
cenidet

**Centro Nacional de Investigación y Desarrollo
Tecnológico**
Departamento de Ciencias Computacionales

TESIS DE MAESTRÍA EN CIENCIAS

**Sistema de Re-Ingeniería de Software en COBOL, en
Versiones No Estándar para su Reuso**

presentada por

Luisa Alba Aquino Bolaños

Ing. en Computación por la Universidad Tecnológica de la Mixteca

como requisito para la obtención del grado de:
Maestría en Ciencias en Ciencias de la Computación

Director de tesis:

Dr. René Santaolaya Salgado

Co-Director de tesis:

Dr. Guillermo Rodríguez Ortiz



cenidet

**Centro Nacional de Investigación y Desarrollo Tecnológico
Departamento de Ciencias Computacionales**

TESIS DE MAESTRÍA EN CIENCIAS

Sistema de Re-Ingeniería de Software en COBOL, en Versiones No Estándar para su Reuso

presentada por

Luisa Alba Aquino Bolaños

Ing. en Computación por la Universidad Tecnológica de la Mixteca

como requisito para la obtención del grado de:

Maestría en Ciencias en Ciencias de la Computación

Director de tesis:

Dr. René Santaolaya Salgado

Co-Director de tesis:

Dr. Guillermo Rodríguez Ortiz

Jurado:

M.C. Olivia Fragoso Díaz – Presidente

Dr. René Santaolaya Salgado – Secretario

Dr. Jaime Muñoz Arteaga – Vocal

M.C. Mario Guillén Rodríguez – Vocal Suplente

Cuernavaca, Morelos, México.

16 de Diciembre de 2010

"Hay una fuerza motriz más poderosa que el vapor, la electricidad y la energía atómica. Esa fuerza es la voluntad."

Albert Einstein

Dedicatorias

Dedico este trabajo especialmente para mi mamá Artura, porque siempre me consiente y tiene mucho cariño y amor para mí y porque es mi más fuerte motivación de superación y fortaleza.

A mi papá Antonio, por su cariño y apoyo en todo momento y porque siempre me dio un buen ejemplo de responsabilidad.

A mi hermana Nora porque es además mi mejor amiga, siempre me ha apoyado en todas mis decisiones y es mi más grande ejemplo a seguir. De la misma manera a su esposo Rubén porque siempre me ha visto y tratado como su hermanita.

A mi hermana Hilda, porque siempre he admirado su inagotable fortaleza para superarse.

A mis hermanos Roberto, René y Edwin por brindarme su apoyo en todo momento.

Con mucha ternura a todos mis sobrinos: Itayetzi, Johan, Arturo, Laila, Gacel, Shadai y Kevin, porque deseo que cada día aprendan a superarse y salir adelante.

A la memoria de mi abuelita “maicha”, porque su recuerdo, historias y enseñanzas ocupan por siempre un lugar único en mi corazón.

A mi amiga Lilian y a mi ahijada Natalia, porque hemos formado una amistad muy bonita, llena de confianza y cariño y siempre me hacen sentir en familia.

A mis amigos de toda la vida Rubí y Paz porque a pesar de la distancia y el tiempo siempre los recuerdo con mucho cariño y alegría.

Y muy especialmente dedico mi tesis a mi novio, amigo
compañero y cómplice ***“Miguel”*** porque junto a él
simplemente la vida es muy bonita. Eres mi complemento
perfecto, te amo.

Agradecimientos

Agradezco al Consejo Nacional de Ciencia y Tecnología (CONACyT) por el apoyo económico brindado durante estos dos años de estudio.

Al Centro Nacional de Investigación y Desarrollo Tecnológico (CENIDET) y personal que ahí labora, por permitirme ser parte de esta institución y recibir mi formación de posgrado.

De manera muy especial a mi director de tesis el Dr. René Santaolaya Salgado, por el invaluable apoyo que siempre me ha proporcionado. También quiero agradecerle por su amistad y consejos que siempre me ha ofrecido.

A mi co-director de tesis y a mis revisores, el Dr. Guillermo Rodríguez, el M.C. Mario Guillén, el Dr. Jaime Muñoz, y muy especialmente a la M.C. Olivia Fragoso porque además siempre me ha escuchado y ha representado un gran apoyo en estos dos años de estancia en el CENIDET.

A mis amigos de Ing. de Software Alejandro y Fernando, me alegra haberlos conocido y espero que nuestra amistad perdure.

A todos mis compañeros de generación que compartieron conmigo la aventura de estudiar la maestría.

Resumen

A pesar del surgimiento de nuevos lenguajes y paradigmas de programación, COBOL sigue siendo uno de los lenguajes más usados en la implementación de sistemas financieros. Algunas de las desventajas de los programas hechos en COBOL son que resulta complicado llevar a cabo su mantenimiento y extensión, ya que es un lenguaje de programación estructurado.

Como antecedente a esta tesis hubo otros dos trabajos de tesis los cuales se enfocaron en la traducción de programas hechos en COBOL a un lenguaje de programación orientado a objetos, en este caso Java para los estatutos lógicos y a instrucciones SQL para el sistema de archivos. Todo ello con la finalidad de tener la misma funcionalidad que en los programas hechos en COBOL pero con las ventajas que la programación orientada a objetos y las bases de datos relacionales ofrecen.

Sin embargo, las herramientas de los trabajos de antecedente tienen algunas limitantes. Ambas herramientas funcionan de manera independiente y por lo tanto no es posible tener un solo programa con ambas funcionalidades (estatutos lógicos y sistema de archivos). Por otro lado el número de estatutos que abarcan las aplicaciones es mínimo en el trabajo de traducción de estatutos lógicos y no son considerados todos los tipos de archivos en el trabajo de traducción del sistema de archivos.

En este trabajo de tesis se planteó tener en una sola aplicación la traducción de estatutos de COBOL a Java y la traducción del sistema de archivos de COBOL a una base de datos relacional, que incorpore las dos herramientas de antecedente. De esta manera se generó una sola herramienta implementada como servicio web que lleva a cabo ambas funcionalidades. Así mismo se amplió el número de estatutos traducidos y se consideraron los diferentes tipos de archivos.

Para el desarrollo del servicio web que implementa la herramienta se llevó a cabo el proceso de Ingeniería de software. Desde la especificación de requerimientos, análisis, diseño, implementación, hasta el diseño y ejecución de un plan de pruebas que validara la herramienta.

En este documento se especifica cada uno de los detalles de desarrollo de este trabajo de tesis finalizando con las conclusiones generadas, aportaciones y trabajos futuros.

Abstract

Even with the appearing of new languages and software development paradigms, COBOL remains on one of the most used programming languages in the implementation of financial systems. COBOL is a structured programming language, because of that the COBOL developed software has some disadvantages, like its maintenance complexity and improvement.

Former to this investigation were another two in which the scope was the translation from COBOL developed software to an object oriented language, in this case, to Java for the logical statements and to SQL for the files system. All of these aim to maintain the same functionality of the COBOL developed software, but also with the advantages offered by the object oriented programming and the relational data bases.

However, the software developed in the antecedent investigations has some limitations. Both of them are stand alone software, and because of that it is not possible to have them integrated on an only software with both functionalities (logical statements and files system). In the other hand, the number of logical statements that the software is capable to translate is minimum and the total of file types in the file system translation procedure are not considered.

The purpose of this thesis work is to have only one software with the both functionalities: the translation of the COBOL logical statements to Java and the translation of the COBOL file system to a relational data base, and incorporates both of the antecedent software applications. In this way, only one software was implemented as a Web Service that is capable to make both functionalities. So it, the translation number of logical statements was augmented and different file types were considered.

For the Web Service development that the software implements, the software engineering process was executed. From the requirement specification, analysis, design and implementation to the test plan design and execution that will validate the software.

Each one of the development details of this thesis work is specified on this document, finishing with the generated conclusions, contributions and future work.

Contenido

Contenido.....	i
Introducción.....	1
Capítulo 1. Antecedentes	5
1.1 Antecedentes	5
1.1.1 Cob2SQL [13].....	6
1.1.2 Transformación de Código Cobol a Java [8].....	6
1.2 Problema a Resolver en esta Tesis.....	6
1.3 Objetivos	7
1.3.1 Objetivo General	7
1.3.2 Objetivos Específicos.....	7
1.3.3 Objetivos de la Herramienta Generada.....	7
1.4 Trabajos Relacionados	8
1.5 Producto - Resultado	13
1.6 Beneficios.....	13
1.7 Alcances.....	13
1.8 Limitaciones	14
Capítulo 2. Marco Teórico	15
2.1 Compiladores	15
2.2 JavaCC	16
2.3 Gramáticas Libre de Contexto.....	17
2.4 Reingeniería de Software.....	18
2.5 Características del Software	18
2.6 Versiones Estándares de COBOL.....	18
2.7 Java	19
2.8 SQL versión 92.....	19
2.9 Servlets	19
2.10 Servicio Web	20
2.11 Patrones de Diseño	21
2.12 Patrón de Diseño “Strategy”.....	22
Capítulo 3. Análisis del Problema y Solución	25
3.1 Variables en COBOL	26
3.2 Tipos de datos.....	26
3.3 Variables indivisibles	27
3.4 Registros	27
3.5 Calificación de Variables.....	29
3.6 Variables de Condición	30
3.7 Estatutos Lógicos	31
3.7.1 Correspondencia entre párrafos (COBOL) y métodos (Java)	32
3.7.2 El método “main” del programa en Java	32
3.7.3 La Instrucción GO TO	33
3.7.4 Traducción General de Verbos (Estatutos Lógicos)	35
3.8 Estatutos de Acceso a Archivos.....	36
3.8.1 Organización del Archivo.....	37
3.8.2 Tipos de Acceso al Archivo	38
3.8.3 Traducción del Verbo: Open.....	39
3.8.4 Traducción del Verbo: Close	40
3.8.5 Traducción del Verbo: Delete	41
3.8.6 Traducción del Verbo: Write	42

3.8.7	<i>Traducción del Verbo: Rewrite</i>	43
3.8.8	<i>Traducción del Verbo: Read</i>	43
3.8.9	<i>Códigos de Información para Instrucciones de Acceso a Archivos</i>	44
3.9	Manejo de verbos no definidos en el estándar COBOL 85.....	45
3.10	Arquitectura propuesta.....	45
Capítulo 4.	Análisis, Diseño e Implementación	49
4.1	Análisis y Diseño del Servicio Web	51
4.1.1	<i>Requerimientos Funcionales: Modelo de Casos de Uso</i>	51
4.1.2	<i>Requerimientos de Diseño y Construcción</i>	55
4.1.3	<i>Requerimientos de Reusabilidad</i>	55
4.1.4	<i>Diagramas de Paquetes y Clases</i>	55
4.1.5	<i>Diagrama de Despliegue</i>	62
4.1.6	<i>Interfaz del Servicio Web</i>	63
4.2	Análisis y Diseño del Cliente del Servicio Web.....	64
4.2.1	<i>Requerimientos Funcionales: Modelo de Casos de Uso</i>	64
4.2.2	<i>Diagrama de Paquetes y Clases</i>	66
4.2.3	<i>Diagrama de Despliegue</i>	68
4.2.4	<i>Interfaz del Cliente Web</i>	68
Capítulo 5.	Diseño Experimental	73
5.1	Plan de Pruebas	73
5.1.1	<i>Elementos de Prueba</i>	73
5.1.2	<i>Criterios de Aceptación</i>	75
5.1.3	<i>Criterios de Suspensión</i>	75
5.1.4	<i>Actividades de Prueba</i>	75
5.1.5	<i>Condiciones de Ambiente</i>	76
5.1.6	<i>Personal para Ejecución de Pruebas</i>	77
5.1.7	<i>Riesgos y contingencias</i>	77
5.2	Diseño de las pruebas	77
5.2.1	<i>Pruebas de Integración</i>	78
5.2.2	<i>Pruebas de Función</i>	81
5.3	Resultados de la Ejecución del Plan de Pruebas.....	88
5.3.1	<i>Pruebas de Integración</i>	88
5.3.2	<i>Pruebas de Función</i>	89
5.4	Análisis de los Resultados	110
Capítulo 6.	Conclusiones	111
6.1	Conclusiones.....	111
6.2	Aportaciones	113
6.3	Trabajo Futuro.....	113
Anexo A.	Capacidades del Sistema	115
Anexo B.	Detalle de los de Casos de Uso de la Herramienta	125
Anexo C.	Descripción de Clases por Paquete	143
Referencias	149

Introducción

COBOL (Common Organization Business Oriented Language), fue desarrollado durante los años 60's y fue uno de los lenguajes de programación con más auge por muchos años por su enfoque a procesamiento de datos de negocios, en [18] se refiere que en 1990 había un estimado de 120 mil millones de líneas de código escritas en COBOL. A pesar de la evolución tecnológica y nuevos paradigmas de programación, de acuerdo a las encuestas realizadas por Gartner en el 2006 se siguen usando sistemas heredados hechos en COBOL.

Con el paso del tiempo, mantener los sistemas hechos en COBOL, se vuelve costoso y riesgoso, ya que cada vez menos personas conocen el lenguaje de programación, muchos años de mantenimiento dañan la estructura del sistema y extenderlo a nuevas funcionalidades se vuelve una tarea complicada y cabe la posibilidad de insertar defectos en el código.

En [18] se menciona que existen varias estrategias para cambiar el software como son:

- **Mantenimiento de Software.** Esta estrategia propone dar respuesta a una serie de cambios en los requerimientos, manteniendo la arquitectura original del sistema.
- **Transformación Arquitectónica.** Conforme se realizan solicitudes de cambios, la arquitectura va siendo modificada. Es una estrategia más radical que el mantenimiento del software.
- **Reingeniería de Software.** El sistema se modifica de tal forma que sea más fácil de comprender y cambiar. La reingeniería implica una traducción del código fuente, aplicar ingeniería inversa para documentar el sistema, mejora de la estructura del programa, modularización del programa y aplicar reingeniería a los datos.

El presente documento de tesis muestra la investigación e implementación de una herramienta que realiza una parte del proceso de Reingeniería de Software (§2.4) a sistemas hechos en COBOL enfocándose en la traducción hacia el lenguaje de programación Java.

A continuación se describe la organización del presente documento de tesis.

El capítulo 1 ofrece un panorama general del trabajo de tesis. En la sección 1.1, se presentan los trabajos de antecedente que fueron contemplados para el desarrollo de la tesis. En las secciones 1.2 y 1.3 se plantean tanto el problema como los objetivos respectivamente. Posteriormente, en la sección 1.4 se enlistan los trabajos que buscan resolver problemas similares al planteado por esta tesis. En las secciones 1.5 y 1.6 se presentan el Producto – Resultado y los beneficios que se obtendrán con dicho producto. Finalmente las secciones 1.6 y 1.7 muestran los alcances y limitaciones de esta investigación.

