el sistema que gestiona los procesos financieros o bancarios y el sistema que se conecta a la red SWIFT.

Para la integración entre sistemas existen niveles de integración y formas o estilos de integración. Cada uno de estos estilos puede utilizarse según las circunstancias. Unido a esto, existen varias tecnologías que permiten resolver la problemática de la integración.

Es válido mencionar que este este trabajo es parte del desarrollo de un nuevo sistema de gestión bancaria, el cual propone una base arquitectónica (SF como componente fundamental en la arquitectura) que se debe respetar.

Una solución clásica de integración consiste en mediar solamente en la comunicación entre dos o más sistemas, preferiblemente más de dos sistemas. Sin embargo la problemática aunque incluye la integración de dos sistemas, no es exactamente una solución de integración descrita en el libro [15]; pues el propio sistema tiene su propia lógica de negocio de los mensajes y requiere además la interacción del usuario.

A partir de este contexto se descartan la utilización de Spring Integration y Apache Camel, tecnologías muy buenas para abordar un proyecto como el que se describe en el mencionado libro. Con respecto a las otras tecnologías se puede concluir que prácticamente todas poseen similares características, Spring WS posee al igual que el Apache Axis mecanismos para manejar los XML y las excepciones que pueden ocurrir, además de soportar las especificaciones estándar de WS-Security y WS-Addressing, entre otras características. En este caso Spring WS será utilizado ya que se basa en la arquitectura propuesta en SF, la cual debe ser utilizada. Dentro de la invocación remota se encuentra el propio API RMI y Spring RMI, esta última constituye una capa de abstracción de la primera y por consiguiente reducción del número de líneas de código a introducir en el programa. De la misma manera ocurren con el API JMS y Spring JMS.

# 2. PROPUESTA DE SOLUCIÓN

En el siguiente capítulo se describe la solución obtenida. La descripción se basa en mencionar la metodología de desarrollo y tecnologías utilizadas en la construcción del sistema informático. Se describen los procesos de negocios contenidos dentro del proceso de intercambio de mensajes SWIFT, los requisitos de software identificados, el modelo de diseño, así como los diferentes entornos de despliegue del sistema.

## 2.1 Metodología de desarrollo de software y tecnologías

Como metodología de desarrollo se utilizó Rational Unified Process (RUP), teniendo en cuenta las etapas de desarrollo, los flujos de trabajos y algunos de los artefactos que propone. Los principales artefactos generados fueron: las descripciones de caso de uso (CU) y los diagramas de CU, de actividades, de estados, de clases del diseño, de componente. También se crearon los casos de prueba para realizar las pruebas de caja negra.

Para modelar los artefactos anteriormente mencionados se utilizó el Lenguaje Unificado de Modelado y la herramienta Visual Paradigm en su versión 6.1.

Para llevar a cabo la implementación del sistema se utilizó la herramienta de desarrollo Eclipse 3.5 con sus extensiones y como gestor de base de datos SQL Server 2005. El SubVersion 5.0 para controlar las versiones del código. Entre los componentes de software estuvieron principalmente el SF versión 3.0 como base arquitectónica, Hibernate 3.5 para la interacción con la base de datos, Spring WebFlow para componer los flujos complejos del sistema y Dojo Toolkit 1.3 como librería de Java Script. En la seguridad del sistema se utilizó Spring Security en su versión 2.0.4. Spring Web Services 2.0.2 para consumir y publicar los servicios web, el módulo Spring RMI 3.0 para las invocaciones remotas, Spring JMS 3.0 para enviar mensaje al proveedor de mensaje y Spring Batch 2.1.0 M4 para realizar procesamiento en lotes.

## 2.2 Requisitos del sistema

Los requisitos del sistema son las condiciones o capacidades que el sistema debe cumplir [21]. En el libro [22] se plantea que el flujo de requisitos guía el desarrollo de un sistema.

Por tal motivo en las secciones siguientes se obtienen los requisitos del sistema, dentro de los cuales están los funcionales y no funcionales. Es válido aclarar que se describen

los procesos de negocio y CU más importantes, los que están relacionados con los problemas descritos en la introducción del trabajo.

**Procesos de negocio**

Identificar y describir los procesos de negocios relacionados con el intercambio de mensajes SWIFT constituye una herramienta fundamental para identificar correctamente las funcionalidades que debe brindar el sistema. Los procesos de negocio identificados son:

1. Confeccionar mensaje SWIFT.
2. Gestionar maqueta de mensaje SWIFT.
3. Autorizar mensaje SWIFT.
4. Enviar mensaje SWIFT.
5. Recibir mensaje SWIFT.
6. Procesar mensajes SWIFT recibidos.
7. Supervisar mensajes SWIFT.

**Casos de uso del sistema**

Para lograr una mayor claridad en el modelo del sistema se agrupan los CU según su propósito. Los grupos son los CU relacionados con la conformación, autorización, recepción y supervisión de mensajes SWIFT. Existen dos grupos más, uno que agrupa los CU relacionados con la configuración del sistema y el otro agrupa los que son utilizados en la mayoría de los grupos anteriores.

Dentro de los CU relacionados con la conformación de mensajes SWIFT están:

1. CU-Registrar mensaje SWIFT asociado a una operación bancaria.
2. CU-Registrar mensaje SWIFT sin tener operación bancaria asociada.
3. CU- Registrar los datos del encabezado del mensaje.
4. CU-Introducir datos al mensaje.
5. CU-Seleccionar maqueta para conformar el mensaje.
6. CU-Gestionar maqueta de mensaje.

El actor involucrado en estos CUs es la aplicación empresarial integrada y un operador para el segundo y sexto CU.

Se muestra a continuación la figura 15, la que muestra los CUs relacionados con la conformación de mensaje SWIFT.

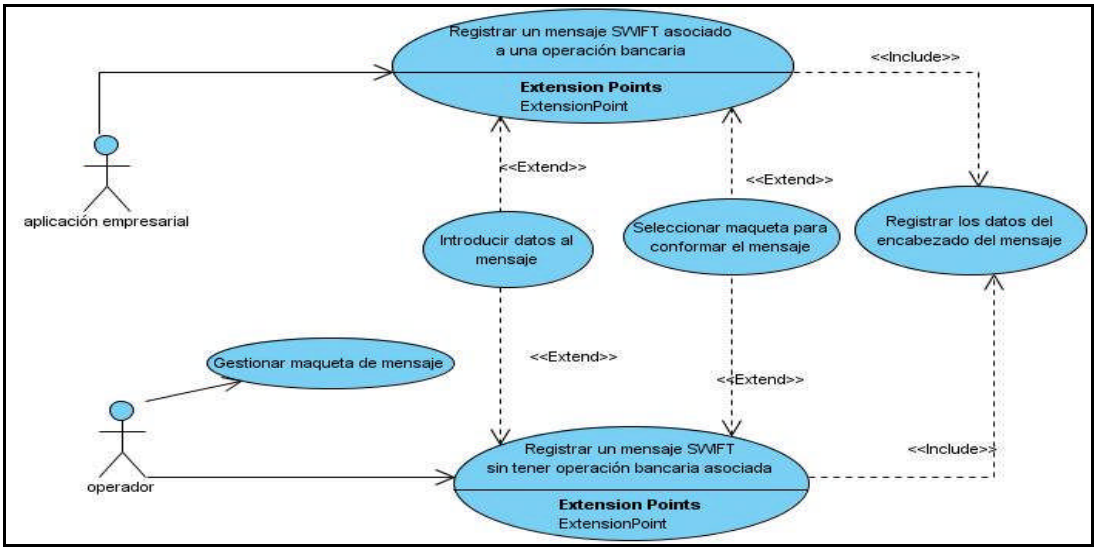



**Figura 15: Diagrama de CU relacionado con la conformación de mensaje.**

Dentro de los CUs relacionados con la configuración del sistema están:

1. CU-Gestionar formato del mensaje SWIFT.
2. CU-Gestionar formato del bloque SWIFT.
3. CU-Gestionar formato de secuencia SWIFT.
4. CU-Gestionar formato del campo SWIFT.
5. CU-Gestionar formato del sub-campo SWIFT.
6. CU-Gestionar formato de la función SWIFT.
7. CU-Gestionar formato de los grupos de componente SWIFT.
8. CU-Gestionar formato del componente SWIFT.
9. CU-Gestionar valores definidos por SWIFT.
10. CU-Gestionar notación SIWFT.
11. CU-Gestionar regla semántica.
12. CU-Gestionar permisos para firmar mensaje SWIFT.
13. CU-Gestionar tipos de firma.
14. CU-Gestionar permisos para procesar (conformar, actualizar, consultar y eliminar) mensaje SWIFT.
15. CU-Gestionar roles del sistema.
16. CU-Gestionar usuarios del sistema.

Los actores involucrados en estos CUs son el administrador de seguridad, administrador de configuración del formato y las reglas, administrador de los tipos de firma y los mensajes.

Se muestran en la figura 16 los CUs relacionados con la configuración del formato del mensaje SWIFT.

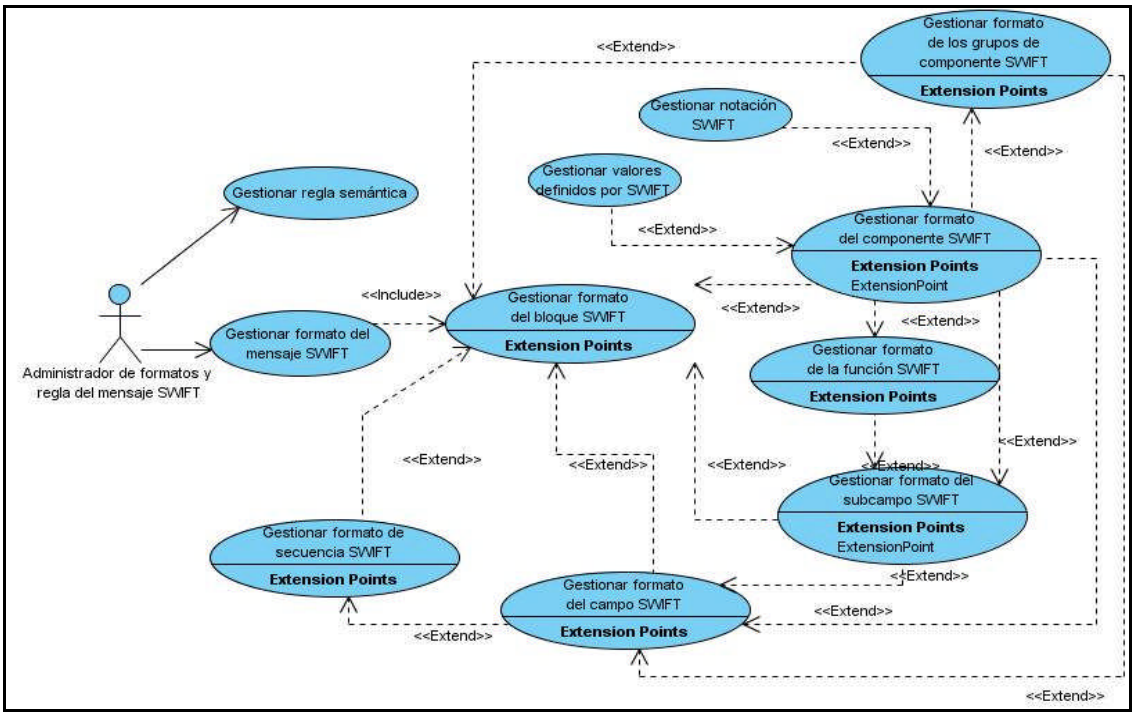



**Figura 16: Diagrama de CU relacionado con la configuración del formato del mensaje SWIFT.**

## Requisitos no funcionales

Los requerimientos no funcionales, como su nombre sugiere, son aquellos requerimientos que no se refieren directamente a las funciones específicas que proporciona el sistema, sino a las propiedades emergentes de este [22].

Para este sistema en cuestión se definió que el sistema debe ser seguro, usable, con buen rendimiento, que esté disponible en todo momento, que sea confiable y brinde facilidades de interoperabilidad con los sistemas informáticos a interactuar. Se describen a continuación las propiedades específicas que deben lograrse con cada uno de los requerimientos mencionados.

Seguridad

a) Cualquier persona que interactúe con el sistema debe ser un usuario activo registrado en la base de datos del sistema y previamente tuvo que autenticarse en el sistema.
b) Se mostrarán las funcionalidades que el usuario autenticado tenga permiso de ejecutar.

c) Se comprobará a través de un algoritmo de encriptación si se modificó el contenido del mensaje.
d) El intercambio de información de computadora cliente a servidor y viceversa se realizará a través de un canal seguro.
e) Cada vez que se genere un mensaje y cambie el estado del mismo se registrará el usuario que lo ha realizado.

Usabilidad

a) Cualquier usuario luego de recibir una capacitación del sistema podrá utilizar el mismo sin contratiempos.

Rendimiento

a) Los tiempos de respuesta en las actualizaciones no serán mayores a 5 segundos, para las transformaciones y recuperaciones menores a 10 segundos.

Disponibilidad

a) El sistema debe estar disponible en todo momento.
b) Un administrador del sistema debe encargarse de su ejecución constante.

Interoperable

a) El sistema debe brindar más de una forma de integrarse con la aplicación empresarial y con el sistema que se comunica con la red SWIFT.

Confiabilidad

a) Todos los mensajes que se envíen hacia la red SWIFT deben ser compatibles con el estándar SWIFT de la norma ISO 15022.
b) Se deben interpretar correctamente todos los mensajes que se reciben mediante la red SWIFT y cumplan con el estándar SWIFT de la norma ISO 15022.

## 2.3 Modelo de Diseño e Implementación

En el flujo de diseño se modela el sistema atendiendo a los requerimientos funcionales y no funcionales definidos anteriormente. El modelo de diseño incluye además la consolidación de la arquitectura del sistema.

En el flujo de implementación se comienza con el resultado del diseño y se implementa el sistema en términos de componentes, es decir, ficheros código fuente, scripts, ficheros de código binarios, ejecutables y similares [21].

En este trabajo existe una traza directa entre las clases diseñadas y los componentes creados para ejecutar la transformación. Por lo que el trabajo se limita solamente a

mostrar y explicar las clases del diseño que inciden directamente en los problemas mencionados en la introducción del trabajo.

**Modelo de datos**

Se definieron tablas para guardar información relacionada con el formato del mensaje y su regla semántica. En el Anexo 1 se muestra una imagen del diagrama entidad relacional con estas tablas. Se definieron además tablas, ver Anexo 2, para guardar el contenido de los mensajes que se conforman o reciben y los estados por donde transitan los mensajes, así como la relación que existe entre las operaciones de negocio que se ejecutan y los mensajes que son creados a partir de estos. También se crearon tablas para conservar los tipos de firma que pueden existir, así como las firmas que pueden realizar los usuarios del sistema.

**Vista lógica del sistema**

El sistema se organizó por módulos, los cuales agrupan funcionalidades que están fuertemente relacionadas entre sí. Los módulos se agruparon, de una forma lógica, en tres capas principales: la capa Núcleo del sistema, la capa Interacción con los usuarios, agrupa los módulos dedicados a la interacción con el usuario y la capa de Integración. Ver figura 17.

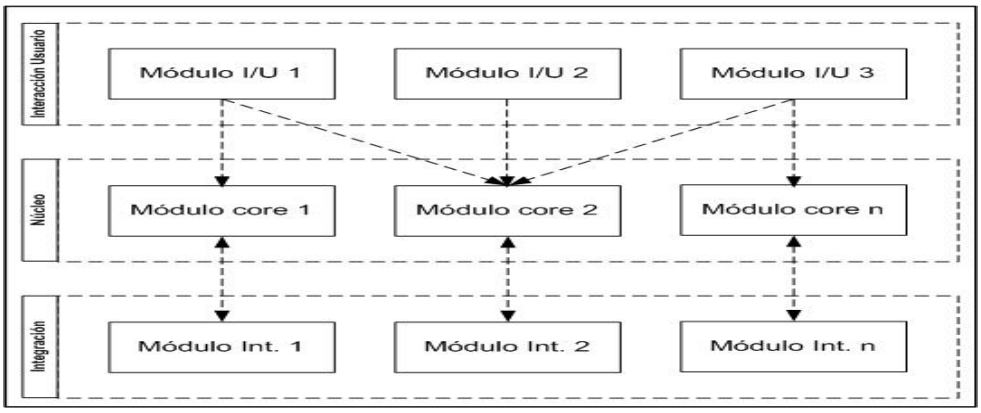



**Figura 17: Agrupación lógica de los módulos del sistema y sus relaciones.**

**Capa Núcleo del sistema**

El núcleo del sistema como su nombre lo indica, posee los módulos que contienen las funcionalidades básicas. De ellos se describirán los módulos Swift y Transformación por ser los más utilizados e importantes porque resuelven directamente algunos de los problemas mencionados en la introducción del trabajo.

Módulo SWIFT

El módulo Swift posee las clases que dan soporte a las funcionalidades relacionadas con el mensaje SWIFT. Contiene las clases que modela el mensaje Swift y los estados en los que puede transitar este. También posee las clases que representan el formato de los mensajes y sus reglas semánticas, además de las clases que permiten la validación sintáctica y semántica del mensaje. Por último, están las operaciones de persistencia y recuperación de los mensajes de la base de datos.

Las clases del formato y las reglas semánticas son utilizadas para representar la estructura sintáctica y semántica del mensaje respectivamente. Estos modelos permiten definir y cambiar en cualquier momento las estructuras y las reglas del mensaje sin afectar las funcionalidades de validación y procesamiento.

La jerarquía de clases y los atributos definidos para representar el formato del mensaje responden a dos cosas fundamentales:

1. Se pueda definir y modificar la estructura sintáctica del mensaje.
2. Los atributos brinden información necesaria para representar gráficamente el formato del mensaje.

Es válido aclarar que se le llama elemento SWIFT al bloque, secuencia, campo, grupo de componentes, función, sub-campo y componente. En la figura 18 se muestra las clases diseñadas para representar el formato del mensaje y sus relaciones. A continuación se explican en qué consisten cada una de ellas.

Las clases definidas en el formato son: SwiftFormatElement, SwiftFormatField, SwiftFormatBlock, SwiftFormatSubField, SwiftFormatSequence, SwiftFormatFunction SwiftFormatGroupComponent y SwiftFormatNodo, Todas estas clases heredan de la clase SwiftFormatElement, la cual contiene los atributos comunes. Estos atributos definen la estructura del mensaje. Algunos de ellos son: el identificador del formato, el nombre del elemento, si es opcional, la cantidad de veces que puede repetirse el elemento, además de una lista de SwiftFormatElement.

La clase de SwiftFormatNodo contiene un atributo del tipo Token, el cual representa el valor que puede tomar este elemento. Esta clase Token tiene como clases hijas TokenValue, TokenKeyValue y TokenSwift.

La clase TokenValue se utiliza para indicar un valor específico, la clase TokenKeyValue para indicar una lista de valores que puede tomar el componente. La clase TokenSwift se utiliza para indicar la notación SWIFT definida en la mencionada

norma ISO 15022 y contiene un atributo de tipo FieldLength, el cual es una clase que representa la cantidad de caracteres que debe tener el componente según la notación SWIFT seleccionada.

La clase MessageSwiftFormat contiene una lista de cinco SwiftFormatBlock y representa como tal el formato del mensaje.

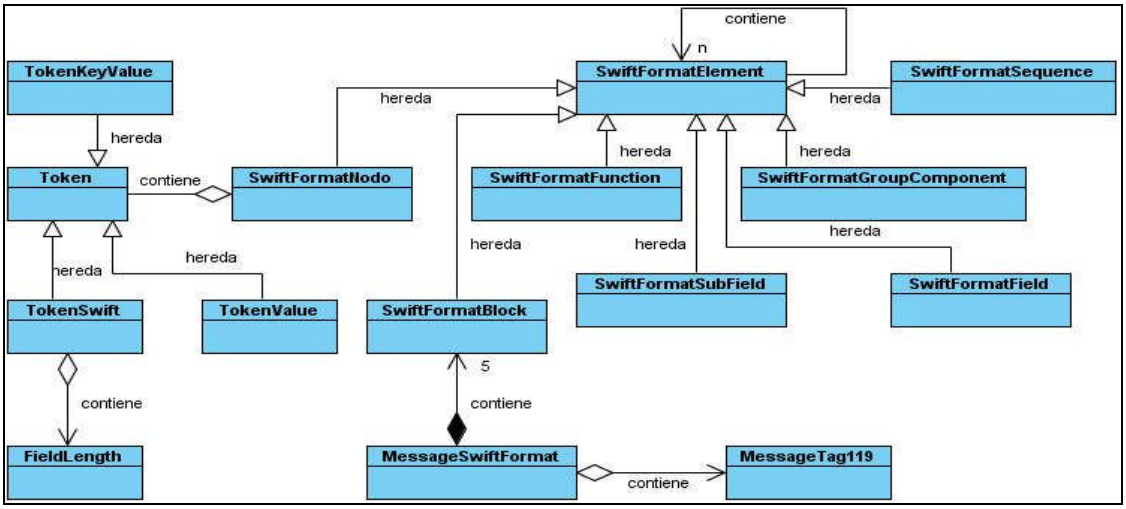



**Figura 18: Diagrama de clases para representar la estructura del mensaje.**

La regla semántica se define en un fichero XML a partir de su correspondiente XSD. El trabajo con la regla se realiza a través de clases. La estructura de este XML responde a la notación especificada en la norma ISO 15022 del estándar SWIFT.

Para el trabajo objetual de la regla, ver figura 19, se definieron la clases: la clase Rule, la cual tiene entre sus atributos un identificador de la regla, una descripción de la regla y una lista de Construct. De la clase Construct hereda IfConstruct y ForConstruct.

La clase IfConstruct define las acciones a tomar según las condiciones que son planteadas en la propia clase. Esta clase contiene atributos de tipo Statement, ThenConstruct, ThenProcedure, ElseConstruct y ElseProcedure.

La clase ForConstruct contiene otro Construct y un atributo de tipo Component, el cual se describirá posteriormente.

La clase Statement representa una condición que debe cumplirse. Existen varias formas de representar una condición, lo cual se traduce en seis clases que heredan de Statement. Estas son: OrderStatement CompoundStatement PresentStatement, ComparatorComponentStatement, ComparatorFunctionFunctionStatement y NotRepeatStatement. Cada una de estas clases responde a los tipos de condiciones definidas en el estándar.

Las clases que heredan de Statement utilizan por lo general la clase Component y las clases que heredan de Function para establecer la condición. La clases Component representa el valor a evaluar y las clases Function obtienen un valor ejecutando operaciones como multiplicación, suma o el subconjunto de un valor determinado.

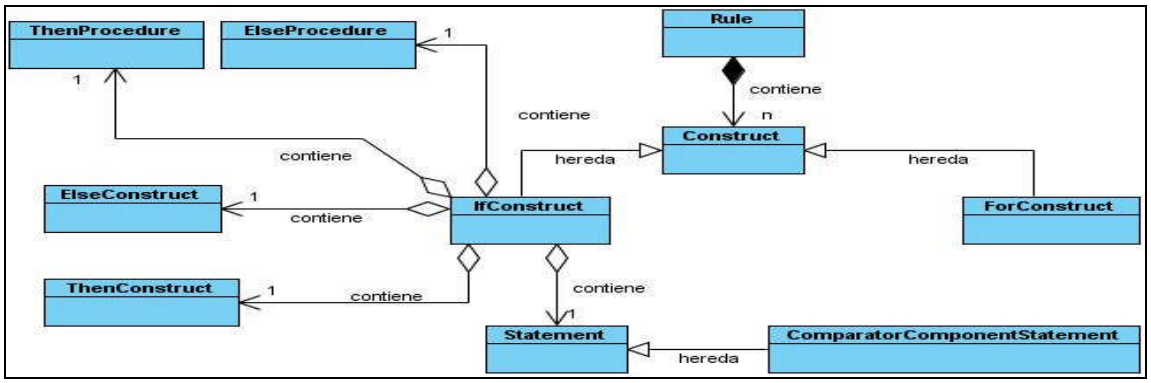



**Figura 19: Diagrama de clases para representar la regla SWIFT. Se muestran solamente las clases relacionadas directamente con la clase Rule.**

La clase ThenConstruct y ElseConstruct representa la ejecución de otro bloque de acciones y condiciones. ThenConstruct se utilizará cuando se cumpla cierta condición y ElseConstruct en caso contrario.

La clase ThenProcedure y ElseProcedure representa la acción terminal después de evaluar una regla. Ambas clases contienen la clase Procedure. ThenProcedure se utiliza cuando se cumpla cierta condición y ElseProcedure en caso contrario.

La clase Procedure representa una respuesta correcta de la evaluación de una regla. De esta clase, hereda la clase ErrorProcedure, representando una respuesta incorrecta.

Una vez creado un mensaje, este necesita tener una estructura de clases que guarde la información, así como los estados por los que transitan los mensajes.

La jerarquía de clases que representa un mensaje SWIFT está compuesta por 10 clases, las cuales son: SwiftElement, SwiftComponent, SwiftGroupComponent, SwiftFunction, SwiftField, SwiftSubField, SwiftBlock, SwiftSequence, SwiftMessage, SwiftMessageElement. Ver figura 20.

La clase SwiftMessageElement contiene entre los atributos más importantes: la referencia, el importe, el tipo de moneda utilizado en el mensaje, el banco corresponsal, el contenido del mensaje en forma de XML, un valor de código hash que se calcula para conocer cuando el mensaje ha sido modificado sin autorización. Incluye también una lista de los estados por donde ha transcurrido el mensaje. Los estados son representados por la clase SwiftFINMessageState. Otro atributo importante es la

clase SwiftMessage, la cual es una representación objetual del contenido del mensaje que se guarda en formato XML. La clase SwiftMessage se apoya en una lista de SwiftBlock, la cual representa el bloque del mensaje.

La clase SwiftFINMessageState representa el estado por el cual transcurren los mensajes. Para esto se auxilia de las clases MessageState, SwiftFINMesageSateOperacion y Usuario. La clase MessageState contiene todos los estados, SwiftFINMesageSateOperacion contiene la operación que provocó que el mensaje tenga ese estado y la clase Usuario representa como su nombre lo indica la persona que intervino en la creación o el cambio de estado del mensaje.

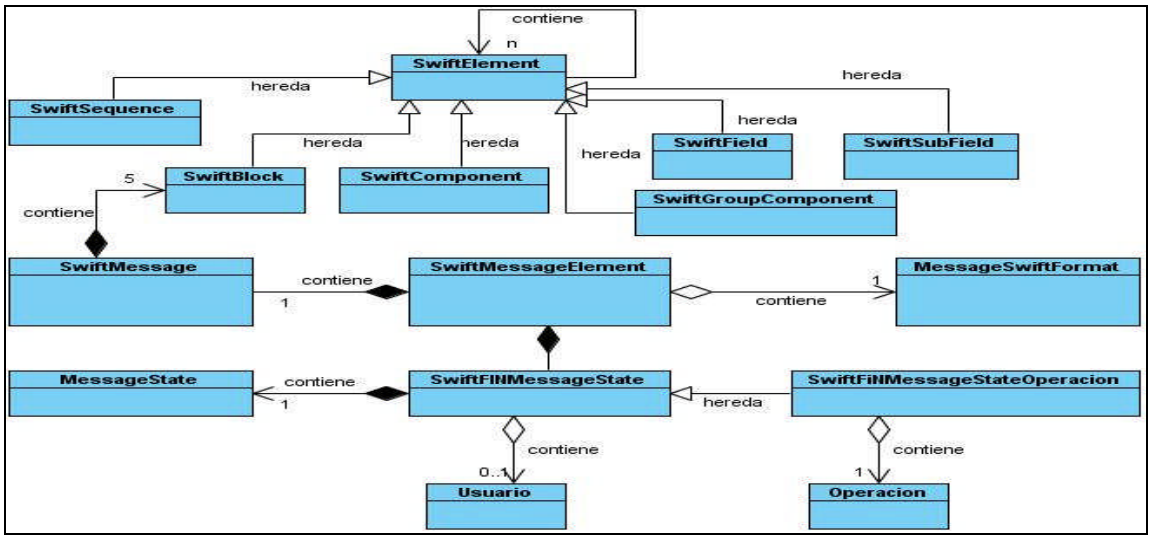



**Figura 20: Diagrama de clases que representa el mensaje.**

La validación sintáctica del mensaje y de su semántica se realiza independientemente del formato y su regla.

Existen dos formas de realizar la validación sintáctica del mensaje. Una consiste en la validación de un mensaje, ver figura 21 y la otra es la validación de un texto que representa el mensaje, ver figura 22.

Para la primera variante se debe utilizar el método parserSwiftMessage de la clase CopySwiftTextParser. Para la segunda variante se debe ejecutar el método parserText de la clase SwiftTxtParser.

La clase CopySwiftTxtParser se apoya para esta función en la clase SwiftAlphabet, SwiftMessageElementComplement y en las implementaciones de las interfaces SwiftAlphabetFactory e ISwiftElementParser.

A través del método loadSwiftAlphabet de la interfaz SwiftAlphabetFactory y sus implementaciones SwiftAlphabetFactoryDAO, SwiftAlphabetFactoryMemory y

SwiftAlphabetFactoryXML se carga el alfabeto de caracteres permitidos según la ISO 15022 y se guarda en la clase SwiftAlphabet.

En este modelo se utiliza el patrón de diseño Estrategia, el cual permite implementar la misma funcionalidad en varias formas.

A través del método parserSwiftElement de la interfaz ISwiftElementParser, junto con sus implementaciones SwiftElementAbstractParser, SwiftElementBlockParser, SwiftElementComponentParser, SwiftElementFieldParser, SwiftElementFunctionParser, SwiftElementGroupComponentParser, SwiftElementSequenceParser, SwiftElementSubFieldParser hacen posible la validación sintáctica.

La clase SwiftElementAbstractParser implementa la interfaz ISwiftElementParser; pero es una clase abstract. Contiene métodos de apoyos que son utilizados por las demás clases. Todas las clases que se describen a continuación heredan de la clase SwiftElementAbstractParser e implementan el método parserSwiftElement.

La clase SwiftElementBlockParser, SwiftElementSequenceParser, SwiftElementFieldParser, SwiftElementSubFieldParser, SwiftElementGroupComponentParser, SwiftElementFunctionParser, SwiftElementComponentParser se encarga de validar los bloques, secuencias, sub-campos, grupos de componentes, funciones y componentes del mensaje respectivamente. Se aplicó también el patrón de diseño Estrategia para implementar las validaciones de cada uno de los elementos del mensaje.

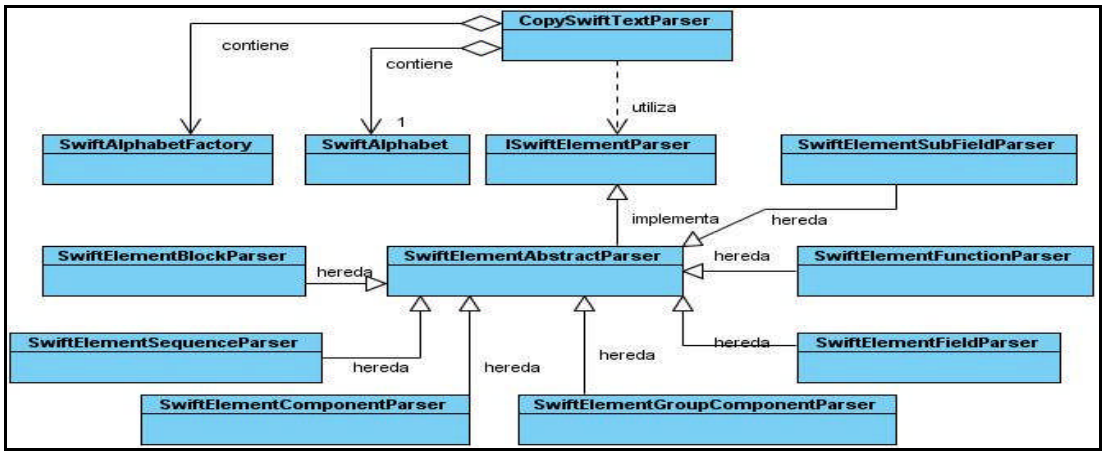



**Figura 21: Diagrama de clases para validar sintácticamente un mensaje representado en la clase SwiftMessageElement.**

La interfaz ISwiftTxtParser define el método se debe implementar para validar sintácticamente el texto de un mensaje. La clase SwiftTxtParser es la implementación

de esta y utiliza principalmente la interfaz ISwiftElementTxtParser y las clases MessageSwiftFormat, SwiftAlphabet, SwiftMessageElement, BlockTxtParser, FieldTxtParser, SubFieldTxtParser, GroupComponentTxtParser, SequenceTxtParser, ComponentTxtParser, FunctionTxtParser y SwiftElementTxtAbstractParser.