El capítulo 2 ofrece el marco teórico que respalda la tesis. Se comienza con la sección 2.1 y la teoría general de compiladores, posteriormente, la sección 2.2 habla de JavaCC una herramienta en la cuál se basó el producto final de la presente tesis y en la sección 2.3 se presenta una idea general de las gramáticas libres de contexto. En la sección 2.4 se describe de manera general la reingeniería de software. La sección 2.5 habla de las características deseables para todo software. En las secciones 2.6, 2.7 y 2.8 se presentan definiciones y características de los lenguajes de programación con los que se trabajaron en esta investigación como son: COBOL, Java y SQL. Las secciones 2.9 y 2.10 ofrecen una breve descripción de los tipos de aplicaciones implementadas en este trabajo de tesis y obtenidas como producto resultado. Finalmente las secciones 2.11 y 2.12 presentan definiciones y características de los patrones de diseño en general y el patrón de diseño “**Strategy**” en particular. Para considerarlo parte fundamental de este trabajo de tesis.

En el capítulo 3 se presenta el análisis realizado al lenguaje COBOL para llevar a cabo la traducción al lenguaje Java. Las secciones 3.1 y 3.2 hablan de las variables y tipos de datos respectivamente. En las secciones 3.3, 3.4 y 3.5 se explica el funcionamiento de las variables indivisibles, los registros y como se lleva a cabo su traducción. En la sección 3.6 se explican las variables de condición, un tipo de variable particular en COBOL. En la sección 3.7 se presenta el análisis hecho a los estatutos lógicos y los verbos en general para llevar a cabo su traducción correspondiente. En la sección 3.9 se describen los estatutos de acceso a archivos y los puntos más importantes tomados en cuenta para su traducción. La sección 3.9 habla de la estrategia que se implementó para los verbos no definidos en el estándar COBOL 85. Finalmente la sección 3.10 presenta una propuesta de la arquitectura.

El capítulo 4 está dividido en dos grandes secciones: la sección 4.1, la cual detalla el análisis, diseño e implementación para el servicio Web encargado de llevar a cabo la traducción de lenguaje COBOL a lenguaje Java y la sección 4.2, que detalla el análisis, diseño e implementación de Cliente del Servicio Web y que corresponde a la interfaz del usuario final.

En el capítulo 5 se presentan las pruebas realizadas a las dos herramientas generadas por este proyecto de tesis. En la sección 5.1 se presenta el plan de pruebas, la sección 5.2 muestra el detalle de las pruebas, en la sección 5.3 se presentan los resultados de la ejecución del plan de pruebas. Finalmente, en la sección 5.4 se lleva a cabo un análisis de los resultados obtenidos.

El capítulo 6 cierra con las conclusiones obtenidas del desarrollo de esta investigación. La sección 6.1 describe las conclusiones generales, la sección 6.2 las aportaciones hechas por el presente trabajo de tesis y en la sección 6.3 se enlistan una serie de propuestas de trabajo futuro.

Capítulo 1. Antecedentes

En este capítulo se abordan los temas generales que fueron tomados en cuenta para el desarrollo de la tesis.

1.1 Antecedentes

En esta sección se describen dos herramientas que son tomadas como base para el desarrollo de la herramienta capaz de traducir código escrito en lenguaje COBOL a código en lenguaje Java. Ambas herramientas son el producto final de dos trabajos de investigación desarrolladas en CENIDET como tesis de maestría. La primera de ellas, se enfoca a la traducción de instrucciones de acceso de datos en archivos indexados y la otra a la traducción de instrucciones de la lógica de la aplicación y el control a Java. Actualmente, ambas herramientas trabajan de forma independiente.

1.1.1 Cob2SQL [13]

El autor de la tesis denominada: “Estudio para la transformación gramatical de instrucciones de acceso a una base de datos escritas en lenguaje COBOL hacia instrucciones escritas en lenguaje SQL”, toma como base el proyecto “SR2: Reingeniería de software Legado para Reuso” descrito en [15]. En esta tesis se construyó una herramienta, que utiliza un analizador léxico y sintáctico del código de entrada COBOL 85 para recuperar la información necesaria y para la conversión de sentencias de manipulación de datos escritas en este lenguaje hacia lenguaje SQL (Estándar 92), aplicando un conjunto de reglas sintácticas y semánticas. La herramienta desarrollada es denominada COB2SQL. La conversión se puede ejecutar en cualquier manejador de base de datos relacional que soporte el estándar SQL 92.

Un punto importante a considerar, es que el programa COB2SQL se desarrolló utilizando el modelo de la programación orientada a objetos, realizando un diseño arquitectural de clases que permite la extensión de nuevas funcionalidades.

1.1.2 Transformación de Código Cobol a Java [8]

El objetivo de este trabajo de tesis denominado: “Estudio empírico para la transformación gramatical de código en lenguaje COBOL hacia lenguaje JAVA”, es transformar el código de la lógica del negocio y del control escrito en lenguaje COBOL a código escrito en lenguaje JAVA basándose en equivalencias gramaticales, así como en métodos heurísticos de reestructura de la arquitectura, para facilitar su reutilización, mantenimiento y futuras modificaciones para ser adaptado a nuevos requerimientos.

La investigación implementa el modelo de reingeniería del proyecto SR2 para transformar código legado escrito en lenguaje COBOL (procedimental), hacia el lenguaje JAVA (orientado a objetos); logrando con ello ampliar el tiempo de vida útil de aplicaciones que fueron desarrolladas en COBOL. El producto final de la tesis se denomina C2J.

1.2 Problema a Resolver en esta Tesis

Se sabe que existen muchas líneas de código escritas en lenguaje COBOL en empresas Mexicanas importantes, tales como los corporativos de Banamex y Bancomer, que tienen problemas de mantenimiento y de adaptación a nuevos usos y/o requerimientos. Por lo que existe la necesidad de aplicar una estrategia para ampliar el tiempo de vida útil de estos programas. Una de tales estrategias es la transformación de este código a un lenguaje que ofrezca o soporte

facilidades de flexibilidad y robustez (§2.5). Hacer esta re-estructura de forma manual, resulta costoso en tiempo y en dinero y hay poca garantía que un sistema re-estructurado cumpla con la misma funcionalidad del código original y que esté libre de defectos.

Los trabajos de [13] y [8], desarrollados en CENIDET, están encaminados a migrar automáticamente código COBOL hacia código en Java bajo el paradigma Orientado a Objetos, sin embargo, cada una de las herramientas trabaja de manera independiente. Es necesaria la integración de ellas para ofrecerlas al mercado en una sola herramienta que realice la migración de forma automática.

Ambos trabajos implementan el estándar COBOL 85, sin embargo, en aplicaciones comerciales existen diversos dialectos, por lo cual se requiere una arquitectura capaz de soportar extensiones para implementar versiones No-Estándar.

1.3 Objetivos

1.3.1 Objetivo General

Generar una herramienta que permita la traducción automática o semi-automática de código legado escrito en lenguaje COBOL hacia código Java.

1.3.2 Objetivos Específicos

- Integrar las herramientas de los trabajos de [13] y [8] con el propósito de generar una herramienta única que permita la traducción de código legado escrito en lenguaje COBOL hacia código JAVA, por medio del desarrollo de un sistema de reingeniería, basado en estos trabajos de antecedente.
- Complementar la gramática usada en los trabajos de [13] y [8], para alcanzar mayor cobertura de verbos o instrucciones de COBOL en la traducción.
- Si la industria nacional proporciona la información, agregar a la gramática instrucciones de un dialecto propietario (Versión No – Estándar).

1.3.3 Objetivos de la Herramienta Generada

- Traducir programas escritos en el lenguaje de programación COBOL a lenguaje de programación Java, con el propósito de ampliar la vida útil de sistemas de software de tamaño considerable y de uso comercial.

- Re-estructurar programas que han sido actualizados muchas veces y por lo cual se vuelven inestables, con el propósito de disminuir su costo por mantenimiento.
- Generar aplicaciones de software flexibles (§2.5), en lenguaje de programación Java, capaces de soportar modificaciones y/o extensiones ante cambios de requerimientos o nuevos requerimientos, sin tener que modificar un porcentaje significativo del código ya existente.

1.4 Trabajos Relacionados

Debido a que la presente tesis es un trabajo de desarrollo tecnológico, se buscaron herramientas que ofrecieran la transformación de COBOL a un lenguaje orientado a objetos, similar a lo que se implementó en el presente trabajo de tesis. A continuación se enlistan y describen cada una de ellas:

1.- **COREL: COBOL Transformation Toolkit [20]**

Desarrollado por la empresa **SoftwareMining**, fue diseñado para permitir a las organizaciones migrar y eventualmente eliminar de sus sistemas las restricciones y costos inherentes asociados con las aplicaciones COBOL. Realiza la transformación a lenguajes como Java (J2EE), C#/ASP y ANSI/MF COBOL. El proceso de migración/traducción puede ser llevado a cabo de dos formas:

- 1.- Traducción Basada en Heurísticas. Un traductor basado en Inteligencia Artificial aplica un conjunto de reglas de simplificación.
- 2.- Aplicación para la extracción de reglas de negocios además de la heurística basada en la traducción para configuraciones más detalladas.

Aproximadamente, el 98% del código legado es traducido automáticamente. Se realiza la traducción sobre algunos dialectos, los cuales no son especificados. Durante la fase de análisis, el sistema construye modelos internos para las estructuras de datos y el programa lógico. El analizador usa heurísticas basadas en algoritmos para identificar los datos usados, identificar y remover código muerto, simplificar "REDEFINITIONS", reestructurarlo dentro de código procedimental, etc.

También cuenta con un módulo de documentación, el cuál opera como una extensión del analizador, generando documentos y reportes. El sistema usa diagramas de actividad UML para representar el flujo lógico de los procesos y genera la documentación en código HTML.

Por otra parte, la herramienta convierte las estructuras de datos de COBOL a SQL (Oracle, MSSQL, MySQL, DB2 o ANSI).

2.- Language Migration for COBOL [19]

Desarrollado por la empresa **BLUEPHOENIX**, promete realizar la traducción de COBOL a JAVA y con ello permitir llevar a cabo actualizaciones de forma más sencilla por el equipo de desarrollo.

Las especificaciones técnicas son:

- Permite la traducción de COBOL CICS, batch e IMS.
- Permite la migración a plataformas como Windows, Unix, Linux y Mainframe.
- Los lenguajes a los que se puede migrar son: Java/J2EE 1.5, C#/.NET, y Micro Focus COBOL.
- Manejadores de Bases de Datos permitidos: SQL Server, DB2 y Oracle.

3.- Fujitsu Data Converter [24]

La herramienta “Data Converter” fue desarrollada por la empresa **Alchemi Solutions**. Convierte archivos de datos de un formato a otro. Convierte desde diferentes tipos de archivos como: secuenciales, relativos, indexados, archivos de texto, binarios, datos XML y archivos con datos separados por comas. Sus principales características son:

- Mueve datos entre diferentes formatos de archivos de COBOL
- Carga archivos desde archivos de texto
- Convierte datos hacia y desde archivos de texto
- Crea archivos DTD (Document Type Definition) usados con XML
- Convierte datos desde un código y lo coloca en otro (ASCII a EBCDIC)
- Mueve datos de COBOL hacia y desde hojas de cálculo y base de datos.

La empresa, anteriormente ofrecía una herramienta llamada “Dialect Converter”, la cual realizaba la transformación de los dialectos de COBOL a Java y era el complemento de “Data Converter”, ahora esta descontinuada, en la página de la empresa no se explica la razón.

4.- Micro Focus Mainframe Express Enterprise Edition [22]

Jazillian es un servicio de traducción de COBOL a Java, la traducción se realiza a código bajo el estándar de COBOL85. En la página Web donde se ofrece el servicio indica que en el peor de los casos se puede traducir del 90% a 95%, mientras que en el mejor de los casos se realizará del 98 al 100% de la traducción.

Para llevar a cabo la traducción se requiere el trabajo colaborativo entre la empresa que requiera la conversión y Jazillian, para la generación de reglas específicas. La herramienta está limitada a realizar la traducción de sentencias de COBOL a Java, no se maneja ningún tipo de conversión del acceso a datos.

5.- COB2J [23]

Micro-Processor Services, Inc., es la empresa que desarrolló una familia de herramientas de traductores llamada COB2J, soporta tipos de COBOL como: COBOL 68, COBOL 74, COBOL 1985 (ANSI x3.23-1985, ISO 1989-1985), RM/ COBOL, MF/ COBOL (Micro Focus), IBM COBOL, COBOL II, AS400 COBOL, ACUCOBOL – 85, ACUCOBOL – GT, Realia COBOL (COMPUTER ASSOCIATES).

La herramienta realiza la traducción bajo un método de doble conversión, de COBOL hacia un tercer lenguaje (no se especifica explícitamente) y de este a Java o J#. Puede traducir hasta 40,000 líneas por archivo para versiones WIN16 y hasta 500,000 líneas por archivo para WIN32.

6.- Legacy eTrans-Formation™ Service (LeTSSM)[21]

Servicio ofrecido por **Comptramatics Corporation**, quien promete en cuestión de semanas realizar la transformación de programas escritos en COBOL a Java. El costo es determinado por el número de líneas de código. El código entregado mantiene los comentarios hechos en los programas de COBOL. La empresa no proporciona información en cuanto al método de transformación.

7.- Caravel: Automatic translation of RPG and COBOL applications to Java, reengineering tool [25]

TransTOOLS es un conjunto de herramientas y servicios para traducir y aplicar re-ingeniería a sistemas desarrollados bajo lenguajes propietarios como COBOL y RPG a Java.

Caravel (herramienta de TransTOOLS) traduce el 100% de los sistemas COBOL a Java, de acuerdo con la página oficial de la empresa **TransTOOLS, S.A.**, ubicada en España. Las ventajas del estándar de Java incluye: interfaces gráficas, programación orientada a objetos, multiplataforma, aplicaciones Web, entre otras.

Los sistemas son traducidos mientras conservan la misma funcionalidad de las aplicaciones originales, y sin invertir tiempo en la capacitación para modificar los procedimientos hechos en COBOL.

El acceso a base de datos se realiza por medio del estándar JDBC, y las aplicaciones pueden tener acceso a diversos manejadores de base de datos.

La Tabla 1 muestra una comparación de las herramientas similares implementada en este trabajo de tesis, las principales características que fueron consideradas son:

- **Cobertura de los estatutos.** Indica qué estándares o dialectos son traducidos por la herramienta.
- **Nivel de automatización.** Se refiere al nivel de intervención de los usuarios al momento de llevar a cabo la traducción. Si es automático significa que los usuarios no intervendrán durante la transformación, si es semi-automático, los usuarios deben ir configurando la traducción durante el proceso.
- **Control de la Evolución (por extensión a otros dialectos).** Significa que, los usuarios que adquieren la herramienta pueden tener la facilidad de extender la herramienta para la traducción de otros dialectos o si el código es cerrado a extensiones.
- **Costo.** Es el costo del servicio o el costo por la herramienta que lleve a cabo la traducción.
- **Confidencialidad del código fuente.** Si la traducción la llevan a cabo personas con el requerimiento de la traducción, indica que pueden ver el código fuente en COBOL que será traducido y por lo tanto no existe confidencialidad. Por el contrario, si el código solamente es usado por un sistema de software sin involucrar personas, significa que si existe confidencialidad del código.
- **Arquitectura – Programa Resultado.** Indica cómo es la estructura del programa final traducido de COBOL a Java.

Tabla 1. Tabla comparativa de los trabajos relacionados

Identificador de la Herramienta	Cobertura	Lenguaje Destino	Unión Datos/Modelo	Nivel de Automatización	Costo	Confidencialidad	Arquitectura Programa Resultado
1	No especifica	Java (J2EE), C#/ASP y ANSI/MF COBOL.	No	Automático	No especifica	Si	MVC
2	COBOL CICS, Batch e IMS	Java/J2EE 1.5, C#/.NET, y Micro Focus COBOL. SQL Server, DB2 y Oracle.	Si	Automático	De acuerdo a la aplicación	No	No especifica
3	Sólo el sistema de archivos.	XML	No	Semi-automático	No especifica	Si	No especifica
4	COBOL 85(Todas las expresiones gramaticales)	Java, C#/.NET, Micro F. COBOL , SQL Server, DB2, and Oracle	Si	Semi-automático	No especifica	No	Ninguna
5	COBOL 68, 74, 1985 (ANSI x3.23-1985, ISO 1989-1985) RM/ COBOL, MF/ COBOL (Micro Focus) COBOL II, IBM COBOL, AS400 COBOL, ACUCOBOL – 85, ACUCOBOL – GT, Realia COBOL	COBOL -> Tercer Lenguaje Tercer Lenguaje -> Java o J# SQL, DB2	Si	Automático	Por licencia	Si	No especifica
6	No especifica, adapta la herramienta al estándar indicado.	Java	NE	Automático	Por línea de código traducida.	No. Se requiere enviar la aplicación.	Se ofrece como un servicio independiente
7	COBOL/400, CICS, IMS, COBOL, COBOL II.	Java JDBC para base de datos.	Si	Automática o Personalizada	Depende de cada proyecto.	No. Se requiere enviar la aplicación.	Arquitectura MVC
TESIS	COBOL 85. Extensible a otros dialectos.	Java MySQL para la base de datos	Si	Automático	Por líneas de código traducido.	Si. Servicio Web.	Arquitectura MVC y patrones de diseño

1.5 Producto - Resultado

Como Producto – Resultado de esta tesis, se obtuvo un servicio Web para la traducción automática de código escrito en Lenguaje COBOL hacia código en Lenguaje Java, abarcando la traducción de estatutos lógicos y estatutos de acceso a archivo, además de generar un modelo de datos del sistema de archivos.

1.6 Beneficios

Los beneficios esperados para los sistemas traducidos son los siguientes:

- Ampliar la vida útil de sistemas de software de tamaño considerable y de uso comercial
- Disminuir los costos por mantenimiento de sistemas de software que han sido modificados y/o actualizados muchas veces, por lo cual se vuelven inestables y su confiabilidad es disminuida.
- Generar aplicaciones de software flexibles (§2.5), en lenguaje de programación Java, capaces de soportar extensiones.

1.7 Alcances

A continuación se enlistan de manera general los alcances del trabajo esta tesis. En el Anexo A del presente documento se detallan cada una de las capacidades del sistema generado como producto-resultado.

- Se complementó la gramática (Estándar COBOL 85) usada en las herramientas de antecedente [8] y [13] con la gramática oficial ofrecida por ANSI / ISO.
- Se llevó a cabo un análisis a fondo de la viabilidad de traducción de estatutos de COBOL a Java.
- Se analizaron los sistemas de antecedente [8] y [13]. Se llevaron a cabo las pruebas correspondientes y se modificaron de forma que se ampliará la cobertura de los estatutos.
- Se Diseñó e implementó una arquitectura de integración que permite el funcionamiento de las herramientas de antecedente proporcionadas por [8] y [13]. La arquitectura es flexible (§2.5) para dar facilidades de extensión de reglas gramaticales que permita la incorporación de traducciones a otros dialectos de COBOL.
- Se llevó a cabo una reestructuración inicial del código traducido, ya que cada una de las estructuras de datos en el código de COBOL son generadas como una clase en el código en Java.

- La herramienta está implementada como un servicio Web, para su uso con otras aplicaciones. Además se implementó un cliente Web para proporcionar una interfaz al usuario que desee llevar a cabo la traducción de sus programas escritos en COBOL.

1.8 Limitaciones

Las limitaciones con que cuenta el sistema obtenido como producto-resultado son las siguientes:

- Basada completamente en la gramática estándar COBOL 85.
- No se abarcó el 100% de los estatutos de la gramática.
- La traducción se realiza a un sólo archivo de código fuente, es decir, la traducción no incluye subprogramas o módulos de programas separados y accedidos desde el programa principal.
- Para los verbos de Acceso a Archivos no se contempló la escritura directamente a la impresora.
- La herramienta solamente fue probada con programas obtenidos de libros y programas de tutoriales, ya que la industria no proporcionó código completo ni suficiente para pruebas.
- La herramienta final no genera ningún tipo de documentación del programa traducido.
- La reestructuración no es la óptima, ya que es la primera aproximación a tener una arquitectura en el código traducido de COBOL.

Capítulo 2. Marco Teórico

2.1 Compiladores

Un compilador es un programa que puede leer otro programa escrito en algún lenguaje de programación (el lenguaje fuente) y traducirlo a otro programa equivalente en otro lenguaje (el lenguaje destino) [1]. Una de las principales funciones del compilador es reportar cualquier error en el programa fuente que detecte durante el proceso de traducción.

El compilador lleva a cabo diferentes fases, las cuales pueden dividirse principalmente en análisis y síntesis. La fase de análisis divide el programa fuente en componentes e impone una estructura gramatical sobre ellos. Además, la fase de análisis recolecta información sobre el programa fuente y la almacena en una estructura de datos. Por su parte, la fase de síntesis, construye el programa destino deseado a partir de la información generada en la fase de análisis. La 0 muestra todas las fases de un compilador.

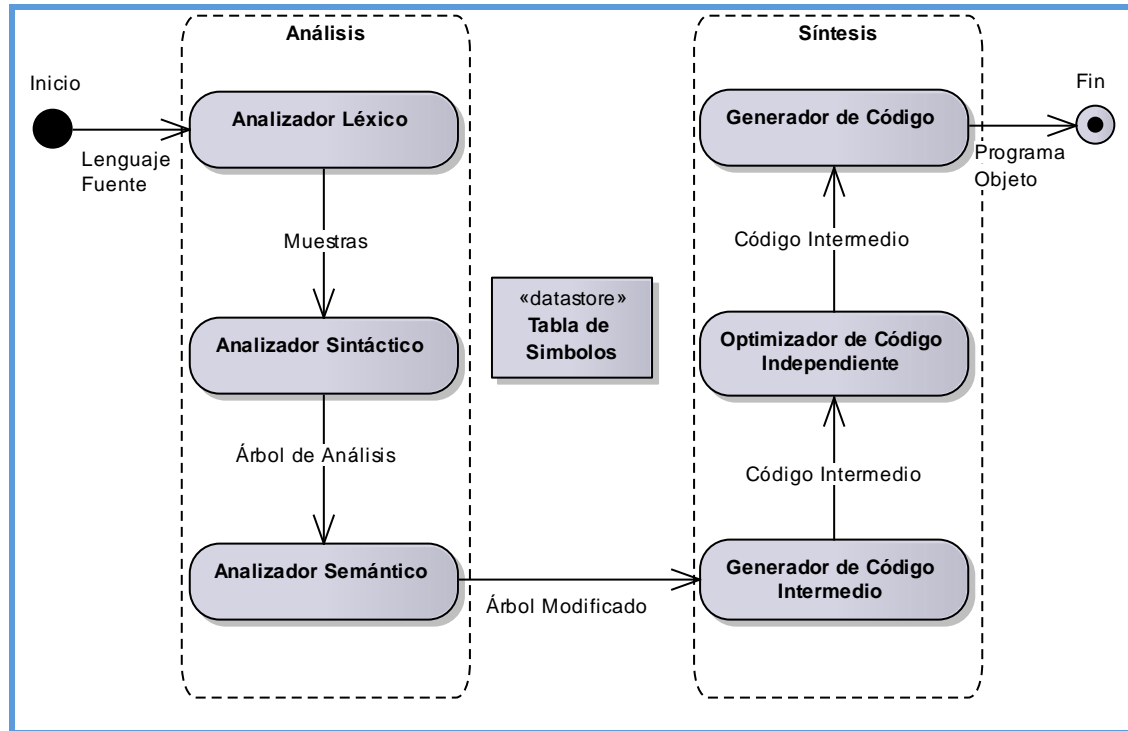


Figura 1. Fases de un compilador

Un traductor es un programa que recibe una entrada en un lenguaje y produce una salida en otro lenguaje. La herramienta generada en este trabajo de tesis es un traductor, en el cual la entrada es un programa escrito en lenguaje de programación COBOL y la salida es un programa en el lenguaje Java y SQL.

Existen herramientas especializadas que ayudan a implementar las diversas fases de un compilador, estas herramientas utilizan lenguajes especializados para especificar e implementar componentes específicos, y utilizan algoritmos sofisticados para llevarlas a cabo. Una de estas herramientas es JavaCC, la cuál fue usada en este trabajo de tesis.

2.2 JavaCC

JavaCC (Java Compiler Compiler – Metacompilador en Java). Es una herramienta que produce de manera automática un analizador sintáctico a partir de una descripción gramatical de un lenguaje en particular. Utiliza el método de funciones recursivas. Los analizadores generados son recursivos descendentes con derivaciones por la izquierda. Sus principales características son [10]:

- Especificaciones léxicas y sintácticas en un solo archivo en notación BNF (forma Backus-Naur).

- Permite agregar fácilmente acciones semánticas.
- Permite especificar si el analizador contemplará distinciones entre mayúsculas y minúsculas.
- Permite llevar a cabo la gestión de errores.
- Lookahead mayor a 1. El analizador sintáctico puede leer más de un token de entrada antes de decidir la regla sintáctica que usará.

La especificación de la gramática del programa fuente debe ser escrita en un archivo con extensión .jj, posteriormente se lleva a cabo la compilación, generando los siguientes archivos:

- **<<nombreParser>>.java**: Es el analizador sintáctico.
- **<<nombreParser>>TokenManager.java**: Es el analizador lexicográfico.
- **<<nombreParser>>Constans.java**: Interfaz que asocia un código a cada token.
- **ParseException.java**: Clase que implementa los errores sintácticos.
- **SimpleCharStream.java**: Clase que implementa los canales de entrada de los cuales puede leer el analizador léxico.
- **Token.java**: Clase que implementa un objeto a través del cuál se comunican el analizador léxico y el sintáctico.
- **TokenMgrError.java**: Clase que implementa el administrador de errores lexicográficos.

2.3 Gramáticas Libre de Contexto

Una gramática describe en forma natural la estructura jerárquica de la mayoría de las instrucciones de los lenguajes de programación. Una gramática libre de contexto tiene cuatro componentes [1]:

- Un conjunto de símbolos terminales, a los que se les conoce como tokens. Los cuales son símbolos elementales del lenguaje definido por la gramática.
- Un conjunto de no terminales, a las que algunas veces se les conoce como “variables sintácticas”. Cada no terminal representa un conjunto de cadenas o terminales.
- Un conjunto de producciones, en donde cada producción consiste en un no terminal, llamada encabezado o lado izquierdo de la producción, una flecha y una secuencia de terminales y no terminales llamada cuerpo o lado derecho de la producción.
- La designación de uno de los símbolos no terminales como el símbolo inicial.

2.4 Reingeniería de Software

Se refiere a re-implementar sistemas heredados para hacerlos más mantenibles, comprende la re-documentación, la organización y re-estructura del sistema, la traducción del sistema a un lenguaje de programación más moderno y la modificación y actualización de la estructura y los valores de los datos del sistema [18].

2.5 Características del Software

De acuerdo a [14] existen aspectos de calidad deseables en todo sistema de software, algunos de ellos son definidos a continuación.

Robustez

Es la habilidad de los sistemas de software para continuar funcionando en condiciones anormales.

Reusabilidad

Es la habilidad de los productos de software para ser reusados, completos o por partes para nuevas aplicaciones.

Extensibilidad

Es la facilidad con las cuales los productos de software podrían ser adaptados a cambios en las especificaciones.

Flexibilidad

Es la facilidad con las cuales los productos de software permiten hacer cambios, sin afectar otras partes del sistema.

2.6 Versiones Estándares de COBOL

Desde el nacimiento de COBOL en 1959, han existido algunos estándares del lenguaje que define su estructura léxica y sintáctica. En 1968 se aprobó una versión estándar del lenguaje por la ahora ANSI (American National Standards Institute) y una versión revisada en 1974 [2]. Posteriormente en 1985 surgió la versión de COBOL 85 con la cual se trabaja en el presente proyecto. Finalmente en el año 2002 surgió una versión revisada que incorpora instrucciones para interactuar con bases de datos relacionales, con manejadores de Base de Datos como DB2 y MySQL.

Las versiones no estándares son todas aquellas que extienden a las versiones definidas como estándar. Esto significa que además de contener las instrucciones definidas en el estándar,

implementan nuevas instrucciones de acuerdo a las necesidades que requiera la organización que use el lenguaje de programación en COBOL no estándar.

En este trabajo de tesis se implantó un traductor de COBOL 85 a Java y SQL, desarrollando una arquitectura que fuera fácil de extender a otras versiones de COBOL.

2.7 Java

“**Java:** un lenguaje simple, orientado a objetos, distribuido, intérprete, robusto, seguro, de arquitectura neutra, portátil, de alto desempeño, de hilos múltiples y dinámico” [7].

Aunque esta definición fue dada por sus creadores de SUN, y son alabanzas al lenguaje, Java si cumple con la mayoría de estas características. Debido a que es uno de los lenguajes de programación con más aceptación y por sus características, fue elegido como uno de los lenguajes destinos del traductor implementado en el presente trabajo de tesis.

2.8 SQL versión 92

SQL (Structured Query Language) es el lenguaje estándar para las bases de datos relacionales.

En 1986, ANSI (American National Estándar Institute) e ISO (International Standards Organizations) publicaron una norma SQL-86. En 1989 ANSI publicó una extensión a la norma SQL:86 denominada SQL:89. La siguiente versión de la norma fue SQL:92 seguida de SQL:1999, Finalmente la versión más reciente es la versión SQL:2003 [17].

El presente trabajo de tesis toma como base la versión SQL:92 para generar tanto la descripción como las instrucciones a la base de datos, ya que el trabajo que fue tomado como antecedente también toma este estándar.

2.9 Servlets

Un servlet es un componente o programa que genera contenido Web dinámico [6]. Los servlets se definen por medio del API Java Servlet y se gestiona por medio de un servidor o contenedor como Tomcat. Los servlets se ejecutan dentro de un servidor compatible con Java (contenedor Web), como lo muestra la Figura 2.