La interfaz ISwiftElementTxtParser define la forma de abordar la validación del texto a través del método parserText.

La clase SwiftElementTxtAbstractParser implementa esta interfaz y se utiliza para recorrer el texto y validarlo contra el formato del mensaje correspondiente. Las clases BlockTxtParser, FieldTxtParser, SubFieldTxtParser, GroupComponentTxtParser, SequenceTxtParser, ComponentTxtParser, FunctionTxtParser heredan de las clases SwiftElementTxtAbstractParser y validan los bloques, campos, sub-campos, grupos de componentes, funciones y componentes respectivamente.

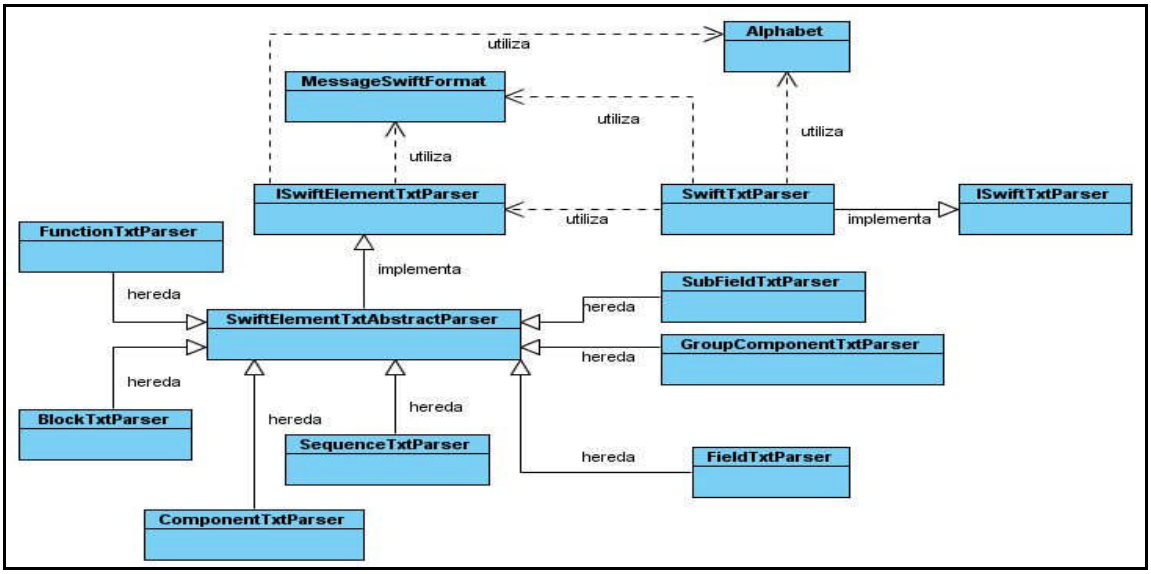



**Figura 22: Diagrama de clases para validar un texto de mensaje.**

La validación semántica se realiza, con una jerarquía de clases diferente.

Las clases principales para realizar la validación semántica son: ManagerSemanticAnalyzer, ManagerSemanticAnalyzerImpl y Analyzer. Esta última clase utiliza las implementaciones de las interfaces IConstructFactory, IfConstructEvaluator, IConstructEvaluator, IStatementFactory, IStatementEvaluator y IProcedureEvaluator.

Cada una de estas clases responde a los elementos definidos en la regla. Las interfaces y clases terminadas en Factory implementan el patrón de diseño Método de fábrica, el cual permite la creación de objetos.

Módulo Transformación

El módulo de Transformación posee las funcionalidades necesarias para llevar a cabo la transformación de un formato de datos a otro. En este caso estas transformaciones se realizan para crear mensajes a partir de datos de negocio y para extraer de los mensajes, los datos que serán utilizados por otra aplicación en la ejecución de sus procesos de negocio.

El sistema establece un modelo de clases para conocer qué transformación realizar cuando se ejecuta una operación de negocio determinada o cuando se reciben ciertos mensajes. Para esto se definió la clase Operacion y NumeroMensajeOperacion.

La clase Operacion contiene el identificador y el nombre de la operación de negocio y la clase NumeroMensajeOperacion contiene el mensaje asociado a la operación y el nombre del fichero que se debe utilizar para realizar la transformación y los usuarios que pueden realizar la operación contenida en la clase.

El modelo de clases para ejecutar la transformación, ver figura 23, está conformado por la interfaz ManagerTransform, ManagerTransformImplement, TransformationUtils, ObtainMsgSwiftImplement, NumeroMensajeOperacionDAO, OperacionDAO, MessageResultTransformation, XsltFileLoaderUtils, SwiftResultTransformation, ObjectXmlEntidad.

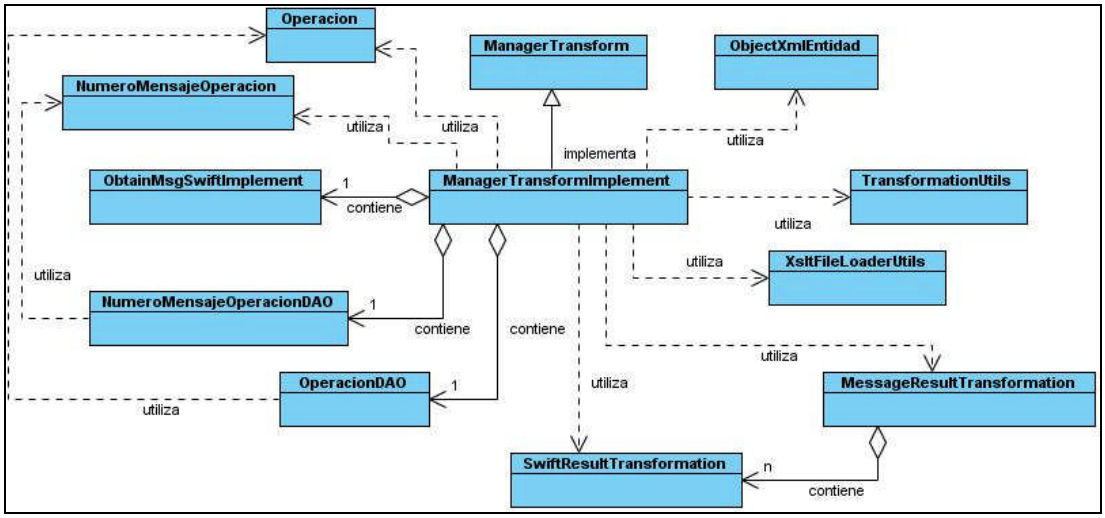



**Figura 23: Diagrama de clases para realizar la transformación.**

La interfaz ManagerTransform define los métodos necesarios para realizar las transformaciones. La clase ManagerTransformImplement implementa dicha interfaz y utiliza las clases restantes. La clase XsltFileLoaderUtils se encarga de obtener el fichero XSL para realizar transformación. La clase ObjectXmlEntidad contiene los datos que son utilizados para ejecutar la transformación. La clase TransformationUtils se

encargada de ejecutar la transformación. La clase ObtainMsgSwiftImplement se encarga de convertir el resultado de la transformación en el objeto SwiftMessageElement.

**Capa de Interacción con el usuario**

En la interacción con el usuario se encuentran los módulos:

- Administración: Funcionalidades relacionadas con la configuración del formato y reglas de los mensajes, los permisos para procesar y autorizar mensajes, así como la seguridad del sistema.
- Captación de mensajes: Funcionalidades relacionadas con la conformación de mensajes y la gestionar de las maquetas
- Autorización de mensajes: Funcionalidades relacionadas con la firma de mensajes conformados.
- Recepción de mensajes: Funcionalidades relacionadas con el procesamiento de mensajes recibidos.
- Supervisión: Funcionalidades que muestran los mensajes gestionados por el sistema en sus estados.

Módulo Administración

El módulo de Administración agrupa las funcionalidades identificadas como los CUs relacionados con la configuración del sistema.

Las funcionalidades de gestionar formato y reglas semánticas de los mensajes son las que brindan flexibilidad al sistema ante los cambios que ocurren en la norma ISO 15022 del estándar SWIFT. Para darle solución a la configuración del formato se utilizó Spring Webflow. En la capa de presentación del lado del servidor están las clases: CommonSwiftServiceFlow, FormatoSwiftServiceFlow, BloqueSwiftServiceFlow, SecuenciaServiceFlow, CampoSwiftServiceFlow, SubCampoServiceFlow, GrupoComponenteServiceFlow, FuncionServiceFlow, ComponenteServiceFlow, ExpresionSwiftServiceFlow, ValorDefinidoSwiftServiceFlow, FormatoSwiftFacade, FormatoSwiftFacadeImpl, SwiftFacade y SwiftFacadeTxt. En la figura 24 se muestra un diagrama de las clases mencionadas anteriormente.

Las clases cuyo nombren terminan en “ServiceFlow” son las encargadas de recibir las peticiones que realiza el usuario. Estas clases interactúan con la clase FormatoSwiftFacadeImpl para ejecutar sus funcionalidades. Por su parte

FormatoSwiftFacadeImpl interactúa con la clase SwiftFacadeTxt, la cual brinda las funcionalidades necesarias para crear, modificar y consultar y listar los elementos del formato del mensaje.

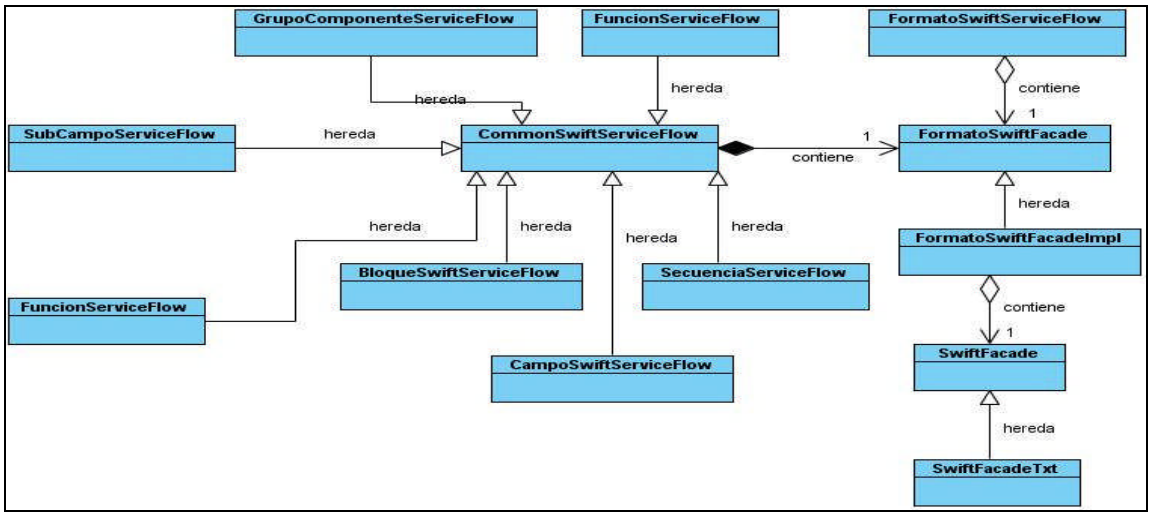



**Figura 24: Diagrama de clases para configurar formato de mensaje SWIFT.**

La clase FormatoSwiftServiceFlow controla las peticiones que listan, crean, actualizan y eliminan los formatos de mensajes SWIFT.

Esta clase se auxilia de la clase FormatoSwiftFacadeImpl para ejecutar sus métodos.

Las clases BloqueSwiftServiceFlow, SecuenciaServiceFlow, CampoServiceFlow, SubCampoServiceFlow, GrupoComponenteServiceFlow, FunctionServiceFlow y ComponenteServiceFlow gestionan la creación, actualización, eliminación y listado de los formatos de los bloques, secuencias, campos, sub-campos, grupo de componentes, funciones y componentes respectivamente. Estas clases utilizan para esto los métodos heredados y sus métodos propios.

Para gestionar las reglas semánticas se diseñaron e implementaron varias clases. En este caso se utiliza Spring MVC para recibir las peticiones clientes. Las clases diseñadas son: ReglaSemanticaMultiActionController, ServletFileUpload, ResourceLoader, ReglaSemanticaFacadeImpl, ValidateReglaXSD, ReglaSemanticaUtil, Rule, SwiftFormatBlock y MessageSwiftFormat. También MessageSwiftFormatDAO y RuleDAO, con sus implementaciones.

La clase ReglaSemanticaMultiActionController es la clase encargada de atender las peticiones que realizan los usuarios. Sus principales métodos son listar los mensajes registrados, obtener los datos de una regla, listar las reglas registradas, obtener el texto de la regla, obtener el texto del formato de un mensaje, leer el fichero que

contiene la definición de la regla, validar el contenido de la regla, eliminar la regla, entre otros.

Esta clase contiene un atributo de tipo ServletFileUpload para obtener el fichero de regla que el usuario introduce por la aplicación. Contiene también la clase ReglaSemanticaFacadeImpl para ejecutar la lógica de negocio. Esta se apoya en la implementación de MessageSwiftFormatDAO y RuleDAO para obtener el formato de un mensaje y las reglas que estén registradas en la base de datos respectivamente. También utiliza la clase ReglaSemanticaUtil y ValidateRegla XSD, las cuales verifican el contenido de la regla que se añada o se actualice. En la figura 25 se muestra el diagrama de clases relacionadas con la gestión de las reglas semánticas.

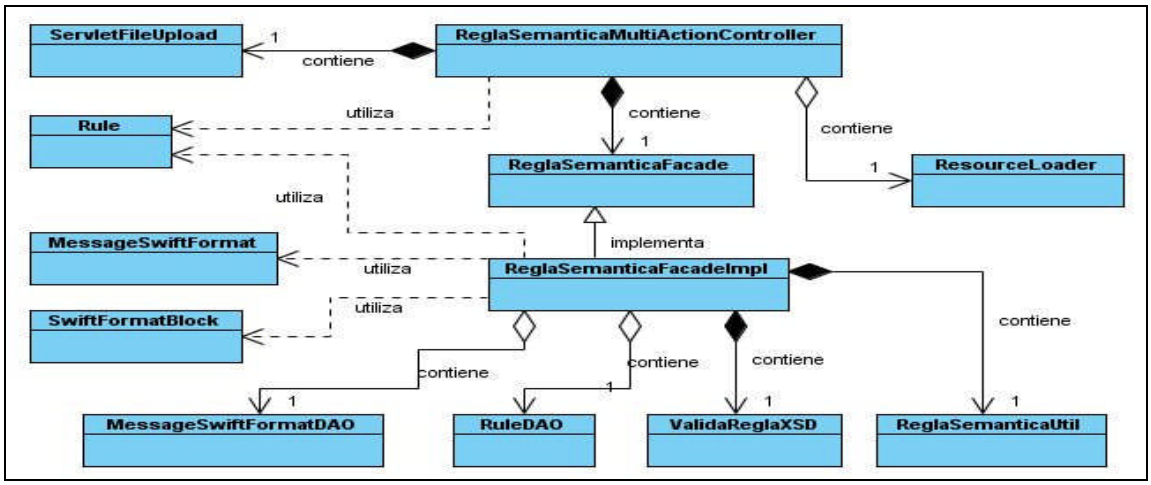



**Figura 25: Diagrama de clases para configurar regla semántica.**

### Capa de integración

El componente de integración contiene las funcionalidades necesarias para comunicarse con el software de conexión a la red SWIFT y la aplicación empresarial. Los adaptadores desarrollados son: Interfaces Java con sus implementaciones, servicios web mediante Spring WS, invocación a métodos remoto a través de Spring RMI, intercambio de mensajes a través Spring JMS, escritura y lectura de ficheros utilizando el paquete de Java IO y Spring Batch, recuperación e inserción de datos contra la base de datos SISCOM mediante Spring JDBC, Hibernate y el Spring Batch.

Módulo de Conformación

El módulo Conformación brinda los adaptadores necesarios para conformar mensajes SWIFT. Estos son: Interfaz y clase Java, Servicios web e Invocación remota por RMI.