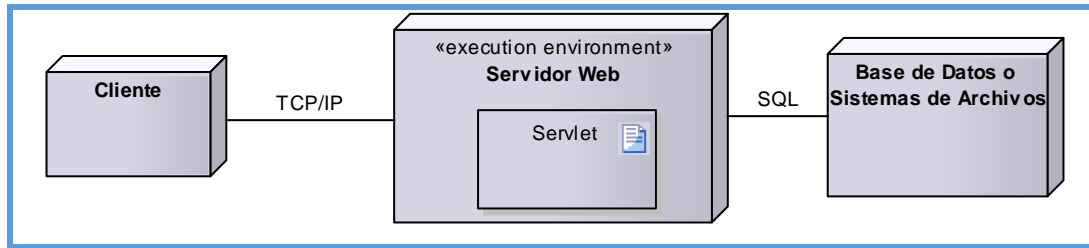


Figura 2. Arquitectura de una aplicación web usando servlets

2.10 Servicio Web

Los servicios Web son aplicaciones modulares auto descriptivas que se pueden publicar, ubicar e invocar desde cualquier punto de la Web o desde el interior de una red local basada en estándares abiertos de Internet [4]. Los servicios Web combinan programación con componentes y programación Web, generando módulos que pueden volverse a utilizar sin preocuparse por la implementación, lenguaje, sistema operativo o modelo de componentes usado. El acceso a los servicios Web se realiza a través de protocolos de Internet omnipresentes como HTTP o SMTP basados en XML.

El W3C (World Wide Web Consortium) define a un servicio Web como:

“... un sistema de software diseñado para soportar la interacción entre dos máquinas a través de una red. Posee una interfaz descrita en un formato WSDL que puede ser procesado por una máquina. Otros sistemas pueden interactuar con el servicio Web en la manera prescrita por su descripción utilizando mensajes SOAP, típicamente transportados utilizando HTTP y serializados con XML, en conjunción con otros estándares relacionados con la Web ” [26].

Un ejemplo de un escenario típico de los servicios Web es tener uno o más proveedores de servicios, los cuales llevan a cabo la implementación de los servicios. Los proveedores definen la descripción de los diferentes servicios y los publican en un registro. Entonces un cliente solicita al registro encontrar un servicio de acuerdo a sus necesidades. El registro busca y envía al cliente la descripción del servicio solicitado. Una vez que el cliente tiene la descripción del servicio solicita el servicio al proveedor correspondiente. Véase la Figura 3.

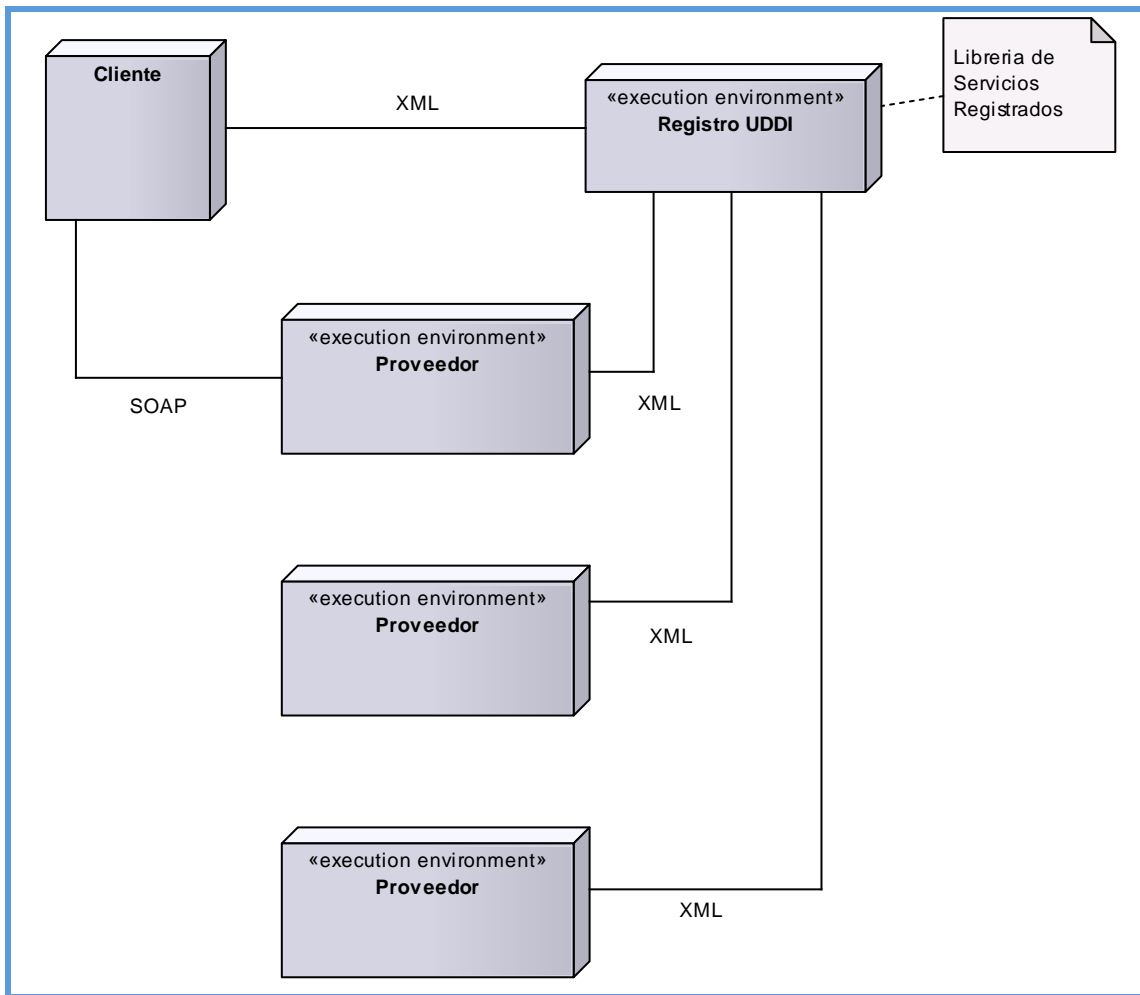


Figura 3. Arquitectura orientada a servicios

2.11 Patrones de Diseño

En general, un patrón describe la manera común en que un grupo de personas realiza o hace cosas [9]. Los patrones son conjuntados por un grupo de personas de acuerdo a los temas que se repiten (pueden ser diseños) y son descritos de tal forma que otros puedan leer y sepan como aplicarlos.

Los patrones de diseño se derivaron de Christopher Alexander, quién en la construcción de edificios sugirió que existían ciertos patrones de diseño que eran comunes, interesantes y efectivos [18].

Los patrones aplicados a los sistemas buscan describir o guiar una solución a un problema determinado de tal forma que pueda ser reutilizado en diferentes casos. Las primeras aplicaciones

de este enfoque de reutilización fueron los algoritmos, después, la documentación de tipos abstractos como pilas, árboles y listas.

En los patrones de diseño no se dan detalles de la implementación, se puede ver cómo una descripción del conocimiento y experiencia [18], pueden ser usados durante la etapa de análisis y/o el proceso de diseño.

Los patrones de diseño son asociados al diseño orientado a objetos donde se busca la reutilización de objetos por medio de la herencia y el polimorfismo. Son usualmente descritos usando un formato que incluye la siguiente información [11]:

- Nombre del patrón
- Una descripción del problema que incluye un ejemplo concreto y la solución específica a él.
- Un resumen de los puntos a considerarse para llegar a la formulación de la solución general.
- La solución general.
- Las consecuencias, buenas y malas de usar la solución propuesta para solucionar el problema.
- Una lista de patrones relacionados.

2.12 Patrón de Diseño “**Strategy**”

A continuación se describen las características principales del patrón de diseño “**Strategy**” [5], que será contemplado en la arquitectura de la herramienta de este trabajo de investigación.

- **Intención:** Define una familia de algoritmos, encapsulando cada uno, y haciéndolos intercambiables. El patrón de diseño **Strategy** permite variar el algoritmo dependiendo de las necesidades del cliente
- **Motivación:** Su motivación esta dada, cuando diferentes clientes necesitan que se realice una algoritmo de diferentes maneras. Por ejemplo, se requiere llevar a cabo el algoritmo “Imprimir”, algunos clientes necesitarían imprimir en pantalla, otros en una impreso, etc.
- **Aplicabilidad:** Principalmente cuando se tienen muchas clases relacionadas que difieren solo en su comportamiento o cuando se necesita variaciones de un algoritmo.
- **Estructura:** La Figura 4 muestra la estructura del patrón “Strategy”.

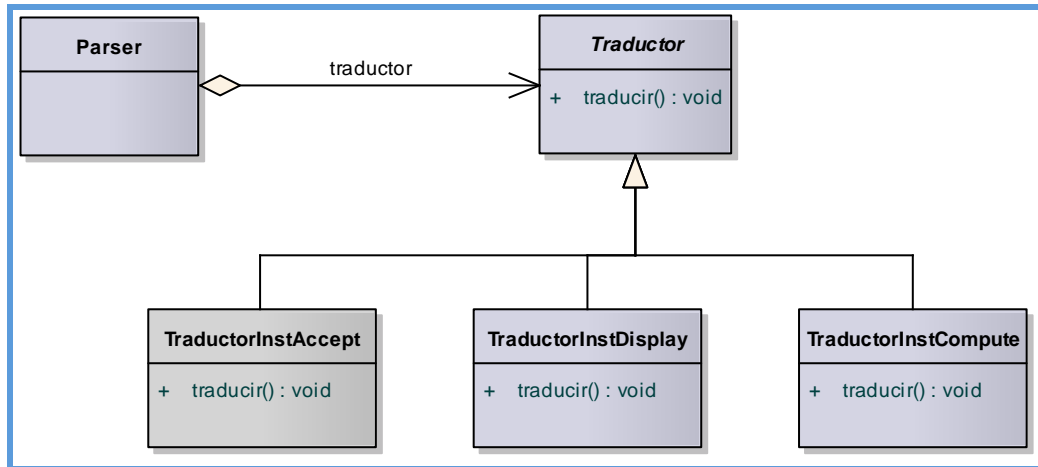


Figura 4. Estructura del patrón “Strategy”

- **Patrones Relacionados:** Template Method y State.

En este trabajo de tesis, el patrón de diseño “**Strategy**” se implementó a la hora de definir las diferentes estrategias de traducción para los distintos tipos de verbos o instrucciones encontrados en COBOL.

Capítulo 3. Análisis del Problema y Solución

En este capítulo se presenta el análisis realizado a COBOL en su versión estándar COBOL 85 y su correspondencia con Java y SQL. Se comienza con los tipos de datos en COBOL y se especifica a que tipos de datos pueden ser traducidos en Java, además de proporcionar un ejemplo. Posteriormente se analiza la traducción de los verbos (correspondientes a estatutos lógicos) más significativos, para los cuales se tendrá que llevar a cabo la construcción de rutinas y variables auxiliares que permitan igualar el funcionamiento entre los programas en COBOL y Java. Más adelante, se realiza el análisis de los verbos relacionados con el acceso a archivos y la manera en la cual serán traducidos a instrucciones en SQL.

Finalmente, se describe la arquitectura deseada para el producto final del presente trabajo de tesis y se describe un modelo conceptual del mismo.

3.1 Variables en COBOL

De acuerdo a la definición del lenguaje de programación COBOL las variables son definidas en la sección “Data Division” y provienen de tres fuentes:

- **File Section.** Variables usadas en el almacenamiento de información en archivos de datos. Las variables definidas en esta sección pueden ser registros y variables de condición.
- **Working Storage Section.** Variables usadas de forma local dentro de la sección de procedimientos. Estas variables pueden ser registros, variables indivisibles, de condición, etc.
- **Linkage Section.** Ligas hacia variables de datos definidas en otros programas.

El presente proyecto de tesis trabaja sólo con las dos primeras fuentes, ya que la tercera está fuera del alcance.

Las variables son únicas en todo el programa (variables globales) para evitar ambigüedades, no existe distinción entre mayúsculas y minúsculas y son definidas de acuerdo a un número de nivel (Ver Tabla 2). Las variables de tipo registro definen una estructura de acuerdo a las necesidades del programa en particular.

Tabla 2. Niveles de las variables en COBOL

Nivel	Descripción	Sección	
		Working-Storage	File
01-49	Composición de datos (registros)	X	X
66	Re-definibles durante la ejecución	X	
77	Indivisibles (primitivas)	X	
88	Variables de Condición	X	X

A continuación se explican tanto los tipos de datos como los tipos de variables y cómo puede ser su correspondencia en Java.

3.2 Tipos de datos

COBOL cuenta con tres tipos de datos: numéricos, alfanuméricos y alfabéticos. Los datos alfanuméricos y alfabéticos se pueden traducir a Java como el tipo de dato “String”. Por otro lado en los datos numéricos se puede llevar la siguiente clasificación:

- **Numérico de punto fijo (int).** Si en la definición no indica el punto decimal.
- **Numérico de punto flotante (float).** Si en la definición indica el punto decimal.

- **Numérico doble (double).** Si en la definición indica que corresponde a números con exponente.

Por lo tanto, para los tipos de datos numérico, alfanumérico y alfabético en COBOL corresponden los tipos de datos String, int, float y doble en Java de acuerdo a su definición.

3.3 Variables indivisibles

Estas variables corresponden al número de nivel 77 y significa que no pueden dividirse o descomponerse en otros datos. A la variable se le asigna directamente el tipo de dato al que corresponde. Además, en su definición, puede inicializarse con algún valor. En Java estas variables corresponden a las variables primitivas y deben ser declaradas directamente en la clase principal como públicas. Son consideradas variables globales.

3.4 Registros

En COBOL es posible definir estructuras a través del tipo de dato “registro”. Los niveles permitidos comienzan en 01 hasta el 49. Una variable puede ser definida en base a otras que deben estar declaradas con un número de nivel superior al número de nivel de ella.

```

01 REGISTRO-NAVES.
  02 CODIGO-NAVE          PIC 9(04) .
    88 VAL-COD-NAV        VALUE 1 THRU 9999.
  02 NOMBRE-NAVE          PIC X(15) .
  02 NOMBRE-CAP-NAVE      PIC X(15) .
  02 NACION-NAVE          PIC X(15) .
  02 FECHA-NAVE.
    05 DIA-NAVE           PIC 9(02) .
      88 VAL-DIA-NAV      VALUE 1 THRU 31.
    05 MES-NAVE           PIC 9(02) .
      88 VAL-MES-NAV      VALUE 1 THRU 12.
    05 AÑO-NAVE           PIC 9(04) .
      88 VAL-AÑO-NAV      VALUE 1999 THRU 9999.
  02 TRIPULANTES-NAVE     PIC 9(03) .
    88 VAL-TRIP-NAV      VALUE 1 THRU 999.
  02 EDAD-CAP-NAVE        PIC 9(02) .
    88 VAL-EDAD-CAP      VALUE 20 THRU 60.

```

Figura 5. Ejemplo de variable tipo registro

En la Figura 5 tenemos a la variable CODIGO-NAVE, la cual está constituida por las variables CODIGO-NAVE, NOMBRE-NAVE, NOMBRE-CAP-NAVE, etc., y a su vez, ellas pueden dividirse

de la misma manera. Las variables que ya no serán divididas deben especificar el tipo de dato (alfabético, alfanumérico o numérico) al que corresponde la variable.

En Java se puede implementar la composición de registros de datos por medio de la agregación de objetos, es decir, una clase agregada por cada registro diferente. Para el ejemplo de la Figura 5 deberán ser generadas las clases REGISTRO-NAVES Y FECHA-NAVE, como lo muestra la Figura 6.