En la integración con las interfaces y clases Java se definió una interfaz que tuviera las funcionalidades necesarias para conformar los mensajes. Su implementación se auxilia en varias clases. La interfaz se nombra IConformationSwiftMessageService y su implementación ConformationSwiftMessageService.

Los servicios definidos en la interfaz son:

- getMensajes: Se obtienen los mensajes que se pueden generar a partir de la ejecución de una operación determinada.
- listarMonitoreoMensaje: Se obtienen las diferentes formas de monitoreo de mensajes.
- listarPrioridadMensaje: Se obtienen las prioridades que puede tener un mensaje en su envío.
- *listarBancoDestinoPaginado:* Se obtienen los bancos disponibles en la red SWIFT a través del método**.**
- getSwiftFormatElementPorId: Se obtiene el formato de un elemento SWIFT a partir de un identificador pasado como argumento.
- getMonedas: Se obtienen las monedas disponibles.
- getBancoBySwiftBic : Se obtiene la entidad banco dado el código BIC.
- parseMiniSwiftElement: Valida el texto introducido para un campo o sub-campo del mensaje.
- construirMensaje: Conforma un mensaje SWIFT.
- procesarMensaje: Valida y guarda el mensaje conformado.
- Obtiene una lista de los XML que contienen los datos que se utilizaron en el negocio y que son útiles para conformar mensajes.

La clase ConformationSwiftMessageImpl se apoya en las implementaciones de las interfaces ManagerTransform, SwiftFacade, OperacionDAO, BancoDAO, MonitoreoMessageDAO, PriorityMessageDAO. En la figura 26 se muestran las clases mencionadas y sus relaciones.

Para conformar los Servicios Web se utiliza Spring WS. Primeramente se definieron los contratos y las clases que permitirán la ejecución de estos contratos. Las principales clases definidas para esto son: ConformacionEndPoint, WebServiceLocalPart, WebServiceServerException y ValidationParameterWebServiceClient. Se utiliza además la implementación de la interfaz IConformationSwiftMessageService y las clases que representan los contratos definidos, ver figura 27.

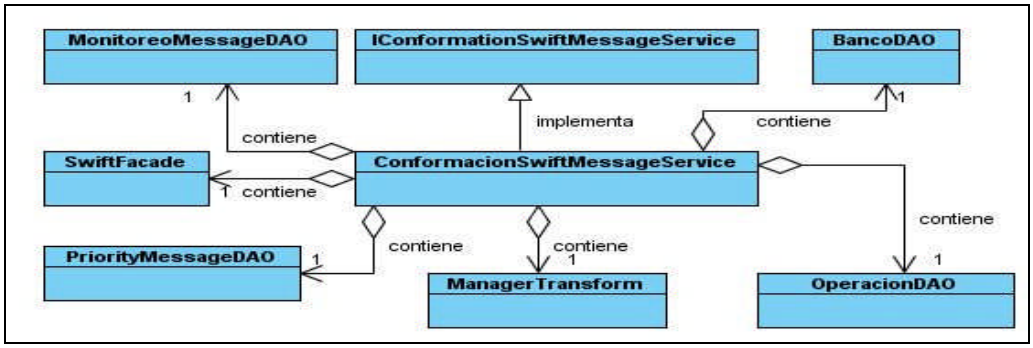



**Figura 26: Diagrama de clases para integrarse con la aplicación en la conformación de mensajes SWIFT.**

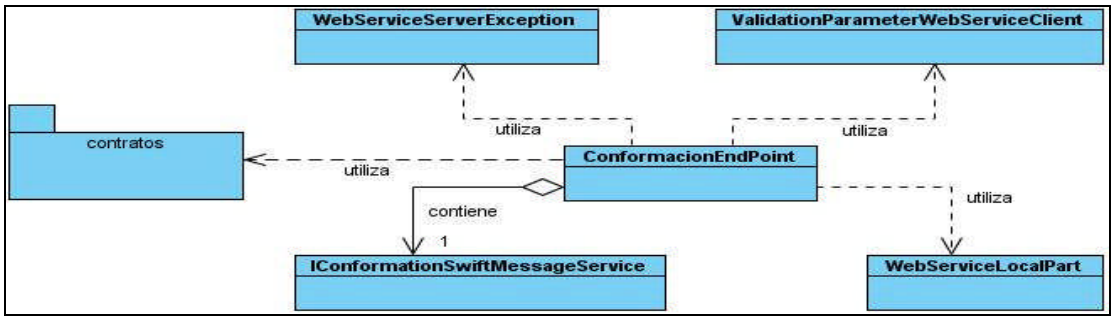



**Figura 27: Diagrama de clases del servicio web que permite conformar mensajes SWIFT.**

La invocación remota se realiza a través de Spring RMI. Se utiliza una clase de SF y la implementación de la interfaz IConformationSwiftMessageService. Los métodos expuestos son los mismos que están en esta interfaz. Ver figura 28.

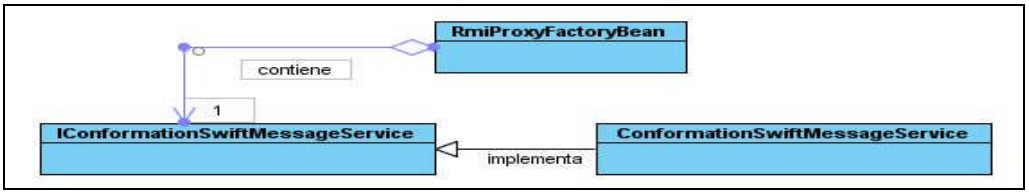



**Figura 28: Diagrama de clases que expone servicios a través de RMI.**

Módulo Envío

El módulo Envío contiene los adaptadores para enviar los mensajes SWIFT. Los adaptadores desarrollados son: Cola de mensajes, Escritura y lectura de ficheros y Base de datos.

Se utilizó Spring JMS para desarrollar el adaptador de cola de mensajes. La interfaz ISender define el método que debe implementarse para enviar mensaje. Junto con esta se utiliza otras interfaces y clases, las cuales son: IJmsSender, JmsGatewaySupport,

AbstractJmsSender, SwiftMessageElementJmsSender y ActiveMQConnectionFactory. En la figura 29 se muestra el diagrama de las clases mencionadas.

El envío de mensajes a través de base de datos se utiliza cuando el sistema SISCOM es el encargado de transportar los mensajes al servidor conectado a la red SWIFT. Las clases utilizadas para enviar mensaje a la base de datos son: ISender, IDataBaseSender, AbstractDataBaseSender, StoreProcedureAbstractDataBaseSenderMessage, SiscomSwiftStoreProcedureDataBaseSenderMessage. Ver figura 30.

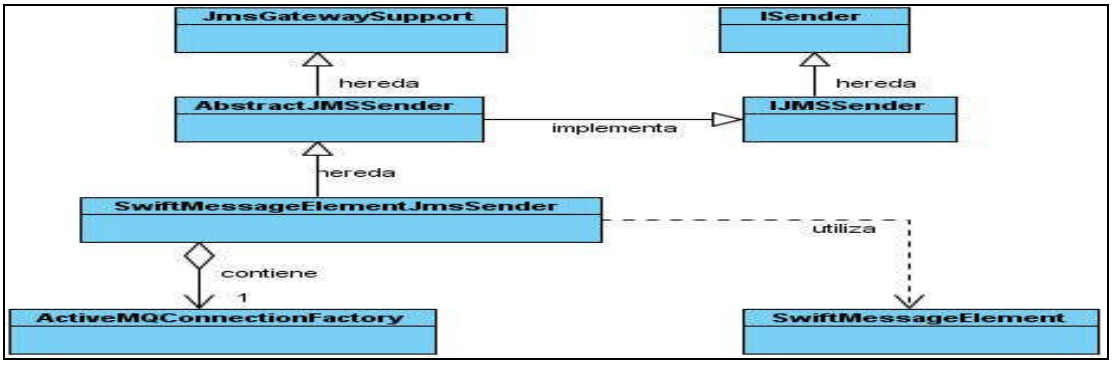



**Figura 29: Diagrama de clases enviar mensaje por cola de mensaje.**

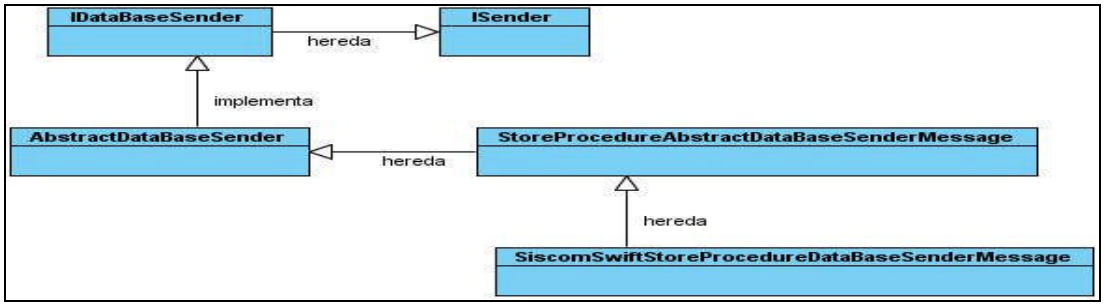



**Figura 30: Diagrama de clases para enviar mensaje por BD.**

Módulo Recepción

En el módulo Recepción se implementan los adaptadores que reciben los mensajes SWIFT. Los adaptadores desarrollados son: Cola de mensajes, Escritura y lectura de ficheros y Base de datos.

En la recepción de mensaje se obtienen mensajes enviados por otros bancos y la confirmación por parte de la red SWIFT de los mensajes enviados. Esta confirmación puede ser la aceptación o el rechazo del mensaje enviado al estar incorrecto su contenido.

Se describe solamente la escritura y lectura de ficheros ya que los otros adaptadores fueron explicados en el módulo de envío porque son utilizados también.

La recepción de mensajes a través de escritura y lectura presenta la misma particularidad que el envío. Cada cierto tiempo el sistema debe revisar si existen mensajes por recibir. Para esto también se utiliza Spring Batch y clases que permiten ejecutar tareas cada cierto tiempo. Para realizar la copia de ficheros se utilizan los paquetes del lenguaje Java disponible.

Las clases diseñadas para esto son: ReceiveConfirmationSwiftMessageByAFT y ReceiveNewSwiftMessageByAFT. También están las clases ReceiveSwiftMessageByAFTHelper y ReceiveSwiftMessageByAFTProcessor, así como la clase ItemWriteAdpater y la interfaz IReceiveSwiftMessageService con su implementación, ReceiveSwiftMessageService. Ver en la figura 31 estas clases y sus relaciones.

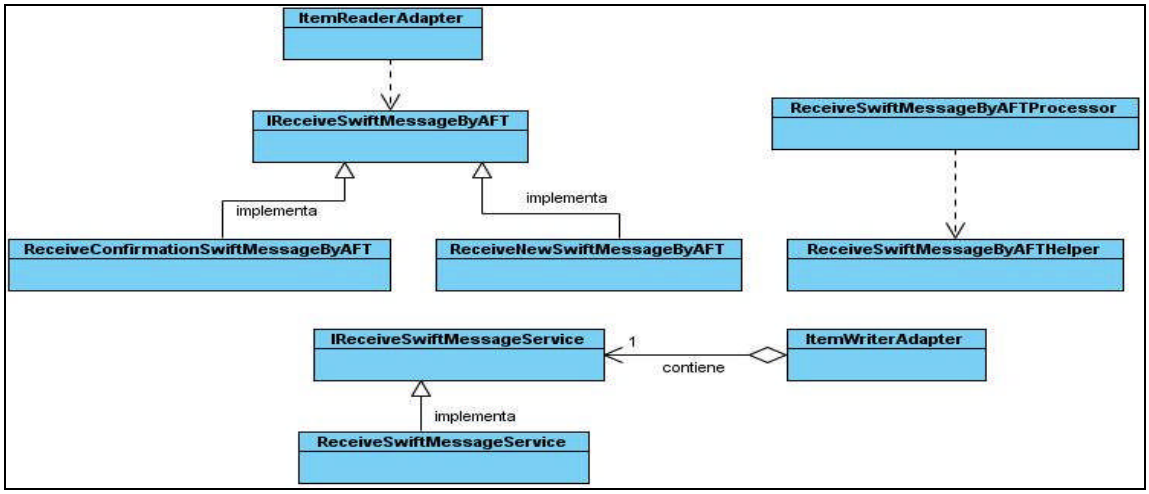



**Figura 31: Diagrama de clases recibir mensaje por AFT.**

Módulo Procesamiento

En el módulo de procesamiento radican los adaptadores necesarios para comunicarse con la aplicación empresarial. Estos adaptadores son: Por evento, Cola de mensajes, Invocación remota por RMI y Servicios web.

En este caso el procesamiento de mensaje es realizado por el sistema a través de una tarea programada.

Para dar solución a esta problemática se utilizó clases Java que permiten configurar una tarea programada y Spring Batch para guiar las ejecuciones de las funcionalidades necesarias. Se utiliza componentes de SF, Spring JMS y Spring RMI para desarrollar los adaptadores.

Se definió una interfaz que establece la forma de enviar los datos extraídos del mensaje a la aplicación empresarial. Esta interfaz se llama ISwiftMessageDataSender. Las clases que la implementan son SwiftMessageDataSenderPublishEvent y SwiftMessageDataSenderRMIClient, SwiftMessageDataSenderJMS SwiftMessageDataSenderWebServiceClient. En la figura 32 se muestra la jerarquía de clases explicada anteriormente.

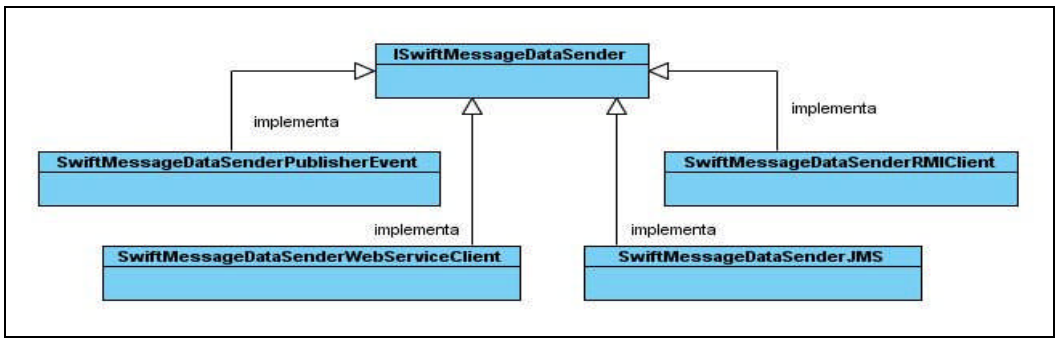



**Figura 32: Diagrama de clases para enviar los datos extraídos de los mensajes procesados hacia a la aplicación de negocio.**

**Seguridad del sistema**

El éxito de toda empresa depende de la calidad de la información que genera y gestiona. La información es de calidad si posee: Confidencialidad, Integridad y Disponibilidad.

Los datos son generalmente el activo informático más preciado para cualquier institución. El hardware y el software pueden ser caros pero fáciles de reponer, en cambio los datos o la información contienen la vida de una institución y su valor a veces es incalculable. Existen 3 características básicas de la seguridad en la información:

- Confidencialidad: la información es accedida solamente por las personas autorizadas para hacerlo.
- Integridad: La información solo pueden ser modificada por las personas autorizadas y de la forma autorizada.
- Disponibilidad: la información solamente es accedida por las personas autorizadas en el momento requerido. [24]

Para esto se le añadió al sistema un componente de Seguridad, se calcula un código Hash al contenido del mensaje y se administra los permisos de procesar (conformar,

actualizar, consultar y eliminar) el mensaje a SWIFT por usuario. A continuación se describe cada uno de estos elementos.

Componente de seguridad

Con relación a la Seguridad del sistema, este tiene integrado un componente de seguridad que fue desarrollado en el mismo proyecto; pero por otro equipo de desarrollo. Por su importancia se describen los elementos fundamentales del mismo.
El componente de seguridad está dividido en 2 partes, ver figura 33:

- Núcleo de la seguridad.
- Administración de la seguridad.