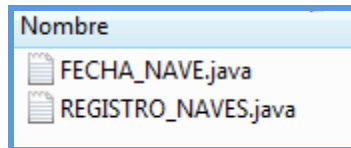


Figura 6. Archivos Java generados por la aplicación de conversión

En la Figura 7 se muestra el diagrama de composición de las clases entre las clases FECHA_NAVE y REGISTRO_NAVES. Es decir la clase REGISTRO_NAVES esta compuesta de la clase FECHA_NAVE y otros atributos primitivos.

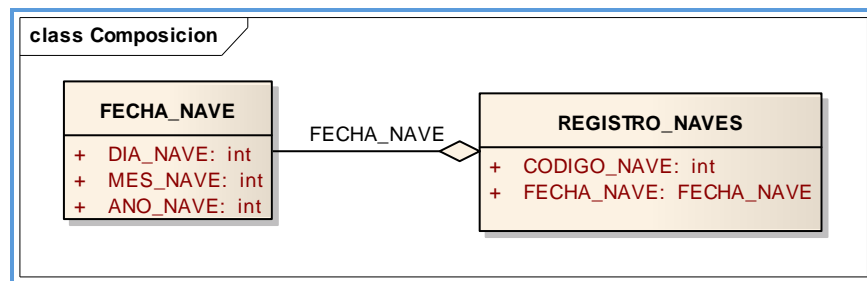
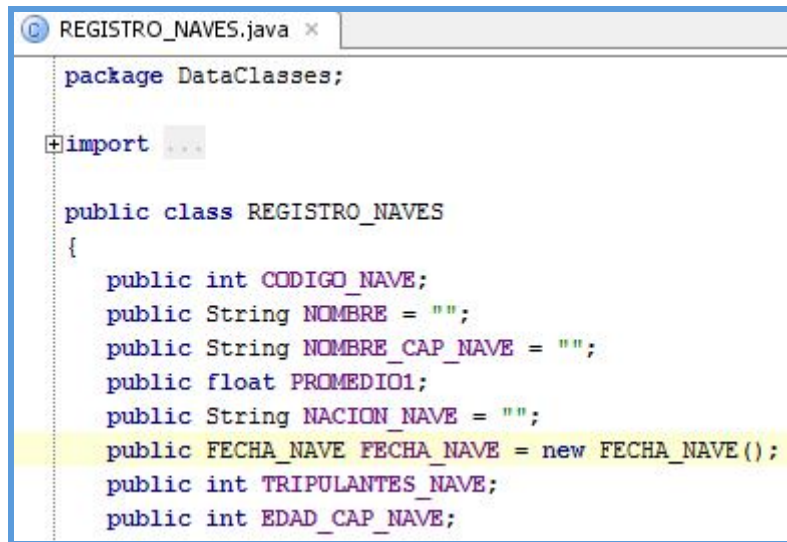


Figura 7. Diagrama de composición de clases

Es importante aclarar que la definición de las variables “registro” contenidas dentro de otra variable “registro” se realizará agregándola dentro de esa clase. Como se muestra en la Figura 8.



```

package DataClasses;

import ...

public class REGISTRO_NAVES
{
    public int CODIGO_NAVE;
    public String NOMBRE = "";
    public String NOMBRE_CAP_NAVE = "";
    public float PROMEDIO1;
    public String NACION_NAVE = "";
    public FECHA_NAVE FECHA_NAVE = new FECHA_NAVE();
    public int TRIPULANTES_NAVE;
    public int EDAD_CAP_NAVE;
}

```

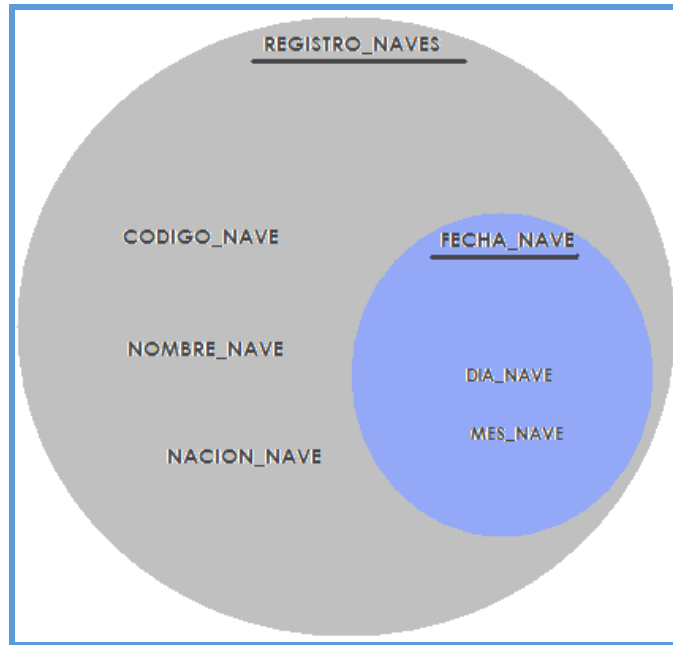
Figura 8. Agregación de clases en Java

Es importante destacar que en Java, no genera ningún conflicto el tener una variable de un objeto con el mismo nombre que su clase, como se muestra en la Figura 8 con el atributo FECHA_NAVE.

3.5 Calificación de Variables

Una característica particular de COBOL es el concepto de calificación de una variable. Cuando dos variables que corresponden a registros distintos son nombrados de la misma manera, es necesario llevar a cabo una calificación de la variable para saber a qué variable nos referimos en cierto momento, esto se logra colocando la palabra reservada “in” u “of” y posteriormente el nombre de la variable a la que pertenece, en tantos niveles como sea necesario hasta eliminar ambigüedades. Por el contrario si un variable que pertenece a un registro tiene un nombre único no es necesario declarar a que registro se refiere.

En Java, el acceso a los atributos de un objeto en su nivel más interno forzosamente se tiene que llevar a cabo recorriendo la asociación de los objetos hasta llegar al atributo/variable deseado. Por ejemplo la Figura 9 muestra el registro REGISTRO_NAVES, el cuál está compuesto por un conjunto de datos primitivos y otra variable de referencia como FECHA_NAVE. En Java para acceder a los atributos de FECHA_NAVE forzosamente se tiene que pasar a través de REGISTRO_NAVES; como la instrucción “System.out.println” mostrada en la Figura 9, utilizando un punto entre los objetos participantes.



```
System.out.println("El día es: " + REGISTRO_NAVES.FECHA_NAVES.DIA_NAVES);
```

Figura 9. Composición de datos y acceso a variables en Java

3.6 Variables de Condición

Una variable de condición es aquella que está asociada directamente al valor de otra variable. La variable a la cual está asociada debe tener definido un tipo de dato, es decir numérico, alfabético o alfanumérico.

A las variables de condición les corresponde el número de nivel 88 y pueden definir una serie de valores válidos o un rango de valores para la variable a la cual está asociada. Ejemplo Figura 10 (DIA-NAVE).

```
02 FECHA-NAVE.
   05 DIA-NAVE      PIC 9(02) .
   88 VAL-DIA-NAV   VALUE 1 THRU 31.
   05 MES-NAVE      PIC 9(02) .
   88 VAL-MES-NAV   VALUE 1 THRU 12.
   05 AÑO-NAVE      PIC 9(04) .
   88 VAL-AÑO-NAV   VALUE 1999 THRU 9999.
```

Figura 10. Ejemplo de las variables de condición

Las variables son usadas como condiciones, si es verdadero, significa que la variable a la cual esta asociada toma el valor definido en la variable de condición o se encuentra dentro del rango (Ver Figura 11). Es posible, además asignar el valor de "TRUE" a la variable de condición y eso significa que la variable asociada toma el valor de la variable de condición o se encuentra dentro del rango.


```

02 FECHA-NAVE.
    05 DIA-NAVE          PIC 9(02) .
    88 VAL-DIA-NAV      VALUE 1 THRU 31.
    05 MES-NAVE         PIC 9(02) .
    88 VAL-MES-NAV      VALUE 1 THRU 12.
    05 ANO-NAVE         PIC 9(04) .
    88 VAL-ANO-NAV      VALUE 1999 THRU 9999.

VALIDAR-FECHA-ARRIBO.
>*****
    DISPLAY '-Ingreso Fecha de Arribo : [00/00/0000]'
    PERFORM VALIDAR-DIA-ARRIBO UNTIL VAL-DIA-NAV.
    PERFORM VALIDAR-MES-ARRIBO UNTIL VAL-MES-NAV.
    PERFORM VALIDAR-ANO-ARRIBO UNTIL VAL-ANO-NAV.

```

Figura 11. Ejemplo del uso de las variables de condición

El lenguaje de programación Java no define las variables de condición, por lo que es necesario encontrar un conjunto de instrucciones que al ejecutarse lleven a cabo la misma funcionalidad.

Para traducir las variables de condición es necesario implementar los siguientes métodos:

- **Evaluar la variable condicional.** Evaluar si el valor de la variable asociada es igual o está dentro del rango que indica la variable condicional y así determinar el valor (falso/verdadero) de la variable de condición.
- **Asignar a una variable de condición el valor de “True”.** Asignar a la variable asociada el primer valor permitido de la variable de condición.

3.7 Estatutos Lógicos

COBOL fué desarrollado para enfocarse a aplicaciones orientadas a las empresas y no hacia aplicaciones científicas, es por ello que sus instrucciones se centran en operación aritméticas simples como sumas, restas, multiplicación y división. Además cuenta con instrucciones de selección o condicionales como la instrucción “if-else”, invocaciones a etiquetas que determinan el comienzo y fin de un bloque de ejecución (no métodos como los conocidos en otros lenguajes de programación como Java), entre otros. A continuación se analizan y se propone una solución de traducción de las instrucciones más relevantes de COBOL al lenguaje de programación Java.

3.7.1 Correspondencia entre párrafos (COBOL) y métodos (Java)

En COBOL no existe la definición de función o método, en lugar de ello, existen párrafos, que corresponden a la agrupación de instrucciones de control, decisión, etc. Cada uno de estos párrafos tiene definida como primera instrucción el nombre del párrafo y están definidas dentro de la sección "PROCEDURE SECTION".

Es posible establecer una correspondencia directa entre párrafos (COBOL) y métodos (Java), considerando los siguientes puntos:

- Los métodos en Java no recibirán parámetros, puesto que en COBOL no existe el mecanismo de paso de parámetros.
- Los métodos en Java no devolverán ningún valor, ya que en COBOL todas las variables son globales.
- El nombre del método será el mismo que el párrafo intercambiando "-" por "_" y reemplazando todos los caracteres en minúsculas por su correspondiente carácter en mayúscula.

3.7.2 El método "main" del programa en Java

En COBOL, el punto de entrada de todo programa es el primer párrafo encontrado y mientras no sea declarada la instrucción "STOP RUN", serán ejecutados todos los párrafos definidos de forma secuencial y ordenada.

Por su parte, en Java el punto de entrada del programa es el método "main", es en él donde se debe llevar a cabo la orquestación de los métodos (que corresponden a los párrafos en COBOL) para lograr la ejecución secuencial de los métodos. Para poder hacer posible que los métodos sean ejecutados de forma secuencial, conforme son implementados (hasta encontrar la instrucción "STOP RUN"), es necesario utilizar un contador o índice de secuencia de ejecución de métodos. El objetivo del índice es incrementar en uno cada vez que se traduzca un método hasta encontrar la instrucción "STOP RUN" e indicar dentro del método "main" el siguiente método a ser ejecutado. Ver Figura 12.

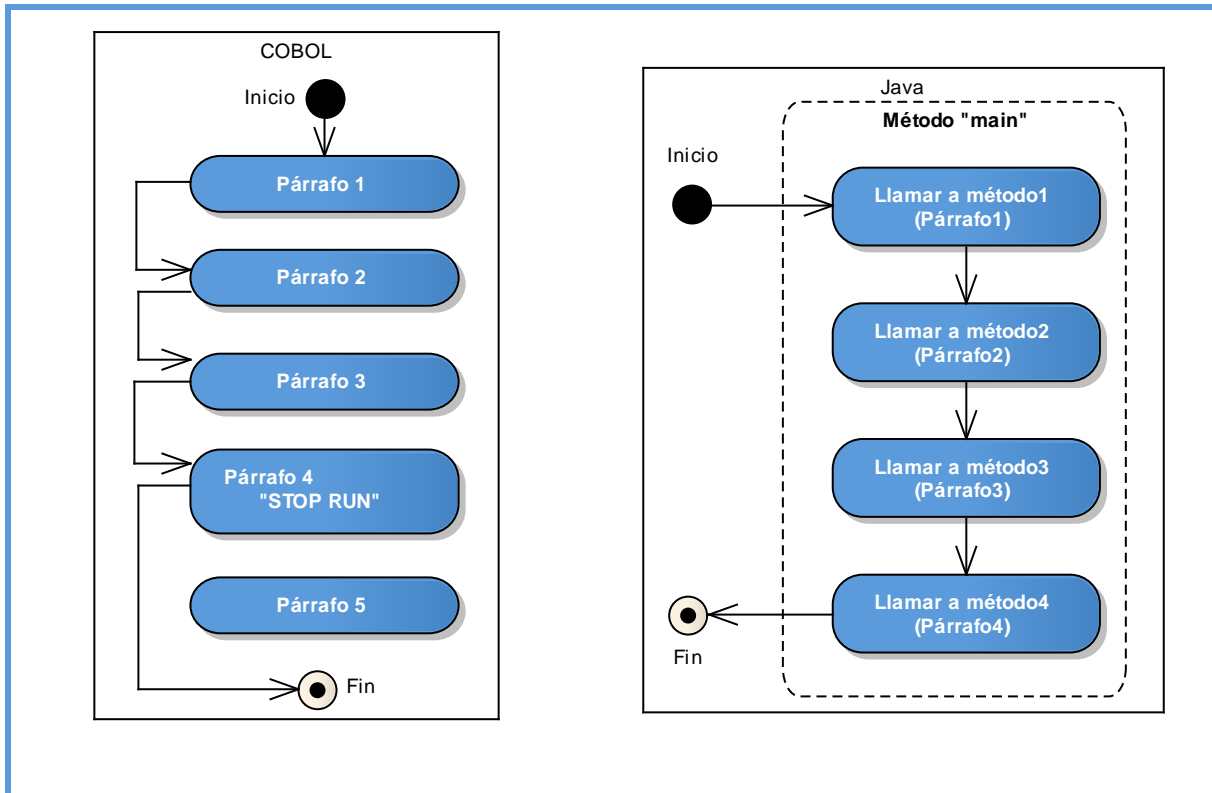


Figura 12. Correspondencia de ejecución secuencial COBOL - Java

3.7.3 La Instrucción GO TO

Una de las características más importantes de COBOL, es la posibilidad de usar las instrucciones “GO TO”. Aunque diversos autores recomiendan evitar el uso de estas instrucciones por la falta de control en determinado momento sobre la ejecución del programa, esto no limita a que existan programas escritos en COBOL con el uso de esta instrucción, principalmente en los programas escritos muchos años atrás.