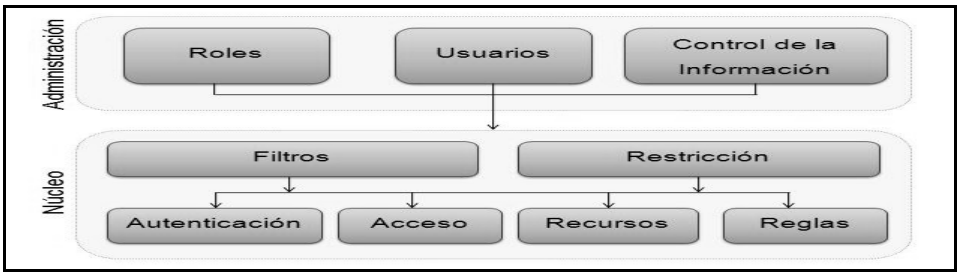



**Figura 33: Estructura del componente de Seguridad.**

El núcleo posee las funcionalidades y configuraciones utilizadas para brindar seguridad al sistema. Se utilizó Spring Security como base; pero fue necesario su modificación para que se pudiera realizar una gestión en cuanto a los roles y usuarios del sistema.
La administración de la seguridad se encarga de la parte visual del módulo. Presenta las funcionalidades de gestionar usuarios y roles, cambiar contraseña y asignarle roles a los usuarios.
La seguridad en el sistema se garantiza mediante filtros. Estos filtros son ejecutados antes de que el servidor atienda la petición realizada por el cliente.

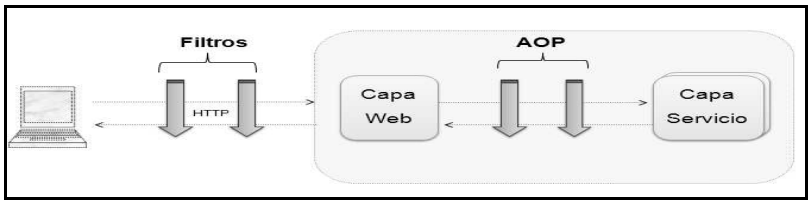



**Figura 34. Filtros de la seguridad.**

Los filtros se traducen en componentes que ejecutan ciertas restricciones. Los filtros utilizados en esta solución son: ChannelProcessingFilter, ConcurrentSessionFilter, HttpSessionContextIntegrationFilter, LogoutFilter, AuthenticationProcessingFilter, ExceptionTranslationFilter y FilterInvocationInterceptor. ChannelProcessingFilter se

encarga de re direccionar las peticiones a un canal seguro (HTTPS). ConcurrentSessionFilter comprueba el tiempo que el usuario ha pasado inactivo en el sistema. El sistema saca el usuario autenticado en el caso que haya superado el tiempo de inactividad establecido. HttpSessionContextIntegrationFilter se encarga de guardar en la sesión el usuario autenticado. LogoutFilter atiende la petición de sacar un usuario autenticado del sistema. AuthenticationProcessingFilter se encarga de procesar los datos usuario y contraseña cuando se están autenticando en el sistema, así como verificar la dirección IP de la maquina cliente. ExceptionTranslationFilter se encarga de bloquear una página y mostrar un pop up para que se introduzca usuario y contraseña cuando un usuario trata de acceder a una dirección web del sistema sin tener una sesión creada. FilterInvocationInterceptor se encarga de verificar si el usuario autenticado tiene permisos suficientes para realizar la petición que solicita.

Código Hash

Un valor hash es un valor numérico de longitud fija que sólo identifica datos. Los valores hash se utilizan con firmas digitales porque representan una gran cantidad de datos como un valor numérico mucho menor. Puede firmar con eficacia un valor hash en lugar de firmar el valor mayor. Los valores hash también resultan útiles para comprobar la integridad de datos enviados a través de canales inseguros. El valor hash de los datos recibidos puede compararse con el de los datos enviados para determinar si se alteraron [25].

En este trabajo se utiliza el código Hash para comprobar si fue alterado el contenido del mensaje. El proceso es el siguiente: una vez que se conforma el mensaje se le calcula un código Hash a través de una DLL [10] brindada por el BCC y se guarda en la base de datos como un atributo más del mensaje. En caso que se actualice el contenido del mensaje se realiza las mismas tareas. Cuando un mensaje se va a enviar se verifica que no haya sido alterado su contenido a través de la comprobación de su código Hash, en caso de detectar un cambio no se envía y se guarda el mensaje en el estado indicado para que sea tratado como tal. En la figura 35 y 36 se muestra de forma general los pasos ejecutados para calcular y comprobar el código Hash del mensaje.

[10] DLL: Biblioteca de enlace dinámico. Son ficheros con código ejecutable que se utilizan cuando un programa del sistema operativo lo necesita. Estos ficheros son únicos en los sistemas operativo de Windows.

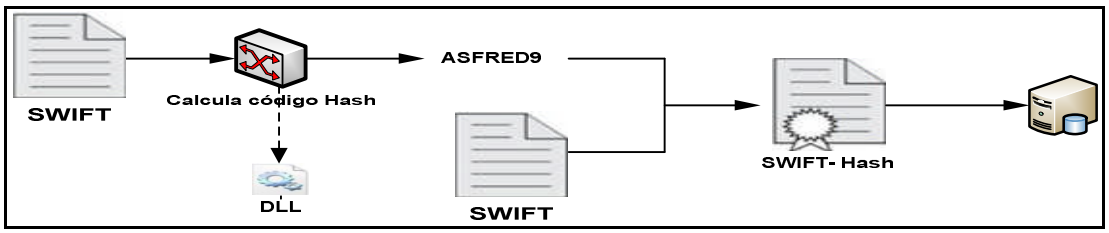



**Figura 35. Calculo del código Hash al contenido del mensaje SWIFT.**

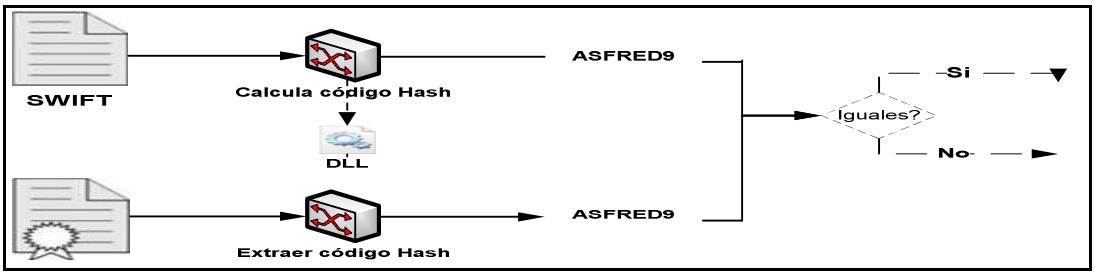



**Figura 36. Comprobación del código Hash del contenido del mensaje SWIFT.**

Permisos sobre el mensaje SWIFT

Se diseñó e implementó un componente que permite seleccionar los usuarios y los mensajes que estos pueden procesar. Con esto se cumple con el requisito definido en la sección 2.2: Gestionar permisos para procesar mensaje SWIFT y se garantiza que solo los usuarios asociados a procesar mensajes pueden trabjar con estos.

Las clases creadas para esto fueron: UsuarioMensajeServiceFlow, UsuarioMensajeFacade, UsuarioMensajeFacadeImpl, MessageSwiftFormatDAO, UsuarioDAO, MessageSwiftFormatDAOImpl, UsuarioDAOImpl, Usuario y MessageSwiftFormat.

A continuación se muestra en la figura 37 un diagrama con las clases mencionadas y sus relaciones.

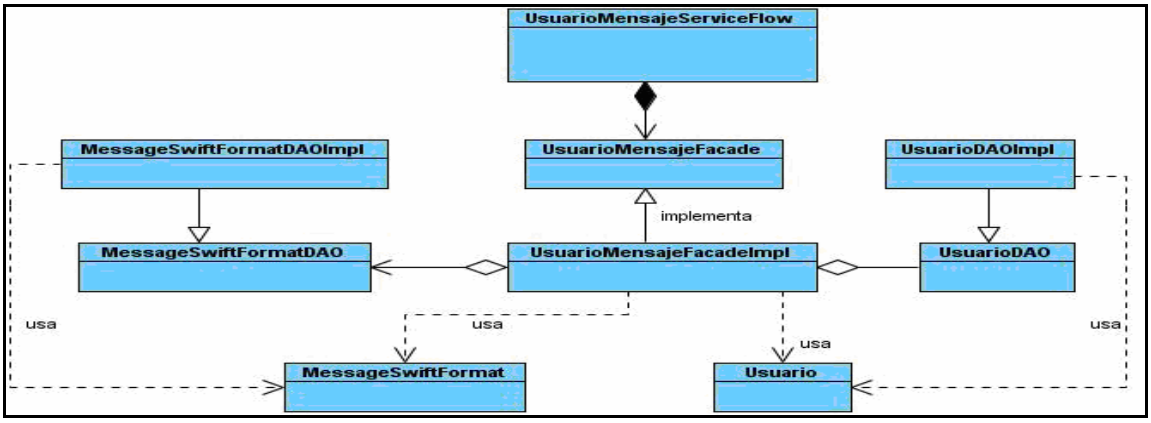



**Figura 37. Diagrama de clases para establecer permisos de usuario para procesar mensaje SWIFT.**

Modelo de despliegue

El sistema presenta dos entornos de despliegue. La aplicación podría funcionar integrado con la aplicación empresarial y el servidor que se comunica con la red SWIFT o podría funcionar integrado con la aplicación empresarial y el SISCOM. En las figura 38 y 39 se muestran cada una de las variantes de despliegue del sistema. Ambos entornos de despliegue requieren una arquitectura cliente-servidor. Las funcionalidades disponibles en los módulos de interacción con el usuario serán accedidas a través del protocolo HTTPS. La integración del sistema se realizará a través de los adaptadores descritos en la sección Modelo de Diseño e Implementación. Se utilizará una base de datos propia del sistema y en dependencia del entorno de despliegue seleccionado se utilizará la base de datos del SISCOM.

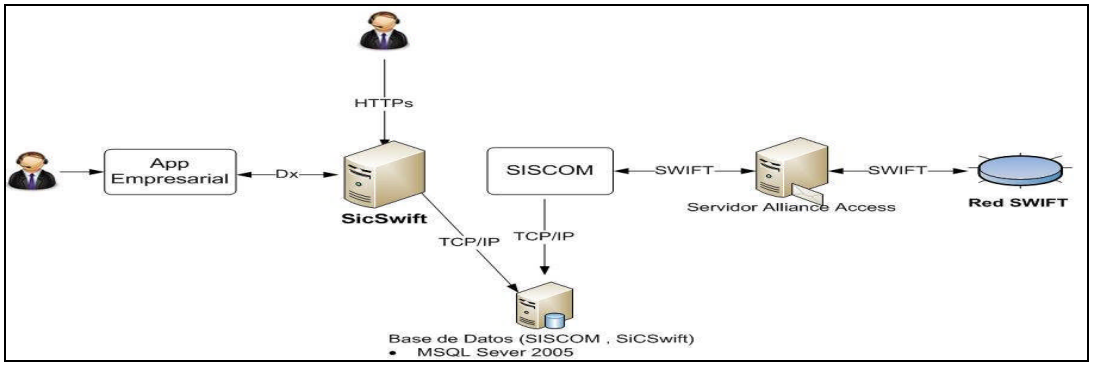



**Figura 38: Modelo de despliegue del SicSwift con la aplicación empresarial y el SISCOM.**

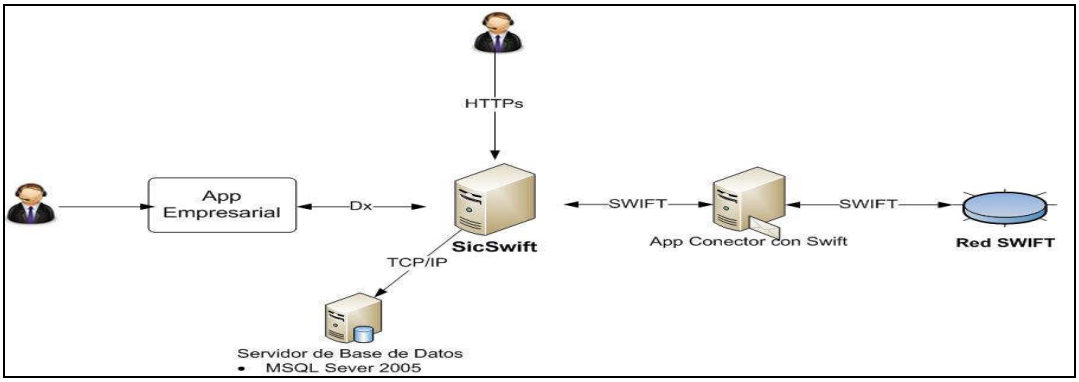



**Figura 39. Modelo de despliegue del SicSwift con la aplicación empresarial y la que se comunica con la red SWIFT.**

A continuación se describen los requerimientos de software y hardware necesarios para el despliegue del sistema. La máquina Cliente debe tener un procesador Pentium IV o superior con 512 Megabyte de RAM, una tarjeta de red y una impresora. Necesita tener instalado el Mozilla Firefox 3.6 y la máquina virtual de java 6.0u23. Se deben

utilizar dos servidores unos para desplegar la aplicación y otro para el gestor de base de datos. Los servidores deben tener un procesador Core 2 duo, memoria de cuatro Gigabyte, un disco duro de 160 Gigabyte, una tarjeta de red y un lector de DVD. Con relación al software deben tener instalados Windows Server 2003, Apache Tomcat 7.0 de 64 bit, máquina virtual de Java 6.0u23 de 64 bit, DLL para calcular CRC y Microsoft SQL Server 2005.

## 2.4 Conclusiones del capítulo

El sistema tiene implementado los requisitos funcionales identificados y cumple con los no funcionales. Brinda las funcionalidades necesarias para crear, actualizar y eliminar el formato de los mensajes SWIFT y las reglas semánticas. Se implementaron también varios adaptadores que permiten integrarse con otras aplicaciones. Gracias a esto, la aplicación puede utilizarse en diferentes entornos de despliegue, utilizando el SISCOM o integrándose directamente con el servidor que está conectado con la red SWIFT. Junto con la capacidad de integración que tiene el sistema, también se implementó un mecanismo de transformación que permite extraer los datos que son introducidos y manejados en el negocio para utilizarlos en la creación de mensaje de manera automática, disminuyendo el esfuerzo de los trabajadores y el riesgo de cometer errores. Esto también posibilita que el hecho contable se registre en el momento que ocurre. Las tecnologías utilizadas en el desarrollo de la aplicación fueron muy útiles porque agilizaron su construcción. Es válido destacar que el framework SF constituye una herramienta base importante para desarrollar las llamadas aplicaciones empresariales. Otra tecnología que arrojó resultados positivos fue Spring Webflow, pues permitió desarrollar flujos de presentaciones muy complejas de una manera sencilla y rápida. Spring WS, Spring JMS y Spring RMI son otras de las tecnologías utilizadas y que facilitaron en este caso el desarrollo de adaptadores. Mención además para Spring Batch, herramienta muy buena para ejecutar tareas que tienen un gran procesamiento de datos.

# 3. VALIDACIÓN DE LA SOLUCIÓN

## 3.1 Pruebas de software

Las pruebas de software tienen como objetivo convencer a los desarrolladores del sistema y a los clientes de que el software es lo suficientemente bueno para su uso operacional [22]. En [22] se divide este objetivo en dos más específicos:

1. Descubrir defectos en el software en que el comportamiento de este es incorrecto, no deseable o no cumple su especificación.
2. Demostrarle al desarrollador y al cliente que el software satisface sus requerimientos.

Cualquier producto de ingeniería (y de muchos otros campos) puede probarse de una de estas formas [26]:

1. Conociendo la función específica para la que fue diseñado el producto, se pueden llevar a cabo pruebas que demuestren que cada función es completamente operativa y, al mismo tiempo buscando errores en cada función.
2. Conociendo el funcionamiento del producto, se pueden desarrollar pruebas que aseguren que “todas las piezas encajan”, o sea, que la operación interna se ajusta a las especificaciones y que todos los componentes internos se han comprobado de forma adecuada.

El primer enfoque se denomina prueba de caja negra y el segundo, prueba de caja blanca.

Existen varias técnicas para ejecutar pruebas de caja blanca. Entre ellas están:

1. Prueba del camino básico
2. Prueba de la estructura de control, dentro de estas están la prueba de condición, prueba de flujo de datos y prueba de bucle.

Con relación a las pruebas de caja negra, estas se centran en los requisitos funcionales del software. Los errores que pueden encontrarse aplicando este tipo de prueba están agrupados en las categorías siguientes [26]: funciones incorrectas o ausentes, errores de interfaz, errores en estructuras de datos o en acceso a base de datos externas, errores de rendimiento y errores de inicialización y terminación.

Existen diferentes técnicas para ejecutar pruebas de caja negra como lo existen en las cajas blancas: Estas son [26]: Pruebas basadas en grafo, Partición equivalente, Análisis de valores límites, Pruebas de comprobación y Prueba de la tabla ortogonal.

Las pruebas se desarrollaron siguiendo las etapas del llamado modelo V de pruebas, el cual consiste en ejecutar primeramente las pruebas de unidad, luego las pruebas de integridad y por último las pruebas de alto nivel [26].

Las pruebas de unidad fueron ejecutadas de manera informal por el equipo de desarrollo cada vez que se culminaba el desarrollo de un componente. Para ejecutar estas pruebas se utilizaron las técnicas de las pruebas de caja blanca. En la medida en que se obtuvieron otros componentes se ejecutaron de manera informal también las pruebas de integración por un integrante del equipo de desarrollo. Una vez terminado el sistema se pasó a ejecutar las pruebas de alto nivel. Aquí se utilizaron las pruebas de liberación que ejecuta CALISOFT[11] utilizando la técnica de partición equivalente del método de prueba caja negra. Las pruebas que se aplican en CALISOFT verifican si el software en cuestión satisface los requisitos funcionales, de comportamiento y de rendimiento.

**Diseño de caso de prueba**

A continuación se expone el diseño de un caso de prueba que se realizó para probar la funcionalidad: Registrar mensaje SWIFT asociado a una operación bancaria.

Primeramente se identificaron las clases válidas (valores válidos) e inválidas (valores inválidas) para cada condición de entrada. En la tabla 7 se puede observar esto.

| No | Nombre de campo | Clasificación | Nulo o vacío | Descripción | Clases válidas | Clases inválidas |
|---|---|---|---|---|---|---|
| 1 | Mensajes | Checkbox | No | Se muestran varios checkbox con los números de los mensajes asociados a la operación para seleccionar los que serán enviados. | (1) seleccionar uno de los mensajes mostrados en la pantalla. | (2) No seleccionar uno de los mensajes mostrados en pantalla. |
| 3 | Prioridad | Lista desplegable | No | Este campo es una lista donde se selecciona unos de los elementos | (3) Seleccionar uno de los valores mostrados en pantalla. | (4) No seleccionar uno valores mostrados en pantalla. |

[11] CALISOFT: Centro nacional de calidad para soluciones informáticas.

| | | | | | | |
|---|---|---|---|---|---|---|
| 4 | Destino | Lista desplegable | No | Este campo es una lista donde se selecciona unos de los elementos | (5) Seleccionar uno de los valores mostrados en pantalla. | (6) No seleccionar uno valores mostrados en pantalla. |
| 5 | Monitoreo | Lista desplegable | No | Este campo es una lista donde se selecciona unos de los elementos. | (7) Seleccionar uno de los valores mostrados en pantalla. | (8) No seleccionar uno valores mostrados en pantalla. |
| 6 | Texto del sub-campo o campo | Caja de texto | No | Valor de un campo o sub-campo del mensaje SWIFT | (9) Cualquier valor valido según la expresión SWIFT definida. | (10) No introducir texto. (11) Texto invalido según la expresión SWIFT definida en el estándar. |
| 7 | Valor definido por SWIFT para el sub- | Lista desplegable | No | | (12) Seleccionar uno de los valores mostrados en pantalla. | (13) No seleccionar uno valores mostrados en pantalla. |

**Tabla 7. Clases validas e invalidad por condiciones de entrada.**

Se muestra a continuación en la tabla 8, las secciones y los escenarios de pruebas definidos para el caso de prueba.

| Nombre de la sección | Escenarios de la sección | Flujo Central |
|---|---|---|
| **SC1:** Registrar mensaje SWIFT asociado a una operación bancaria. | **EC 1.1**: Registrar Encabezado con éxito. | 1. El sistema de mensajería recibe una petición para enviar mensajes teniendo en cuenta una operación registrada.<br>2. El sistema muestra una interfaz con un listado de todos los mensajes SWIFT que están asociados a dicha operación.<br>3. El usuario selecciona el o los mensaje(s) que se deben conformar.<br>4. El sistema muestra una interfaz con los mensajes seleccionados por el usuario y da la posibilidad de Registrar el encabezado a cada uno de ellos.<br>5. En caso de que varios mensajes tengan un mismo encabezado, el usuario tiene la posibilidad de seleccionar cuantos mensajes desee y seleccionar la opción “Registrar encabezado”.<br>6. El sistema, muestra la interfaz para registrar el encabezado de o de los mensaje(s).<br>7. El usuario introduce el encabezado del o de los mensaje(s) y el sistema valida los datos según son introducidos.<br>8. El sistema obtiene el o los encabezado(s) de cada mensaje, marca al mensaje en la interfaz visual indicando que se registró el encabezado. |

| | | |
|---|---|---|
| | | 9. Se le da la posibilidad al usuario de eliminar encabezado. |
| **SC2:** Registrar mensaje SWIFT asociado a una operación bancaria. (Registrar texto del mensaje) | **EC 2.1:** Registrar texto del mensaje. | 1. Se ejecutan los escenarios **EC 1.1 o EC 1.2.** Un mensaje es no automático cuando los datos que vienen del negocio no son suficientes para conformar el mensaje.<br>2. Cuando el o los mensaje(s) no son automáticos, el usuario debe seleccionar un mensaje con encabezado y seleccionar la opción "Siguiente".<br>3. El sistema muestra una interfaz con los datos de todos los posibles mensajes a enviar. En la interfaz existe un tab para cada mensaje a conformar.<br>4. El usuario selecciona el tab del mensaje al cual le introducirá los datos.<br>5. El sistema muestra el texto del mensaje con campos llenos y otros vacíos.<br>6. El usuario introduce los datos en los campos vacíos. En caso de existir varios mensajes a conformar el usuario debe llenar los datos de cada uno de manera independiente.<br>7. El sistema valida sintáctica y semánticamente los datos del mensaje.<br>8. En caso de no existir error el sistema guarda el o los mensaje(s) en estado "Conformado" y el sistema muestra un mensaje: Operación realizada satisfactoriamente.". |
| | **EC 2.2: Cancelar.** | 1. Cancela la operación |

**Tabla 8. Secciones y escenarios del caso de prueba.**

Luego se conformó la Matriz de datos, ver tabla 9, la cual permite diseñar los valores a introducir en cada escenario identificado y registrar el resultado de la prueba.

| Escenario | Mensajes | Prioridad | Monitoreo | Destino | Texto del sub-campo o campo | Valor definido por SWIFT para el sub-campo | Respuesta del Sistema | Resultado de la Prueba |
|---|---|---|---|---|---|---|---|---|
| **EC 1.1:** Registrar Encabezado con éxito. | V | V | V | V | NA | NA | Se registra el encabezado satisfactoriamente**.** | Se registra el encabezado satisfactoriamente**.** |
| **EC 2.1:** Registrar texto del mensaje. | NA | NA | NA | NA | V | V | El sistema muestra un mensaje: "Operación realizada satisfactoriamente. " | El sistema muestra un mensaje: "Operación realizada satisfactoriamente. " |
| | NA | NA | NA | NA | V | I | El sistema muestra el siguiente cartel: Debe seleccionar un valor definido por SWIFT. | El sistema muestra el siguiente cartel: Debe seleccionar un valor definido por SWIFT. |

| Escenario | Mensajes | Prioridad | Monitoreo | Destino | Texto del sub-campo o campo | Valor definido por SWIFT para el sub-campo | Respuesta del Sistema | Resultado de la Prueba |
|---|---|---|---|---|---|---|---|---|
| | NA | NA | NA | NA | I | V | El sistema muestra el siguiente cartel: Debe introducir un texto correcto según la expresión SWIFT. | El sistema muestra el siguiente cartel: Debe introducir un texto correcto según la expresión SWIFT. |
| **EC 2.2:** Cancelar | NA | NA | NA | NA | NA | NA | El sistema muestra el siguiente cartel: Operación cancelada. | El sistema muestra el siguiente cartel: Operación cancelada. |

**Tabla 9. Matriz de datos.**

Así mismo se diseñaron casos de prueba para el resto de las funcionalidades que fueron probadas por CALISOFT. El producto fue liberado en la tercera iteración de pruebas.

Las pruebas de aceptación se realizaron posteriormente y en la actualidad el sistema está en despliegue. Las funcionalidades del sistema también fueron analizadas por especialistas del BCC, donde la gerente de operaciones y encargada de los sistemas de intercambio de información financiera emitió certificado avalando la nueva solución.

## 3.2 Validación y evaluación de la solución

Cuando se ha desarrollado una solución se piensa que está correcto y se tiene además una hipótesis de la solución. Por lo que es necesario validar la solución para que sea aceptado por la comunidad científica. Los siguientes patrones Demostración, Experimentación, Simulación, Uso de métricas, Evaluación comparativa, Razonamiento lógico y Pruebas matemática son unas de las vías para validar la solución desarrollada [27]. Para la validación de la solución se seleccionó la Experimentación y el Razonamiento lógico.

Se validarán las características del sistema que impactan directamente con los objetivos impuestos en el desarrollo de este trabajo. Los aspectos a validar son:

1. Solución capaz de soportar los cambios que ocurren en la norma ISO 15022.
2. Brindar mecanismos necesarios para que se registre el hecho económico en el momento que ocurre.
3. Solución capaz de controlar y restringir el acceso al contenido del mensaje SWIFT.

**Experimentación**

La experimentación tiene como objetivo validar o rechazar un conjunto de hipótesis asociada a la solución propuesta. Se debe utilizar este tipo de validación cuando no se puede probar las hipótesis planteadas matemáticamente o lógicamente [27].

Dentro de la experimentación existen diferentes tipos: Experimento hipotético/deductivo, Prototipos hermenéutica/inductivo, Basado en caso e Histórico.

Se aplicó el experimento hipotético/deductivo para evaluar el aspecto 1 por ajustarse a las condiciones necesarias para aplicarlas. A continuación se describe el experimento realizado.

Aspecto1: Solución capaz de soportar los cambios que ocurren en la norma ISO 15022

La norma ISO 15022 propone una sintaxis para estructurar mensaje y regla semántica con el ánimo de validar que los datos que posee el mensaje estén correctos y coherentes.

Los cambios que ocurren en la norma se caracterizan por ser: creación, eliminación o actualización de uno o varios elementos del mensaje.

Para realizar el experimento se escogió el mensaje MT 103 NOT STP, el cual sufrió cambio en el campo 52A y 70, en la figura 40 se puede constatar el cambio en el campo 52A. En la figura 44 la del campo 70. Los cambios se pueden observar a partir de lo siguiente:

- Lo que está en color rojo subrayado era el valor que estaba establecido anteriormente en el elemento SWIFT afectado.
- Lo que está en color azul es el nuevo valor que tiene el elemento SWIFT afectado.

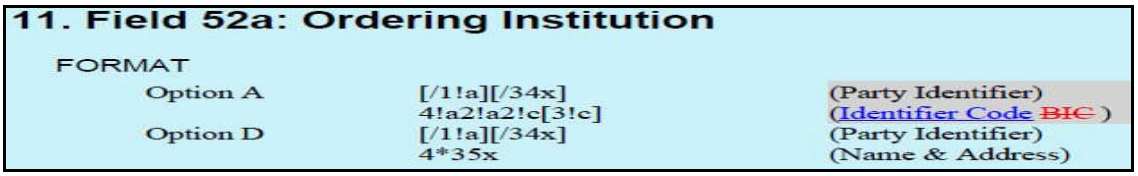



**Figura 40. Ejemplo de un cambio ocurrido en el campo 50A del mensaje MT 103 NOT STP.**

Como se muestra en la figura 40, el campo 50A tenía como sub-campo un valor BIC. La expresión SWIFT de un elemento BIC era el siguiente: 4!a2!a1!c[3!c]. La nueva expresión SWIFT es como se indica en la misma figura.

Para ejecutar tal cambio se interactúa con las funcionalidades que brinda el sistema. Lo primero que se realiza es buscar el bloque 4 del mensaje MT 103 NOT STP, luego

se selecciona el campo50A para que se muestre su estructura. Ver figura 41. Aparece en color amarillo el sub-campo que tiene la expresión SWIFT obsoleta.

| Elemento SWIFT | Significado | Notación SWIFT |
|---|---|---|
| componenteDosPuntos | - | : |
| identificadorCampo52A | - | 52A |
| componenteDosPuntos | - | : |
| subcampoDebito/Credito | - | [/[]] |
| subcampoParty | - | [/34x]CRLF |
| subcampoIdentifierCode | Corresponsal | 4!a2!a1!c[3!c] |
| retornoCarro | - | CRLF |

**Figura 41. Elementos SWIFT del campo 52A con la estructura antigua.**

Luego se procede a eliminar el elemento "retorno de carro y el sub-campo con la expresión obsoleta". Se crea un nuevo sub-campo con la nueva expresión SWIFT y se adiciona al campo 52A. Ver Figura 42 y 43.

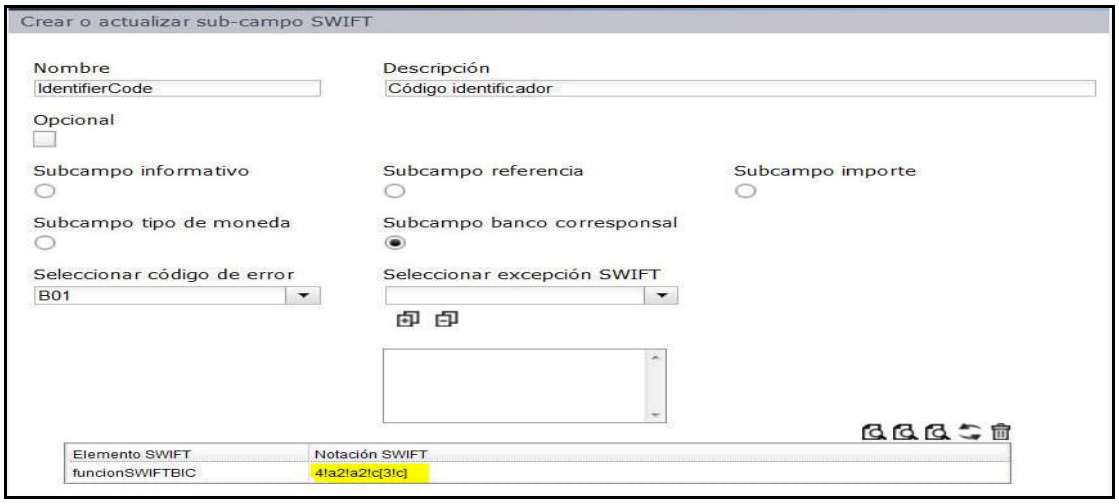


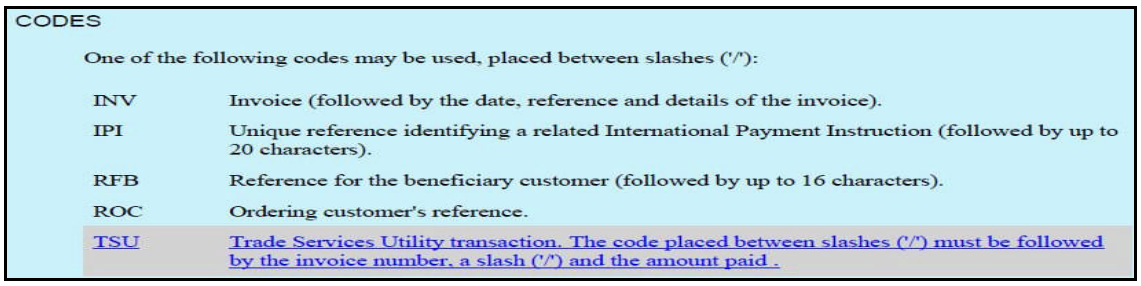

**Figura 42. Adicionando nuevo sub-campo con la nueva expresión SWIFT.**

| Elemento SWIFT | Significado | Notación SWIFT |
|---|---|---|
| componenteDosPuntos | - | : |
| identificadorCampo52A | - | 52A |
| componenteDosPuntos | - | : |
| subcampoDebito/Credito | - | [/[]] |
| subcampoParty | - | [/34x]CRLF |
| IdentifierCode | Corresponsal | 4!a2!a2!c[3!c] |

**Figura 43. Elementos SWIFT del campo 52A actualizados.**

El cambio que sufrió el campo 70 consistió en la adición del valor TSU a la lista de valores definidos. Ver figura 44.