Debido a que el uso de la instrucción GO TO es una práctica no recomendada, los lenguajes de programación como Java no implementan dicha instrucción por lo que es necesario buscar una alternativa de llevar a cabo la traducción de forma tal que la funcionalidad sea la misma a la dada por la instrucción en COBOL.

Cuando se usa la instrucción “GO TO” <nombre_párrafo>, significa que se quiere hacer un salto al párrafo indicado, rompiendo la forma secuencial de la ejecución. La instrucción puede ser colocada en cualquier parte dentro del párrafo. Una vez que se ejecutan las instrucciones correspondientes al párrafo invocado por la instrucción “GO TO”, el programa ya no ejecuta las instrucciones siguientes en el párrafo invocador, debe continuar con el siguiente párrafo que encuentre

secuencialmente después del párrafo invocado. La Figura 13 muestra el orden de ejecución de los párrafos/métodos.

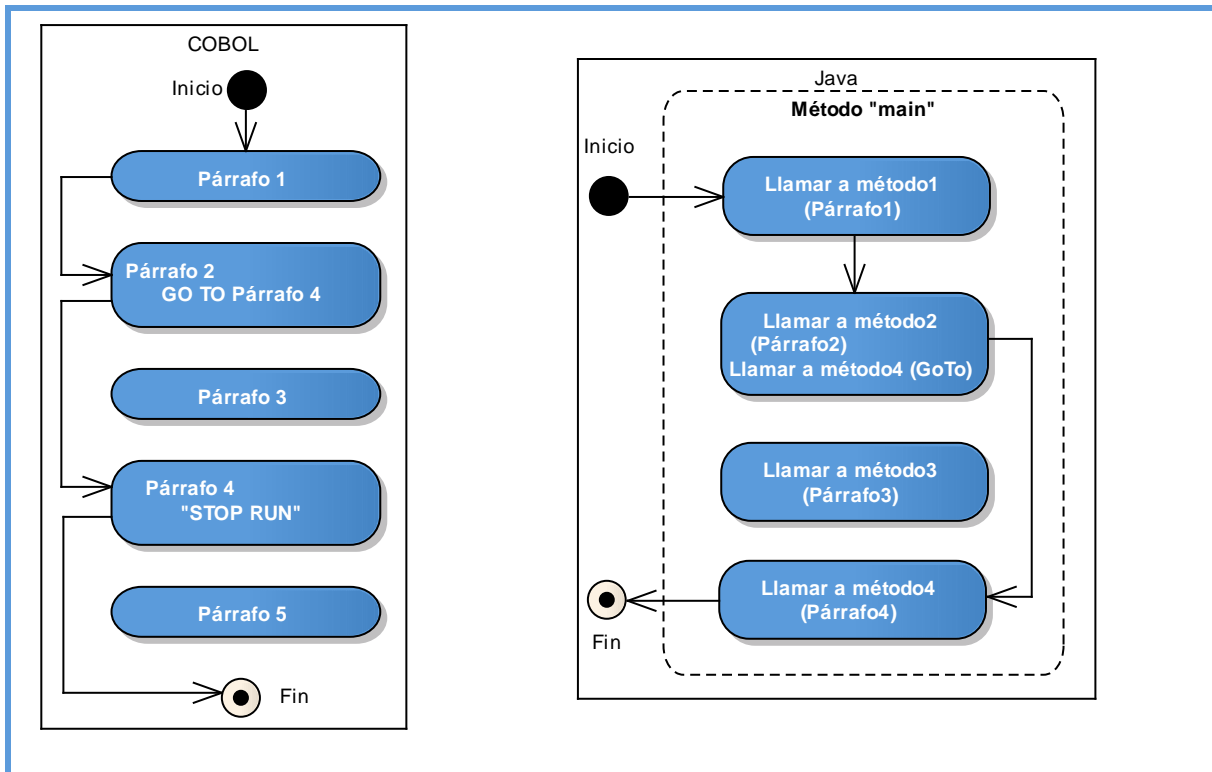


Figura 13. Correspondencia de ejecución de la instrucción "GO TO" en COBOL - Java

En Java, para evitar la ejecución de las instrucciones de un método, después de una instrucción "GO TO", primero se identifica que se encontró una instrucción "GO TO", posteriormente, una vez finalizado el método invocado por la instrucción "GO TO", se obliga el retorno al método principal "main", saltándose todas las demás instrucciones definidas posteriormente. La secuencia se controla dentro del método "main".

3.7.4 Traducción General de Verbos (Estatutos Lógicos)

En la Tabla 3 se enlistan los verbos de COBOL 85, un breve análisis de su funcionalidad y traducción a Java; y finalmente si se encuentran dentro del alcance del presente proyecto.

Tabla 3. Verbos (estatutos lógicos) de COBOL 85

Verbo	Descripción	¿Dentro del alcance?
Accept	Su propósito es capturar datos de entrada. En Java puede ser traducida como una instrucción de entrada del teclado.	Si
Add	La instrucción implementa la suma aritmética. Una diferencia particular es la asignación de la suma a más de una variable, por lo cual una instrucción en COBOL puede equivaler a más de una instrucción en Java.	Si
Alter	La instrucción modifica aquellos párrafos en los cuales sólo es ejecutada una instrucción "GO TO", y modifica el párrafo que es invocado. El estatuto correspondiente en Java debe modificar el nombre del método que es invocado.	Si
Call	La instrucción es usada para invocar subprogramas definidos en un archivo diferente.	No
Cancel	La instrucción inicializa subprogramas invocados desde el programa principal.	No
Compute	La instrucción implementa operaciones aritméticas con los símbolos de "+", "=", "-", etc. La traducción es directa.	Si
Continue	Rompe con la secuencia de instrucciones dentro de un "if-else". Su equivalente en Java es la instrucción "break".	Si
Display	La instrucción permite mostrar mensajes o información en el dispositivo de salida. La traducción en Java es la instrucción de salida "System.out.println", analizando el tipo de variable para poder ser mostrada.	Si
Divide	La instrucción implementa la división aritmética. La traducción es directa, considerando: <ul style="list-style-type: none"> En COBOL se puede dividir dos valores almacenando el resultado en alguno de los argumentos o en otra variable Es posible indicar que el residuo será almacenado Una instrucción en COBOL puede equivaler a más de una instrucción en Java 	Si
Evaluate	La instrucción se refiere a la selección de diferentes alternativas de flujo. La traducción se puede llevar a cabo usando instrucciones "if-else".	Si
Exit	La instrucción permite terminar el bloque de párrafos definidos. La traducción se puede llevar a cabo con la instrucción "break", en Java.	Si
Exit Program	La instrucción permite salir de subprogramas definidos en un archivo diferente.	No
Goback	La instrucción permite regresar al programa desde donde fué invocado el subprograma.	No
Goto	El detalle se encuentra en la sección §3.7.3.	Si
If-else	La instrucción es de tipo decisión y evalúa una o más condiciones, el equivalente en Java corresponde a las instrucciones "if" y "else".	Si
Initialize	Provee una forma conveniente para mover datos a campos seleccionados en cero y campos no-numéricos a espacios. Es necesario analizar el tipo de dato para evaluar la inicialización de la variable.	Si
Inspect	Realiza búsquedas de información en datos de tipo alfanumérico. Comprende las	Si

	siguientes funcionalidades: <ul style="list-style-type: none"> • Contador de caracteres • Contador de palabras dentro de otra palabra (Las primeras, todas, en una sub-cadena) • Reemplazar palabra (al principio, todas, en una sub-cadena, la primera) • Reemplazar todos los caracteres por otro carácter. • Reemplazar y contar 	
Merge	La instrucción trabaja con tablas.	No
Move	La instrucción asigna un valor a una variable. En Java corresponde a la asignación.	Si
Multiply	La instrucción implementa la multiplicación aritmética. La traducción es directa, considerando: <ul style="list-style-type: none"> • En COBOL se puede multiplicar dos valores almacenando el resultado en alguno de los argumentos o en otra variable • Una instrucción en COBOL puede equivaler a más de una instrucción en Java 	Si
Perform	La instrucción invoca la ejecución de un párrafo. La traducción en java corresponde a la invocación de los métodos.	Si
Release	La instrucción trabaja con tablas (arreglos bidimensionales).	No
Return	La instrucción trabaja con subprogramas definidos en un archivo diferente.	No
Search	La instrucción trabaja con tablas (arreglos bidimensionales).	No
Set	La instrucción asigna "true" a las variables de condición (dentro del alcance de la tesis). Además asigna un valor a una variable de tipo arreglo (fuera del alcance de la tesis).	Parcial
Sort	La instrucción trabaja con tablas.	No
Start	La instrucción trabaja con tablas.	No
Stop	La instrucción determina el fin en la secuencia de ejecución de párrafos. En el análisis de programas de COBOL indica hasta donde serán ejecutados los métodos en orden secuencial.	Si
String	La instrucción realiza operaciones de concatenación de cadenas.	Si
Sustract	La instrucción implementa la resta. Su traducción fue directa, solo con algunos cambios por ejemplo, en COBOL se pude restar un valor a un conjunto de variables, por lo cual una instrucción en COBOL puede equivaler a más de una instrucción en Java.	Si
Unstring	La instrucción realiza operaciones sobre cadenas.	Si

3.8 Estatutos de Acceso a Archivos

Para el estándar COBOL 85 el medio de almacenamiento de información solamente se puede llevar a cabo por medio del sistema de archivos. Uno de los objetivos del presente proyecto de tesis, es traducir el conjunto de instrucciones de acceso a archivos a un conjunto de clases en Java y una base de datos en SQL (Ver Figura 14).

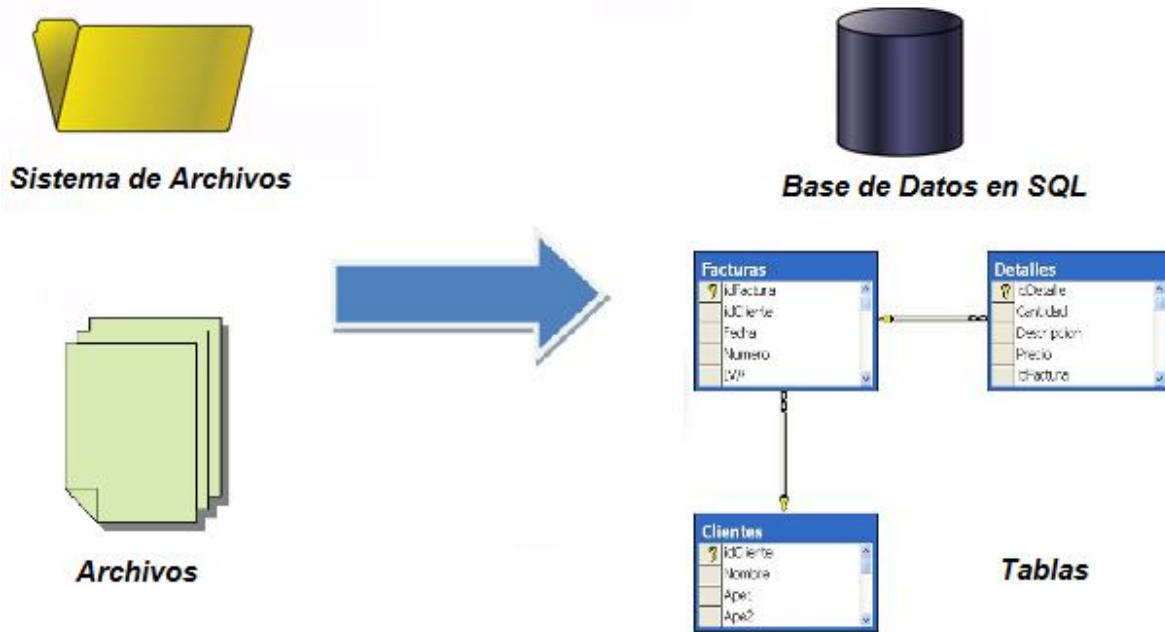


Figura 14. Esquema de la traducción de Cobol a SQL

De manera general la Tabla 4 muestra la equivalencia entre estatutos de acceso a archivos en COBOL e instrucciones en SQL.

Tabla 4. Correspondencia COBOL-SQL (estatutos de acceso a archivos)

Instrucción en COBOL	Equivalencia en SQL
Open	Control de acceso a las tablas en SQL (Instrucciones en Java)
Close	
Read	Select
Write	Insert
Rewrite	Update
Delete	Delete

3.8.1 Organización del Archivo

El concepto de organización de un archivo se refiere a la forma como se ordenan los registros de datos en un medio de almacenamiento de archivos. En COBOL existen tres métodos principales para la organización de archivos: secuencial, relativo e indexado.

En un archivo secuencial, los registros se almacenan en orden consecutivo y son leídos en el mismo orden como fueron escritos. En los archivos indexados, se crea un índice para que los

registros se puedan localizar directamente, sin llegar a ellos secuencialmente. Finalmente, la organización de archivos relativo significa que la información se almacena de tal manera que cada registro tenga un identificador o llave a través del cual se pueda acceder.

Para llevar a cabo la traducción de los diferentes tipos de archivos se propone generar en cada una de las tablas de SQL generadas un campo auxiliar, el cuál sea un índice y permita llevar a cabo el acceso secuencial a los registros, y permita simular el manejo de la dirección en los archivos de tipo relativo.

3.8.2 Tipos de Acceso al Archivo

El acceso a los archivos en COBOL puede ser de tres tipos: secuencial, aleatorio o dinámico. Para los tres tipos de archivos, los registros pueden ser accedidos de manera secuencial, es decir se recorren los registros uno a uno, como fueron almacenados en los archivos secuenciales, de acuerdo a una llave relativa para los relativos y un registro llave para los indexados.

Por otro lado, el acceso aleatorio sólo aplica para los archivos indexados y relativos. El acceso se realiza por medio del registro llave para los archivos indexados y la llave relativa para los archivos relativos.

El acceso debe declararse como dinámico cuando se acceda a los archivos de forma aleatoria y otras de forma secuencial, en el mismo programa de acuerdo a las necesidades del negocio.

Los tipos de acceso permitidos por tipo de archivo se muestran en la Tabla 5.

Tabla 5. Tipo de acceso permitidos en COBOL

Tipo de Acceso	Tipo de Archivo		
	Secuencial	Relativo	Indexado
Secuencial	✓	✓	✓
Aleatorio		✓	✓
Dinámico		✓	✓

Los verbos (estatutos) relacionados con el manejo de los archivos son seis, mismos que se describen a continuación [13]:

- **Close.** La instrucción se refiere al cierre de archivos. De acuerdo al trabajo descrito en el trabajo de tesis definido en [13], existen unas clases de interfaz hacia la base de datos que sustituye a los archivos. La funcionalidad se debe implementar impidiendo acceder a la base de datos una vez que el archivo sea cerrado.
- **Delete.** La instrucción se refiere a la eliminación de registros de archivos. De acuerdo a las clases de interfaz hacia la base de datos que sustituye a los archivos, se implementa invocando el método “delete” de la tabla correspondiente contenida en la clase CBDSQL*.
- **Open.** La instrucción se refiere a la apertura de archivos. De acuerdo a las clases de interfaz hacia la base de datos que sustituye a los archivos, se implementa indicando que la tabla equivalente al archivo puede ser accedida, indicando el tipo de apertura.
- **Read.** La instrucción se refiere a la lectura de registros de archivos. De acuerdo a las clases de interfaz hacia la base de datos que sustituye a los archivos, se implementa invocando el método “read” de la tabla correspondiente contenida en la clase CBDSQL*.
- **Rewrite.** La instrucción se refiere a la actualización de registros de archivos. De acuerdo a las clases de interfaz hacia la base de datos que sustituye a los archivos, se implementa invocando el método “update” de la tabla correspondiente contenida en la clase CBDSQL*.
- **Write.** La instrucción se refiere a la escritura de registros de archivos. De acuerdo a las clases de interfaz hacia la base de datos que sustituye a los archivos, se implementa invocando el método “write” de la tabla correspondiente contenida en la clase CBDSQL*.