La estructura del campo 70 del MT 103 NOT STP está compuesta por un sub-campo que recibirá unos de los códigos definidos en la figura 44, más un sub-campo de 30 caracteres x del alfabeto SWIFT y 3 líneas de 35 caracteres x.



CODES

One of the following codes may be used, placed between slashes ('/'):

| | |
|---|---|
| INV | Invoice (followed by the date, reference and details of the invoice). |
| IPI | Unique reference identifying a related International Payment Instruction (followed by up to 20 characters). |
| RFB | Reference for the beneficiary customer (followed by up to 16 characters). |
| ROC | Ordering customer's reference. |
| TSU | Trade Services Utility transaction. The code placed between slashes ('/') must be followed by the invoice number, a slash ('/') and the amount paid . |

**Figura 44. Códigos definidos para el campo 70 del MT 103.**

El cambio radica en actualizar el sub-campo definido en el campo 70 que soporta los códigos definidos. El primer paso es buscar los elementos SWIFT del campo 70 del MT 103. Ver figura 45.

| Elemento SWIFT | Significado | Notación SWIFT |
|---|---|---|
| componenteDosPuntos | - | : |
| identificadorCampo70Opcio | - | 70 |
| componenteDosPuntos | - | : |
| subcampo Codigo Campo70 MT103 | - | /ROC\|IPI\|INV\|RFB.../ |
| subcampo30X | - | 30x[CRLF34x]1-3 |
| retornoCarro | - | CRLF |

**Figura 45. Lista de elementos SWIFT sin el nuevo valor SWIFT añadido en el sub-campo Código Campo70 MT 103.**

Seleccionar la opción de actualizar el elemento SWIFT afectado y agregar el nuevo valor. Ver Figura 46. Se indica guardar los cambios del sub- campo y se adiciona el nuevo valor como se muestra en el Figura 47.

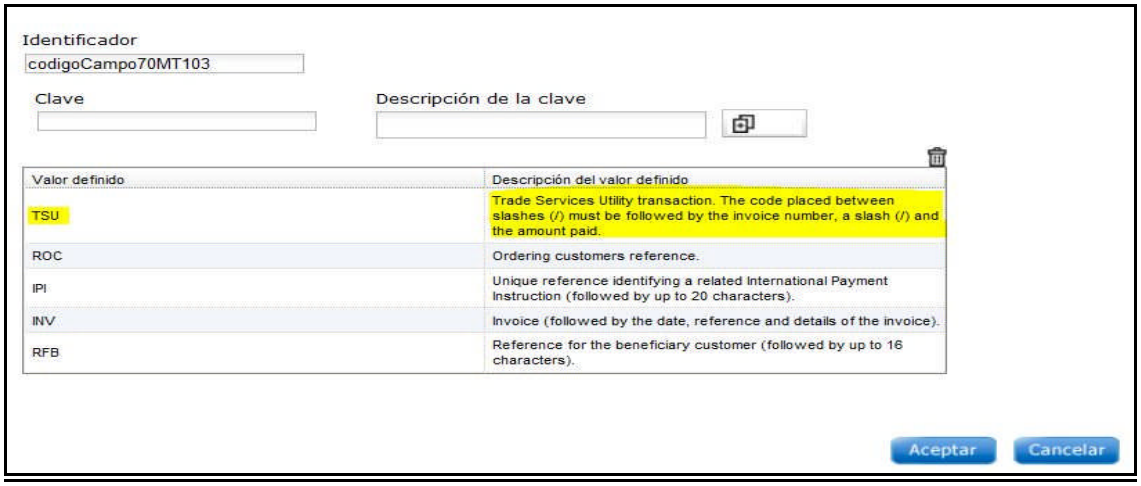



**Figura 46. Adicionando nuevo valor SWIFT.**

| Elemento SWIFT | Significado | Notación SWIFT |
|---|---|---|
| componenteDosPuntos | - | : |
| identificadorCampo70Opcio | - | 70 |
| componenteDosPuntos | - | : |
| subcampo Codigo Campo70 MT103 | - | /TSU\|ROC\|IP\|INV.../ |
| subcampo30X | - | 30x[CRLF34x]1-3 |
| retornoCarro | - | CRLF |

**Figura 47. Lista de valores definidos para el campo 70 del MT 103.**

**Razonamiento lógico**

Cuando no es posible usar un modelo matemático, cuando el problema es complejo o no se pueden establecer un prototipo o mecanismo que muestre la validez de la solución entonces es preciso utilizar argumentos lógico que valide la solución propuesta. En la bibliografía [27] se plantea que es necesario crear axiomas o reglas que deben cumplirse y que estén además relacionado con el problema en cuestión. Luego basándose en esa regla se realiza una explicación lógica de cómo la solución desarrollada cumple con tales axiomas o reglas.

Aspecto 2: Brindar mecanismos necesarios para que se registre el hecho económico en el momento en que ocurre

A continuación se brindan los argumentos necesarios para mostrar que el mecanismo implementado es suficiente para cumplir con el propósito planteado en el Aspecto 2.

Reglas que deben cumplirse en la solución:

1. El sistema mediador entre la comunicación con la red SWIFT y la aplicación empresarial debe brindar los servicios necesario para conformar mensaje SWIFT.
2. Los servicios brindados por el mediador con la comunicación SWIFT deben exponerse a través de adaptadores o puntos de integración implementados, para así facilitar la comunicación con la aplicación empresarial.
3. El sistema mediador entre la comunicación con la red SWIFT y la aplicación empresarial debe tener un mecanismo de transformación de datos para tomar los datos introducidos en la aplicación empresarial y crear la estructura necesaria para conformar mensaje SWIFT.
4. El sistema no debe impedir ni intervenir en la contabilización de los datos manejados en la aplicación empresarial.

La solución implementada cumple con la regla 1. Se exponen un total de 11 servicios entre los más importantes están:

- Listar los mensajes que podrían crearse dado la ejecución de una operación de negocio.
- Listar los tipos de prioridad y formas de monitorear los mensajes.
- Listar los bancos a los que se le puede enviar mensajes.
- Construir mensaje a partir de los datos enviados por la aplicación empresarial
- Validar y guardar mensaje SWIFT una vez concluido su conformación.

Ver los demás servicios expuestos en el Capítulo 2, sección 2.3 Modelo de Diseño e Implementación, sub-sección Módulo Conformación.

Cumpliendo con la regla 2, los servicios son expuestos mediante tres adaptadores: Interfaz y clase Java, Web Services e Invocación remota por RMI.

Estos son explicados en el Capítulo 2, sección 2.3 Modelo de Diseño e Implementación sub-sección Módulo Conformación.

Con relación a la regla 3, en el servicio “construir mensaje”, mencionado anteriormente, se ejecuta un mecanismo de transformación implementado para conformar mensajes. A través de este se extraen los datos enviados por la aplicación empresarial y se colocan en el lugar indicado según un fichero XSL. Luego de crear la estructura del mensaje en XML, se convierte ese XML en un objeto que representa el mensaje conformado. Este objeto se devuelve a la aplicación empresarial para que este lo muestre al usuario y sea completado por el propio usuario. En el Capítulo 2, sección 2.3 Modelo de Diseño e Implementación sub-sección Módulo Transformación.

Y por último cumpliendo con la regla 4, una vez completado el mensaje el usuario debe indicar guardar los datos y contabilizarlos. En este instante la aplicación empresarial debe contabilizar los datos que son manejados en tal proceso y enviar el mensaje conformado al sistema que se está analizando para que este lo valide y guarde. De esta forma se registra el hecho económico en el mismo momento en que se conforma, valida y guardan los mensajes SWIFT.

Aspecto 3: Solución capaz de controlar y restringir el acceso al contenido del mensaje SWIFT

A continuación se brindan los argumentos necesarios para mostrar que el componente de seguridad implementado es suficiente para cumplir con tal propósito.

En la sección 2.2 Requisitos del sistema del Capítulo 2 se definieron los requisitos no funcionales que debía cumplir el sistema. Entre ellos estaban requisitos relacionados con la seguridad del sistema. A continuación se expresan como reglas, ya que estas se utilizarán como base para argumentar lógicamente cómo el componente de seguridad utilizado es suficiente para cumplir con tal propósito:

1. Cualquier persona que interactúe con el sistema debe ser un usuario activo registrado en la base de datos del sistema y previamente tuvo que autenticarse en el sistema.
2. Se deben mostrar las funcionalidades que el usuario autenticado tenga permiso para ejecutar.
3. El intercambio de información de computadora cliente a servidor y viceversa se debe realizar a través de un canal seguro.
4. Debe comprobarse a través de un algoritmo de encriptación si se modificó el contenido del mensaje.
5. Cada vez que se genere un mensaje y cambie el estado del mismo se registrará el usuario que lo ha realizado.

Con relación a la regla 1, el componente de seguridad utilizado en la solución muestra una interfaz gráfica que permite registrar usuarios al sistema. Unos de los datos que debe introducir es el rol o roles que tendrá el usuario y si será un usuario activo del sistema. A partir de esto la solución comprueba si el usuario que realizo una petición desde una maquina cliente está activo en el sistema. Ver figura 48, donde se muestra la ventana para registrar un usuario del sistema.


**Figura 48. Registrar usuario en el sistema.**

La regla 2 se cumple a partir de la posibilidad que brinda el componente de seguridad de configurar las funcionalidades que puede ejecutar un rol. Luego de registrar los roles con los permisos necesarios, estos son asociados a los usuarios cuando se crean, Ver figura 48. En cada pantalla del sistema se utiliza un script que comprueba si el usuario autenticado tiene permiso suficiente para ejecutar las funcionalidades.

Por otra parte, como el sistema se ejecuta en un entorno web, las computadoras o máquinas clientes realizan peticiones HTTP al servidor que contiene la lógica de la aplicación. En estas peticiones viaja tanto la información que introduce el operador en la máquina cliente como la información que devuelve el servidor respondiendo a una petición. Esta información en muchos casos es sensible para la institución por lo que se decidió utilizar el protocolo HTTPS, el cual utiliza el canal seguro SSL[12]. La comunicación a través de HTTPS se garantiza a través del componente ChannelProcessingFilter, utilizado en la construcción del componente de Seguridad y disponible en el framework de Spring Security. De esta forma se asegura la información que se transfiere a través del protocolo HTTP y se cumple con la regla 3.

Con el objetivo de cumplir con la regla 4, se utilizó una DLL que permite calcular un código Hash al contenido del mensaje. Una vez que se procese un mensaje se comprueba primero si su código Hash coincide con el que tiene guardado. Este código Hash calculado se utiliza para identificar si su contenido fue alterado por otra vía. En el Capítulo 2, sección Modelo de Diseño e Implementación, sub-sección Seguridad del sistema se explica cómo se realiza el cálculo del código Hash.

Para cumplir con la regla 5, se agregó un atributo llamado COD_USUARI en la tabla o_estado_mensaje_SWIFTFIN. Ver Anexo 2. Este atributo va a guardar el código del usuario del usuario que conforme, manipule el contenido del mensaje o cambie el estado del mismo.

Es válido mencionar que el componente de Seguridad también fue liberado por CALISOFT, por lo que todas sus funcionalidades responden a los requerimientos establecidos.

## 3.3 Encuesta realizada a operadores del BNC

Se realizó una encuesta con el objetivo de verificar y validar que la solución desarrollada cumplía con los objetivos definidos inicialmente. Esta solución se ejecuta

[12] SSL: Capa de conexión segura. Es un protocolo criptográfico que proporciona comunicación segura por una red.

en el BNC como un subsistema del software SAGEB, desarrollado también en la UCI. La encuesta se les realizó a 10 personas que constituían alrededor del 30 por ciento de las personas que operaban con mensaje SWIFT. En la encuesta se vieron involucrados todos los departamentos que trabajan con mensaje SWIFT. Las preguntas realizadas fueron cerradas, se respondían afirmativa o negativamente. A continuación se exponen cada una de ellas:

1. ¿Puede usted acceder al contenido de un mensaje SWIFT por otra vía que no sea a través de este sistema?
2. ¿Puede usted acceder usted a las funcionalidades del sistema antes de autenticarse en el sistema?
3. ¿Con las nuevas funcionalidades del sistema usted introduce solamente una vez los datos contables cuando tiene que conformar mensaje SWIFT?
4. ¿Las funcionalidades del sistema brinda la posibilidad de contabilizar y al mismo tiempo conformar mensaje SWIFT?

Todas las personas encuestadas respondieron negativamente las dos preguntas iniciales y afirmativamente la tercera y cuarta. Estos resultados validan también que el sistema impide el acceso al contenido del mensaje SWIFT y a sus funcionalidades deliberadamente y corrobora que en el proceso de conformación del mensaje también se contabilizan los datos contables.

## 3.4 Publicaciones y eventos

Los resultados de este trabajo fueron presentados y publicados en la XV Convención Científica de Ingeniería y Arquitectura celebrado en Diciembre de 2010 en el Palacio de las Convenciones, La Habana. Se presentó también en el Fórum de Ciencia y Técnica a nivel de Facultad en mayo 2011, obteniendo relevante como premio. Se publicó además un artículo en la serie Científica de la UCI, en diciembre de 2011 y otro artículo en el evento Uciencia 2012, celebrada en la UCI.

## 3.5 Conclusiones del capítulo

Las pruebas de software realizadas al sistema fue un instrumento importante para medir el grado de cumplimiento de los requisitos definidos inicialmente. Especialmente las pruebas de caja negra permitieron conocer si las funcionalidades implementadas se ejecutaban acorde a lo requerido. Por otra parte las pruebas de aceptación llegaron a

complementar las pruebas de liberación, pues estas propiciaron que el propio usuario del sistema validara con sus propias operaciones el funcionamiento del sistema.

El experimento y el razonamiento lógico son patrones de validación científicos establecidos. El experimento descrito arrojó que la nueva solución puede ajustarse a los cambios que ocurren en la norma; pues esta tiene las funcionalidades de adicionar, actualizar y eliminar la estructura de un mensaje SWIFT.

Se encuesto además a personas que trabajan directamente con el sistema y se pudo constatar que los objetivos propuestos fueron cumplidos desde el punto de vista de la opinión de los operadores del BNC.

Por otra parte se explicó a través del razonamiento lógico cómo la solución cumple con las 4 reglas que son necesarias para permitir el registro del hecho económico en el momento en que ocurre. También se puede plantear que el sistema tiene las funcionalidades y arquitectura necesaria para cumplir con las características de una solución informática segura: confiable, integridad y disponible.

En la actualidad, luego de haber pasado por una serie de pruebas, el sistema se utiliza en el BNC.

Se obtuvo una nueva solución informática desarrollada con tecnologías actuales y libres, con soporte para varios gestores de base de datos. Su despliegue está basado en una arquitectura cliente-servidor para aprovechar las ventajas que presenta este tipo de arquitectura.

La utilización de esta herramienta informática mejora la operatividad del proceso de intercambio de mensajes SWIFT; pues las diferentes formas de integración desarrolladas y el mecanismo de transformación desarrollado permiten disminuir el esfuerzo y riesgo de cometer errores por parte de los operadores del banco y registrar el hecho económico en el momento que ocurre; ya que en el instante que se envía uno o varios mensajes se puede además contabilizar la operación asociada a éstos.

Los cambios que ocurren en la norma del estándar SWIFT podrán ser actualizados a través de las interfaces gráficas del sistema y no será necesario modificar ni los datos de la base de datos, ni su código fuente.

El contenido de los mensajes no será visto por usuarios que no tengan los permisos necesarios para esto. Solo se accederá al contenido de los mensajes mediante las interfaces gráfica de la aplicación. El sistema constituye una herramienta que puede ser utilizada por cualquier entidad financiera que necesite enviar, procesar y recibir mensajes SWIFT compatible con la norma ISO 15022.

## CONCLUSIONES

La nueva solución informática, además de ser compatible con la norma ISO 15022, dispone de un grupo de interfaces gráficas que simplifica la actualización que debe realizársele a la norma ISO 15022 anualmente. A través de estas interfaces se puede gestionar la estructura de los mensajes y sus reglas semánticas. No es necesario modificar el código fuente ni interactuar directamente contra las tablas de base de datos que guardan las estructuras de los mensajes.