A continuación se especifican las particularidades de la traducción de las instrucciones de acceso a archivos (Open, Close, Read, Write, Rewrite y Delete) a instrucciones en Java y SQL.

3.8.3 Traducción del Verbo: Open

El verbo Open inicia el procesamiento de un archivo. La Figura 15 muestra la regla gramatical para el verbo Open [3].

```
<OPEN> ( <INPUT> ( FileName ()
                [ ( <REVERSED> | [ <WITH> ] <NO> <REWIND> ) ] )+
    | <OUTPUT> ( FileName () [ [ <WITH> ] <NO> <REWIND> ] )+
    | <I_O> ( FileName () )+
    | <EXTEND> ( FileName () )+
    )+
```

Figura 15. Regla gramatical de la instrucción Open

* Contiene todas las tablas equivalentes a los archivos de la aplicación.

Los modos <INPUT> y <OUTPUT> contienen la opción <NO REWIND> que se puede emplear con los archivos de cinta magnética. En el caso de la traducción a instrucciones SQL, esta

funcionalidad no fue implementada, ya que no existe el concepto de rebobinar, los registros están disponibles en todo momento.

La opción <I-O> se emplea para archivos de almacenamiento masivo y permite al programa tener entrada y salida de un archivo. Se usa cuando se desea cambiar un registro que ya se ha leído. La opción <EXTEND> se usa para agregar nuevos registros al final del archivo y es un caso especial del modo <I-O>. La Tabla 6 resume el empleo correcto de los verbos de entrada y salida dependiendo del modo de la instrucción OPEN [2], es necesario conocerlo para llevar a cabo las validaciones correspondientes en la implementación.

Tabla 6. Combinaciones permitidas para opciones del modo Open y los verbos de entrada-salida

Instrucción	Modo Open			
	INPUT	OUTPUT	I/O	EXTEND
READ	X		X	
WRITE		X		X
REWRITE			X	
DELETE			X	

Nota importante: En esta tesis la traducción de la instrucción Open no abarca las opciones <REWIND> y <NO REWIND>, ya que en SQL, en cualquier momento los registros están disponibles; sin embargo el uso de estas instrucciones en el código COBOL no afecta el funcionamiento del analizador, ya que simplemente la identifica sin realizar ninguna traducción o generación de código.

3.8.4 Traducción del Verbo: Close

El verbo Close es usado para cerrar un archivo, antes de cerrarlo el archivo debe ser abierto y significa que se ha terminado de procesar la información de un archivo. La Figura 16 muestra la regla gramatical para el verbo Close.

```

<CLOSE>
( FileName ()
    [ ( ( <REEL> | <UNIT> ) [ ( [ <FOR> ] <REMOVAL>
                                | [ <WITH> ] <NO> <REWIND>
                                )
      ]
    | [ <WITH> ] ( <NO> <REWIND> | <LOCK> )
    )
]
)+

```

Figura 16. Regla gramatical de la instrucción Close

Los modos <INPUT> y <OUTPUT> contienen la opción <NO REWIND> que se puede emplear con los archivos de cinta magnética. En el caso de la traducción a instrucciones SQL, no existe una traducción específica. El cierre de un archivo simplemente invalida su uso en instrucciones posteriores.

3.8.5 Traducción del Verbo: Delete

El verbo Delete es usado para eliminar un registro de un archivo. Para los archivos secuenciales, indexados o relativos y de acceso secuencial la regla gramatical es la siguiente:

```

<DELETE> FileName () [ <RECORD> ]
[ <END_DELETE> ]

```

Figura 17. Regla gramatical de la instrucción Delete (acceso secuencial)

En el momento de llevar a cabo la traducción es necesario considerar lo siguientes puntos:

- Validar que el modo de apertura permita la operación.
- Validar y verificar que se ha leído un registro previamente, como lo hace COBOL.
- Si se trata de un archivo relativo indicar el valor de la llave relativa.
- Para un archivo indexado verificar el registro llave.
- Considerar que para los archivos relativos e indexados se pueden ejecutar otras instrucciones tanto si existe un error como si no en las llaves (ver Figura 18).

```

[ <INVALID> [ <KEY> ] StatementList() ]
[ <NOT> <INVALID> [ <KEY> ] StatementList() ]
[ <END_DELETE> ]

```

Figura 18. Regla gramatical de la instrucción Delete (archivos indexados y relativos)

3.8.6 Traducción del Verbo: Write

El verbo Write es un estatuto de salida y es usado para imprimir información ya sea en un archivo o en la impresora. En un archivo puede escribir información de registros, como medio de almacenamiento de información o reportes generados durante la ejecución de algún programa.

La traducción de esta instrucción abarca la impresión de reportes en archivos de texto y el almacenamiento de información en una base de datos en SQL. En este trabajo de tesis, la salida a impresora no es considerada para la traducción.

Para los archivos secuenciales, la escritura se realiza siempre al final del archivo, en el caso de los programas traducidos, el registro debe ser agregado a la tabla correspondiente en la base de datos. Por otro lado, si se trata de la escritura de un reporte, entonces la información debe ser escrita en un archivo de texto. Es posible agregar la ejecución de instrucciones tanto si el archivo ha llegado al final como si no, de acuerdo a la gramática de COBOL [12]. Ver Figura 19.

```
<WRITE> RecordName() [ <FROM> Identifier() ]
[ ( <BEFORE> | <AFTER> ) [ <ADVANCING> ]
  ( <PAGE>
    | ( IntegerConstant()
      | Identifier()
      | Literal() ) [ ( <LINE> | <LINES> ) ]
  )
]
[ [ <AT> ] ( <END_OF_PAGE> | <EOP> ) StatementList() ]
[ <NOT> [ <AT> ] ( <END_OF_PAGE> | <EOP> ) StatementList() ]
[ <END_WRITE> ]
```

Figura 19. Regla gramatical de la instrucción Write (archivos secuenciales)

En los archivos indexados o relativos se usa una llave para escribir, ya sea relativa o dentro del registro, para especificar y validar que los registros sean únicos. De acuerdo a la regla gramatical es posible agregar instrucciones cuando la operación fué llevada correctamente o en caso de haber ocurrido algún error. Ver Figura 20.

```
<WRITE> RecordName() [ <FROM> Identifier() ]
[ <INVALID> [ <KEY> ] StatementList() ]
[ <NOT> <INVALID> [ <KEY> ] StatementList() ]
[ <END_WRITE> ]
```

Figura 20. Regla gramatical de la instrucción Write (archivos relativos e indexados)

En la sección §3.8.9 se presentan los diferentes códigos de información generados después de la ejecución de instrucciones de acceso a archivos y su significado.

3.8.7 Traducción del Verbo: Rewrite

La instrucción Rewrite en COBOL es el equivalente de la instrucción Update en SQL, pero con algunas particularidades de acuerdo al tipo de archivo al que se acceda. Es importante resaltar que la actualización es llevada a cabo registro por registro, no existen actualizaciones masivas (como es permitido en SQL).

```
<REWRITE> RecordName() [ <FROM> Identifier() ]  
[ <END_REWRITE> ]
```

Figura 21. Regla gramatical de la instrucción Rewrite (archivos secuenciales)

En el caso de los archivos secuenciales, el registro a ser actualizado debe ser el último leído (Figura 21). Para los archivos indexados y relativos es necesario indicar la llave del registro a ser actualizado. Al igual que en los verbos Delete y Write es posible ejecutar instrucciones en caso de error o ejecución exitosa (Ver Figura 22).

```
<REWRITE> RecordName() [ <FROM> Identifier() ]  
[ <INVALID> [ <KEY> ] StatementList() ]  
[ <NOT> <INVALID> [ <KEY> ] StatementList() ]  
[ <END_REWRITE> ]
```

Figura 22. Regla gramatical de la instrucción Rewrite (archivos relativos e indexados)

3.8.8 Traducción del Verbo: Read

El verbo Read en COBOL es usado para leer la información de los archivos, es uno de los verbos más importantes y más complejos de los verbos de acceso a archivos ya que dependiendo de la información proporcionada se ejecuta la instrucción, leyendo la información secuencial o aleatoriamente.

Para los archivos secuenciales, la lectura se realiza en el orden en que fueron agregados los registros (Ver Figura 23).

```
<READ> FileName() [ <NEXT> ] [ <RECORD> ]  
[ <INTO> Identifier() ]  
[ [ <AT> ] <END> StatementList() ]  
[ <NOT> [ <AT> ] <END> StatementList() ]  
[ <END_READ> ]
```

Figura 23. Regla gramatical de la instrucción Read (archivos secuenciales)

En el caso de los archivos relativos, el acceso puede ser secuencial o aleatorio, dependiendo si se especifica que se requiere leer el siguiente registro o la llave del registro a leer. Los programas traducidos de COBOL a Java hacen uso del índice para el acceso secuencial y aleatorio para el caso de los archivos indexados (Ver Figura 24).

```
<READ> FileName() [ <NEXT> ] [ <RECORD> ]
[ <INTO> Identifier() ]
[ <KEY> [ <IS> ] Identifier() ]
[ <INVALID> [ <KEY> ] StatementList() ]
[ <NOT> <INVALID> [ <KEY> ] StatementList() ]
[ <END_READ> ]
```

Figura 24. Regla gramatical de la instrucción Read (archivos indexados y relativos)

3.8.9 Códigos de Información para Instrucciones de Acceso a Archivos

COBOL implementa una serie de códigos de información una vez que han sido ejecutadas las instrucciones de acceso a archivos. Todos ellos ayudan al programador a tener más información específica de los problemas ocurridos o por el contrario si la ejecución fue exitosa. La Tabla 7 muestra los códigos dependiendo del tipo de archivo. Estos códigos deben ser considerados durante la implementación de los verbos relacionados con el acceso a archivos en SQL.

Tabla 7. Códigos de Error en el manejo de archivos secuenciales/relativos/indexados

Código	Tipo de Archivo	Descripción
00	(S)(I)(R)	Ejecución exitosa
02	(I)	Terminación satisfactoria pero se detectaron claves duplicadas.
04	(S)(I)(R)	Un registro cuya longitud es inconsistente con la descripción del registro para el archivo que se ha leído.
05	(S)(I)	Un archivo que se ha abierto pero no está presente.
10	(S)(R)	Condición de Fin de archivo.
16	(S)	Una instrucción READ se ejecutó mientras que la condición de fin de archivo era verdadera. (Se intentó leer después de fin de archivo).
21	(I)	Errores de secuencia encontrados. El programa cambió la clave primaria entre una instrucción y un REWRITE.
22	(R)	Se trató de escribir un registro donde ya se había escrito un registro.
23	(I)(R)	Intenta leer un registro que no existe.
30	(S)	Se intentó OPEN como INPUT, I-O, EXTEND a un archivo sin esa opción.
38	(S)	Se intentó OPEN en un archivo que ha sido cerrado CLOSED With LOCK.
39	(S)	Error durante la ejecución del OPEN, debido a inconsistencias entre la descripción y el archivo real.
41	(S)(I)(R)	Se intentó OPEN en un archivo que ya esta abierto.
42	(S)(I)(R)	Se intentó CLOSE en un archivo que no esta abierto.
43	(S)(R)	Un READ correcto no precede la ejecución del comando REWRITE.

	(I)	Intenta DELETE o REWRITE sin la ejecución anterior de un correspondiente READ.
46	(S)	Intentó READ al registro siguiente que es inexistente.
47	(S)(I)(R)	Intentó READ en un archivo no abierto en los modos INPUT o I-O.
48	(S)(I)(R)	Intentó WRITE en un archivo no abierto en los modos OUTPUT o EXTEND.
49	(S)(R)	Intentó REWRITE en un archivo no abierto en modo I-O.
	(I)	Intentar DELETE o REWRITE en un archivo no abierto en modo I-O.

(S). Secuencial

(I). Indexado

(R). Relativo

3.9 Manejo de verbos no definidos en el estándar COBOL 85

Además del estándar de COBOL 85 existen otros dialectos derivados de éste. Para todos ellos es necesario implementar una estrategia de recuperación de errores, la cuál considere ignorar los verbos no definidos en el estándar. La estrategia considera el punto (.) como delimitador, una vez que la herramienta no reconozca un verbo pasará todas las palabras hasta encontrar un punto y a partir de él, continuar con la traducción, tantas veces como sea necesario. Si se sabe previamente que el programa está escrito en una versión diferente a COBOL 85 debe ser inspeccionado para encontrar los verbos no definidos y colocar un punto enseguida de la finalización de la instrucción (si es que no lo tuviera). Esta inspección debe ser manual, ya que el proyecto no contempla el pre-procesamiento para identificar los verbos fuera del estándar COBOL 85.

3.10 Arquitectura propuesta

Uno de los alcances planteados para este proyecto de tesis es que la herramienta que permite llevar a cabo la interpretación de código escrito en COBOL para generar instrucciones en lenguaje Java, considerando los estatutos lógicos y de acceso a archivos, sea desarrollada como un Servicio Web. Esto con el objetivo de brindar amplias opciones de conectividad y uso por parte de otras aplicaciones, ya que una de sus características es la independencia de plataforma y sistema operativo.

El ambiente que soporta a los Servicios Web se encuentra basado en la Arquitectura Orientada a Servicios (SOA), el cual se ilustra en la Figura 25.



Figura 25. Arquitectura orientada a servicios

El servicio debe ser instalado del lado del Proveedor del servicio, opcionalmente puede ser registrado en la UDDI, mientras los clientes acceden al servicio mediante una aplicación vía Internet.

Por otra parte, la Figura 26 muestra un modelo conceptual del servicio de traducción con una serie de elementos clave relacionados para llevar a cabo la traducción del lenguaje COBOL a Java. Los principales elementos son:

- **Programa COBOL.** Programa fuente que será traducido.
- **Analizadores.** Conjunto de clases en Java generadas por JavaCC que implementan el parser.
- **Acciones Semánticas.** Conjunto de instrucciones en Java que de acuerdo a la expresión reconocida en el lenguaje fuente lleva a cabo la traducción en Java correspondiente.
- **Bitácora de Error.** Archivo con extensión .txt que contiene la descripción de los errores encontrados durante la traducción.
- **Clases del Modelo.** Conjunto de clases encargadas de interactuar con la base de datos en SQL y el programa final traducido en Java.
- **Clases de Control.** Conjunto de clases generadas como resultado de la traducción. Contiene una o más instrucciones en Java equivalentes a una instrucción en COBOL.