Se logra una disminución del esfuerzo y riesgo de cometer errores por parte de los operadores del sistema cuando ejecutan operaciones de negocio que llevan consigo la creación y procesamiento de mensajes. Permite además que no se viole el principio contable de registrar el hecho económico en el momento que ocurre. Todo esto es posible gracias al mecanismo de transformación y los adaptadores de integración implementados.

Otro resultado importante es que el mecanismo de transformación trabaja independiente de las operaciones de negocio que se ejecutan en la aplicación empresarial. En caso que cambien los datos utilizados en la transformación deben actualizar los ficheros de transformación asociados a estos. El sistema dispone de interfaces gráficas que facilitan este proceso.

En cuanto a la seguridad del sistema puede plantearse que:

- las funcionalidades del sistema son restringidas según los roles que posea el usuario
- no podrá accederse libremente al contenido del mensaje y
- se controlará el contenido del mismo a través de un algoritmo facilitado por el BCC.

El sistema en general fue desarrollado con tecnologías actuales y gratis. No está atada a un gestor de base de datos específico y constituye un producto reutilizable por cualquier entidad bancaria que necesite enviar, recibir y procesar información financiera a través del estándar SWIFT bajo la norma ISO 15022.

Las características del sistema resuelven los problemas que originaron el desarrollo del presente trabajo.

## RECOMENDACIONES

A pesar que el sistema es compatible con la norma ISO 15022 del estándar de mensaje SWIFT es recomendable en próximas versiones del sistema extender su compatibilidad con la norma 20022 ya que esta norma se está incorporando gradualmente al intercambio de mensaje SWIFT a nivel internacional.

## BIBLIOGRAFÍA REFERENCIADA


1. **Banco Nacional de Cuba.** *MANUAL DE INSTRUCCIONES Y PROCEDIMIENTOS.* Habana : s.n., 2007.

2. **XBRL**. *eXTensible Business Reporting Language*. [Citado el: 21 de 09 de 2011.] [En línea: http://www.xbrl.org/].

3. **FIX Protocol.** *FIX Protocol*. [Citado el: 21 de 09 de 2011.] [En linea: http://www.fixprotocol.org/].

4. **United Nations Economic Commission for Europe.** *United Nations Directories for Electronic Data Interchange for Administration, Commerce and Transport*. [Citado el: 21 de 09 de 2011.] [En línea: http://www.unece.org/trade/untdid/welcome.htm].

5. **SWIFT.** *SWIFT Corporation*. [Citado el: 12 de 1 de 2010.] [En línea: http://www.swift.com/about_swift/company_information/index.page?lang=es].

6. **SWIFT**. *Alliance Integrator*. [Citado el: 12 de 1 de 2010.] [En línea http://www.swift.com/solutions/connectivity/connectivity_products/alliance_integrator/index.page?lang=es].

7. **Prowide Software.** *WIFE Model*. [Citado el: 15 de 09 de 2011.] [En línea http://www.prowidesoftware.com/es/wife-documentacion/modelo-de-mensajes.html].

8. **Prowide Software**. *WIFE Parser*. [Citado el: 15 de 9 de 2011.] [En línea: http://www.prowidesoftware.com/es/wife-documentacion/parser.html].

9. **Damatation.** *Payment component*. [Citado el: 15 de 09 de 2011.] [En línea: http://www.paymentcomponents.com/development_tools/swift_mt_payments/swift_mt_payments_technical.html].

10. **AnaSys AG.** *MessageObjects*. [Citado el: 15 de 9 de 2011.] [En línea: http://www.anasys.com/products/messageobjects/].

11. **C24.** C24io. [Citado el: 15 de 09 de 2011.] [En línea: http://www.c24.biz/].

12. **SWIFT**. *Alliance Access*. [Citado el: 10 de 11 de 2009.] [En línea: http://www.swift.com/solutions/connectivity/connectivity_products/alliance_access/detailed_features.page?lang=es].

13. **TAS Group.** *Gari Gold TFM for Message Magement. Product Requirements and Configuration.* 2011. CNF-GG-MM.
14. **SIBANC.** *Manual de Usuario del SISCOM.* Habana : SIBANC, 2006.
15. **Linthicum, David S.** *Enterprise Application Integration.* s.l. : Addison – Wesley, 1999. 0201615835.
16. **Hohpe, Gregor y Woolf, Bobby.** *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions.* s.l. : Addison - Wesley, 2003. 0321200683.
17. **TechTarget Corporate.** *The ServerSide Application Server Matrix*. [Citado el: 16 de Marzo de 2011.] [En línea: http://www.theserverside.com/reviews/matrix.tss].
18. **Johnson, Rod, y otros.** *The Spring Framework – Reference Documentation Version 2.5.4.* s.l. : SpringSource Incorporated, 2008.
19. **The Apache Software Foundation.** *Apache WSS4J - Web Services Security for Java. Apache WSS4J.* [Citado el: 17 de Marzo de 2011.] [En línea: http://ws.apache.org/wss4j/].
20. **SpringSource.** *Spring Web Services - Reference Documentation.* 2011.
21. **Jacobson, Ivar, Booch, Grady y Rumbaugh, James.** *El Proceso Unificado de Desarrollo de Software.* Madrid : Addison Wesley, 2000.
22. **Sommerville, Ian.** *Ingeniería de software.* Madrid: Addison Wesley, 2005.
23. **Iglesias Chaviano, Adolfo Miguel.** *Documento de Arquitectura de Software.* Habana : s.n., 2010.
24. **María del Rocio Lemus Zamora**. *Seguridad de la información en las organizaciones. Caso sistema kardex de posgrado de la FCCC*. [Citado el 20 de Noviembre de 2011.][Publicado: 11 de Octubre de 2011.] [En línea: http://bibliotecavirtual.dgb.umich.mx:8083/jspui/bitstream/123456789/1652/1/SEGURIDADDELAINFORMACIONENLASORGANIZACIONESCASOSISTEMAKARDEXDEPOSGRADODELAFCCA.pdf].
25. **Microsoft**. *Asegurar la integridad de los datos mediante códigos hash*. [Citado el: 4 de Abril de 2011.][En línea: http://msdn.microsoft.com/es-es/library/f9ax34y5%28v=vs.80%29.aspx].
26. **Roger S. Pressman**. *Ingeniería de software: Un enfoque práctico. Parte 1. Quinta edición, Capítulo 17 y 18.* España: Mc Graw-Hill, 2002.
27. **Vaishnavi ,Vijay K. y Kuechler, William Jr**. *Design Science Research Methods and Patterns*. Estados Unidos: Auerbach Publications, 2008.

## BIBLIOGRAFÍA CONSULTADA


1. **Fisher, Mark, y otros, y otros.** *The Spring Integration Reference Manual Version 1.0.4.* s.l. : SpringSource Incorporated, 2010.

2. **ISO15022.** *ISO 15022*. [Citado el: 10 de 11 de 2009.] [En línea: http://www.iso15022.org/ISO15022XML/General/MessageStandardsEvolution.pdf].

3. **Oracle Corporation, Project Kenai, Cognisync.** *The Standard Implementation for JAX-RPC*. [Citado el: 17 de Marzo de 2011.] [En línea: https://jax-rpc.dev.java.net/].

4. **Poutsma, Arjen, Evans, Rick y Abed, Rabbo, Tareq.** *Spring Web Services - Reference Documentation Version 1.5.9.* s.l. : SpringSource Incorporated, 2007.

5. **SIBANC.** *Manual del Sistema de Liquidación Bruta en Tiempo Real.* 2008.

6. **SpringSource.** *Spring Security Project Home Site.* [Citado el: 17 de Marzo de 2011.] [En línea: http://static.springframework.org/spring-security/site/].

7. **SWIFT.** *Servicios de mensajería de SWIFT.* [PDF] 2008.

8. **SWIFT.** *User Handbook.* [PDF] 2008.

9. **TechTarget Corporate.** *JAX-WS (Java API for XML Web Services)*. *SearchSOA.com.* [Citado el: 18 de Marzo de 2011.] [En linea: http://searchsoa.techtarget.com/definition/JAX-WS].

10. **The Apache Software Foundation.** *Enterprise Integration Patterns*. *Apache Camel.* [Citado el: 20 de Abril de 2011.] [En línea: http://camel.apache.org/enterprise-integration-patterns.html].

11. **The Apache Software Foundation.** *WebServices – Axis.* [Citado el: 19 de Marzo de 2011.] [En línea:http://axis.apache.org/axis/].

## ANEXOS

**Anexo 1. Modelo entidad relacional para gestionar formato y reglas semánticas del mensaje SWIFT.**

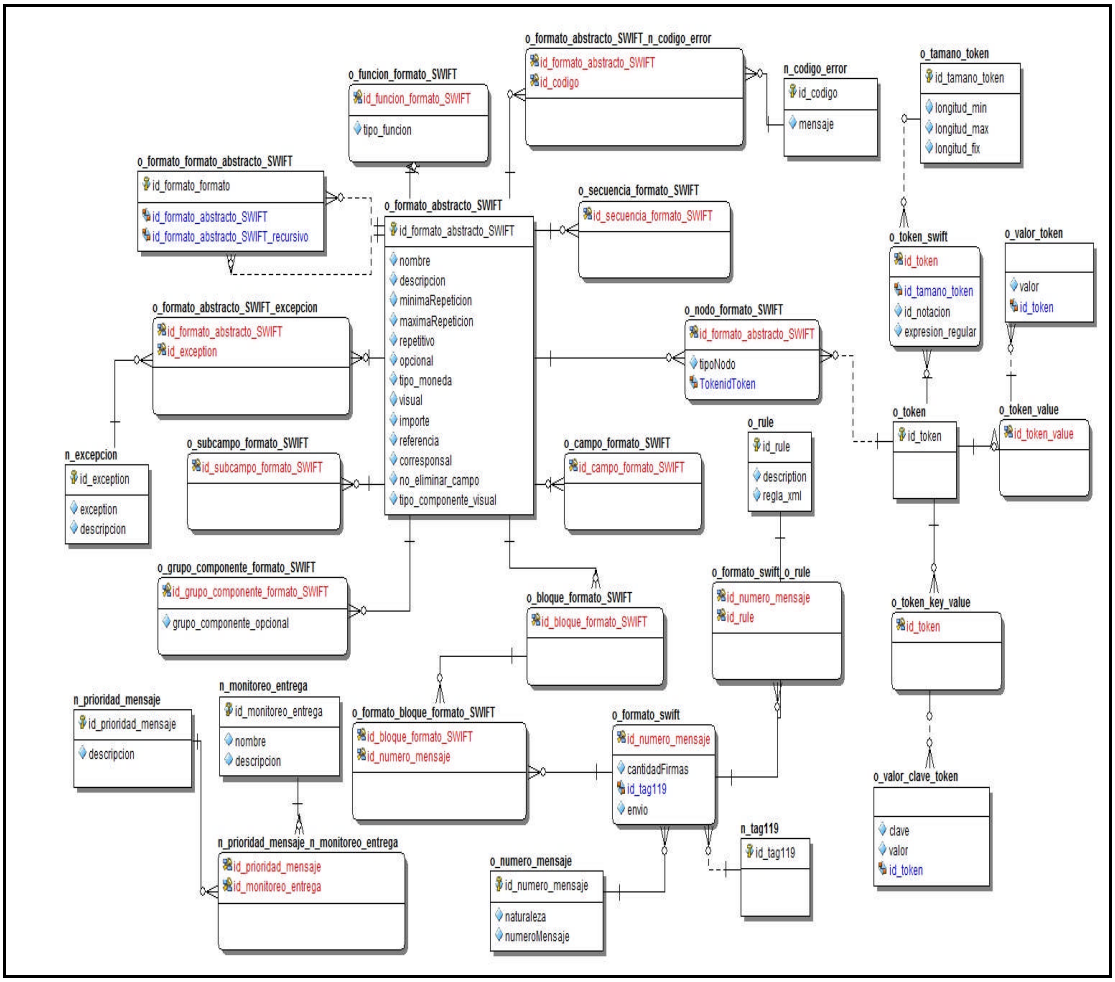



**Figura 49. Modelo entidad relacional para gestionar formato y reglas semánticas del mensaje SWIFT.**

**Anexo 2. Modelo entidad relacional para guardar los datos del mensaje y sus estados, así como la asociación de operación con mensaje SWIFT, las maquetas Swift y las firmas de mensaje a los usuarios.**

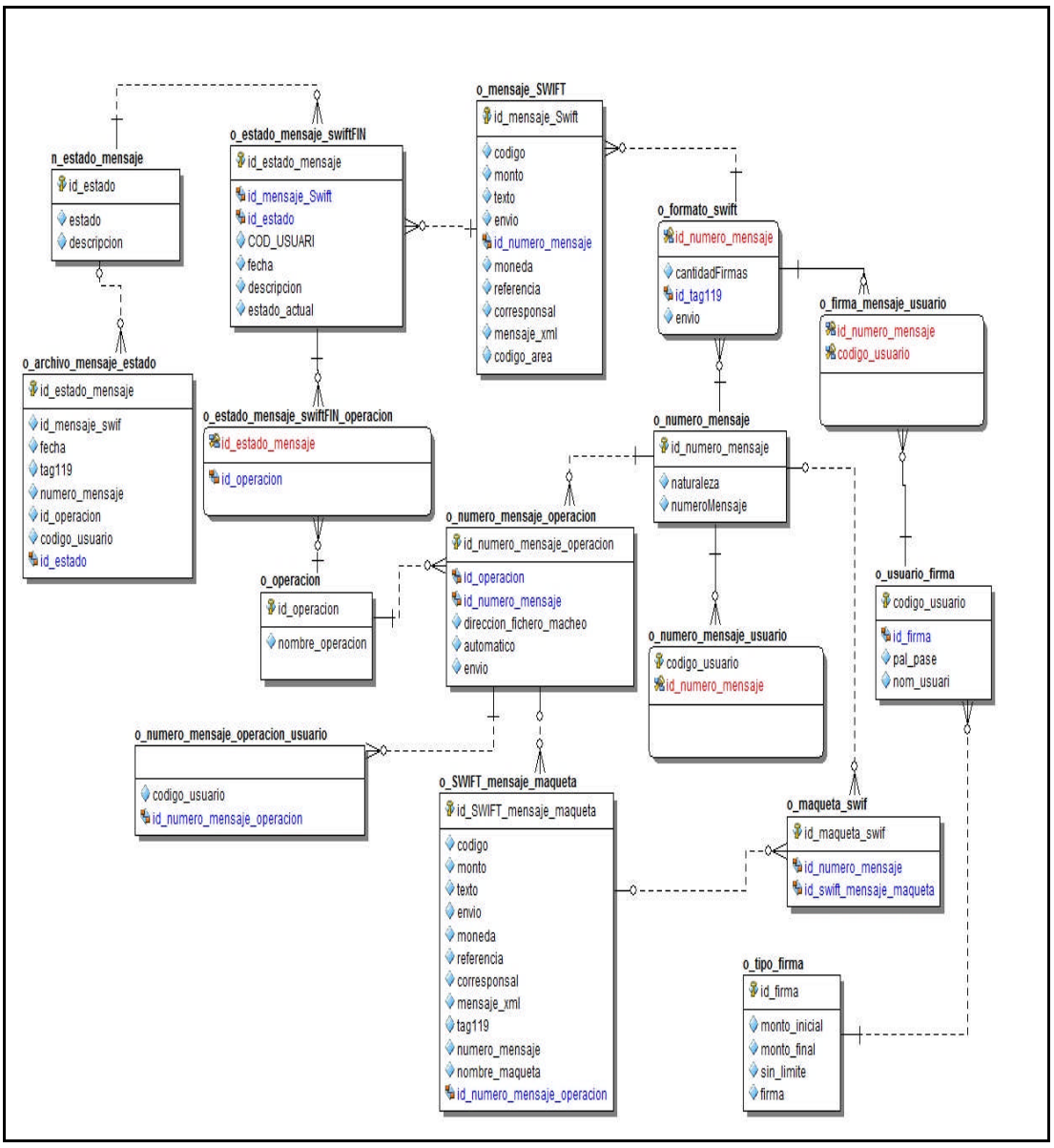



**Figura 50. Modelo entidad relacional para guardar los datos del mensaje y sus estados, así como la asociación de operación con mensaje SWIFT, las maquetas Swift y las firmas de mensaje a los usuarios.**