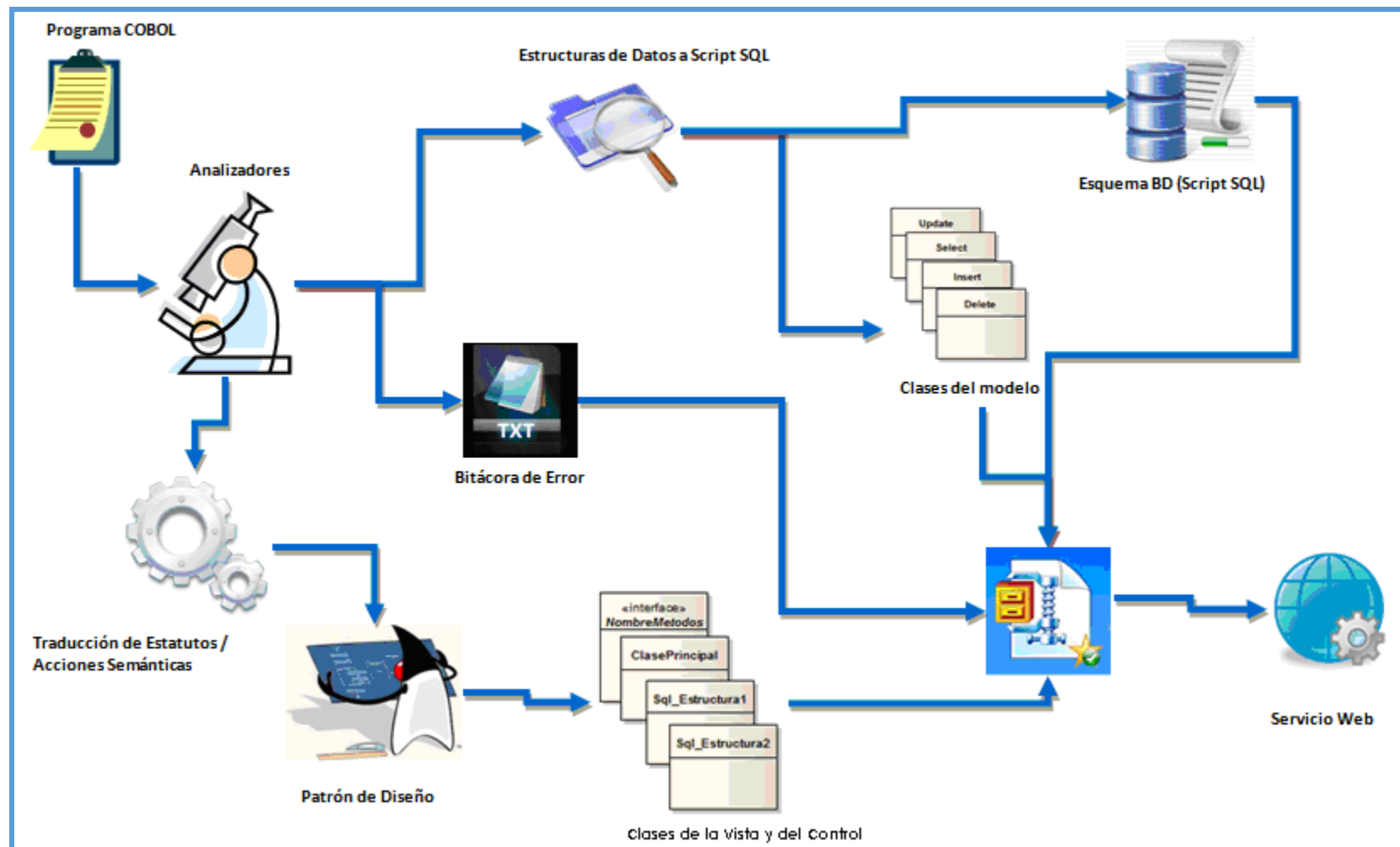


Figura 26. Modelo conceptual de la herramienta de traducción

Capítulo 4. Análisis, Diseño e Implementación

En el presente capítulo se presenta el diseño e implementación realizado para la obtención de un sistema de traducción de COBOL a Java. La herramienta consiste de dos partes:

- **Servicio Web.** Sistema de software realizado bajo la arquitectura SOA (Service Oriented Architecture) encargado de llevar a cabo la traducción de programas hechos en COBOL hacia lenguaje Java.
- **Cliente.** Aplicación Web, la cuál realiza la interfaz al usuario. Envía el código fuente al servicio Web, recibe y procesa la información de forma tal que sea comprensible al usuario final.

En conjunto, las dos aplicaciones deberán corresponder al cumplimiento de las siguientes funcionalidades:

- Acceder al sistema.

- Salir del sistema.
- Análisis de código fuente del sistema.
- Traducción de instrucciones relacionadas con el acceso a archivos.
 - Generación del script correspondiente al esquema de la base de datos relacional.
- Generación de clases de objetos para la interfaz de acceso a la Base de Datos relacional desde cualquier aplicación en Java.
- Traducción completa de programas escritos en COBOL al lenguaje de programación Java.
 - Traducción de sentencias correspondientes a estructuras de datos.
 - Traducción de sentencias de control.
 - Traducción de sentencias relacionadas al acceso, lectura y escritura de archivos, usando la interfaz proporcionada por la herramienta en pasos previos.
- Traducción de sentencias relacionadas con procedimientos en COBOL a métodos en Java.

En la Figura 27 se muestra un esquema donde interactúan tanto el Servicio Web como el cliente, a través de la Internet, aunque las dos aplicaciones pueden convivir en el mismo equipo físico.

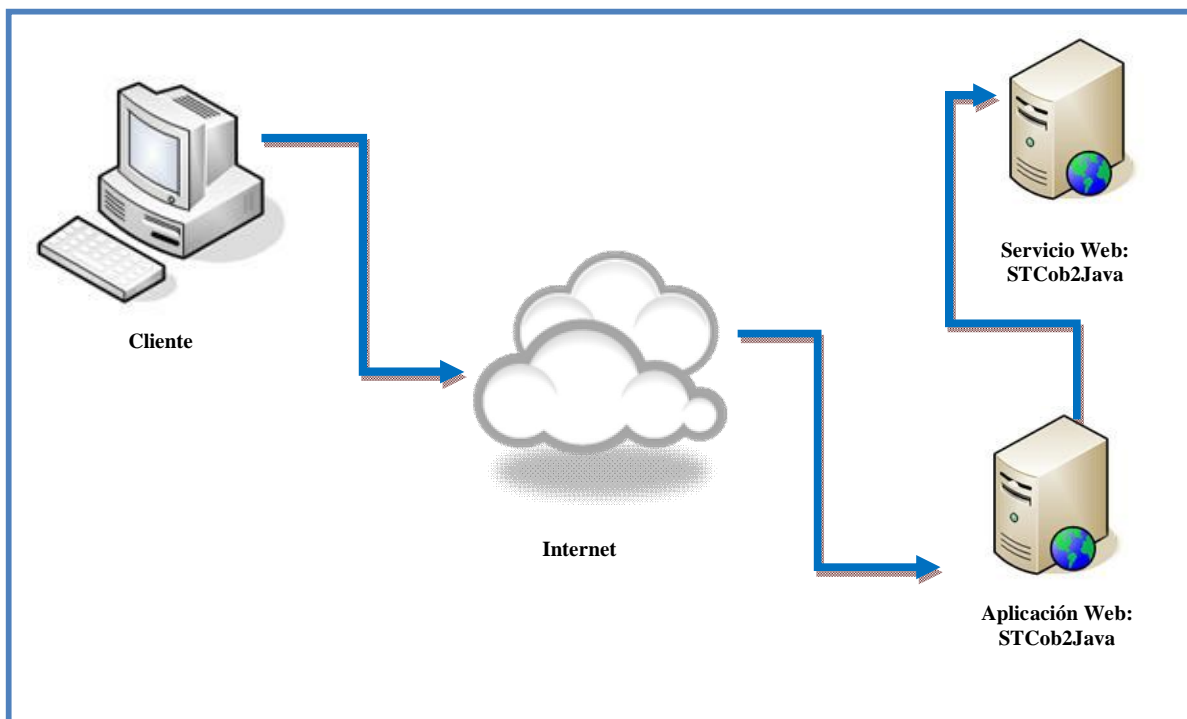


Figura 27. Esquema general de la herramienta STCob2Java

En las siguientes secciones se mostrará y explicará el proceso de Análisis Diseño y Desarrollo de las aplicaciones descritas; desde los listados de requerimientos, pasando por el modelo de casos

de uso y los diagramas de clases. Finalmente se muestran algunas pantallas de la aplicación cliente.

4.1 Análisis y Diseño del Servicio Web

4.1.1 Requerimientos Funcionales: Modelo de Casos de Uso

A continuación se describen cada una de las funcionalidades que la herramienta “Sistema de Transformación de COBOL a Java” llevará a cabo, se presenta una breve descripción de los objetivos planeados, omitiendo detalles y el cómo deberán ser implementados. Además de algunas de las restricciones que deberá presentar el sistema.

1. **SW_STCob2Java_RF_01.** El servicio debe permitir al cliente las opciones de realizar la conversión completa y la traducción únicamente de instrucciones de acceso a archivos del programa escrito en COBOL.
2. **SW_STCob2Java_RF_02.** El sistema debe generar un script en lenguaje SQL estándar 92 que contenga el esquema de la base de datos y pueda ser ejecutado en algún manejador de Base de Datos que soporte el estándar SQL 92.
3. **SW_STCob2Java_RF_03.** El sistema debe ser capaz de generar un conjunto de clases de objetos que permitan realizar la comunicación entre aplicaciones en Java y la base de datos a partir de la definición de archivos en el programa COBOL fuente.
4. **SW_STCob2Java_RF_04.** El sistema debe realizar la traducción automática de estructuras de datos en lenguaje de programación COBOL a una serie de clases abstractas y derivadas en Java, las cuales contengan los datos y métodos relacionados con la estructura.
5. **SW_STCob2Java_RF_05.** El sistema debe realizar la traducción automática de las diferentes sentencias del lenguaje COBOL a Java, como sentencias de procedimientos, tipos de datos simples, y sentencias de control.

Cada uno de los requerimientos funcionales del sistema fueron especificados a través un modelo de casos de uso en UML, para los cuales se identificaron los actores involucrados (Tabla 8) y las diferentes interacciones entre dichos usuarios y el sistema. Los diagramas y descripciones detalladas de cada caso de uso se presentan a continuación.

Tabla 8. Diccionario de actores involucrados del servicio web

Diccionario de Actores			
Actor	Descripción	Responsabilidad	Privilegios
Cliente	Es cualquier aplicación cliente que solicite al Servicio Web la conversión de un programa escrito en COBOL a Java.	Proporciona una cadena (String), la cual contiene el código escrito en lenguaje COBOL 85. Además indica si la traducción será completa o parcial.	Puede solicitar la traducción de archivos escritos en COBOL a Java.

El Sistema de Transformación de COBOL a Java consta de tres casos de uso principales: SW-CU1:ConvertirCobol2J el cuál incluye al caso de uso SW-CU2:TransformarCobol2SQL y extiende a SW-CU3:TransformarCobol2Java, ya que en todos los casos el servicio Web llevará a cabo la traducción de esquema de datos, dejando como opcional la traducción de los estatutos lógicos.

El segundo caso de uso SW-CU2:TransformarCobol2SQL se refiere a la traducción de instrucciones de manipulación de archivos y definiciones de estructuras de archivos escritas en lenguaje del estándar de COBOL 85, hacia instrucciones de manipulación de datos escritas en lenguaje Java y la definición de una base de datos relacional usando el lenguaje SQL 92.

Finalmente el caso de uso SW-CU3:TransformarCobol2Java esta involucrado en la traducción de las instrucciones relacionadas con la lógica de control y que no están relacionadas con la manipulación de archivos escrito en COBOL, hacia código escrito en lenguaje Java. La Figura 28 muestra el diagrama de casos de uso general del sistema, en la que se describe la relación del actor con los casos de uso descritos previamente.

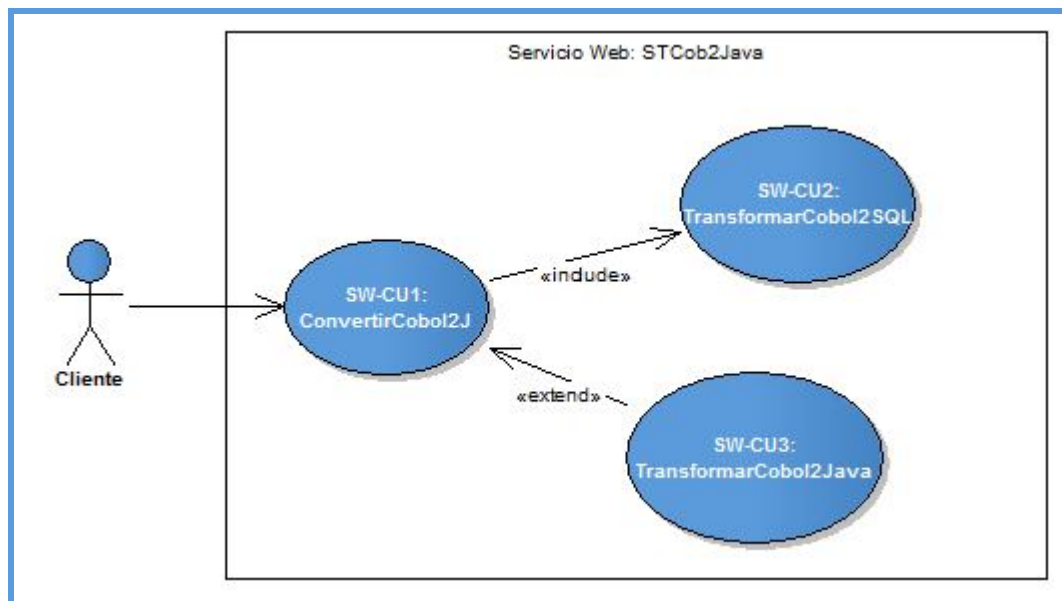


Figura 28. Diagrama general de casos de uso del servicio web

A su vez, el caso de uso SW-CU2:TransformarCobol2SQL consta de tres casos de uso, los cuales definen a un nivel de detalle más profundo los diferentes procesos que se llevan a cabo para realizar su objetivo.

Básicamente para llevar a cabo la traducción, el caso de uso se descompone en tres nuevos casos de uso, SW-CU2.1:AnalizarCodigoCobolSQL, SW-CU2.2:GenerarScriptSQL y SW-CU2.3:GenerarInterfazBD (Figura 29).

El primer caso de uso en ejecutarse es el SW-CU2.1, el cual lleva a cabo el análisis del código original escrito en lenguaje COBOL y almacena información pertinente. Posteriormente con la información del primer caso de uso se genera una clase que contiene la estructura y permite la comunicación con la base de datos (SW-CU2.3:GenerarInterfazBD) y finalmente se genera el script con la base de datos en SQL 92 (SW-CU2.2:GenerarScriptSQL).

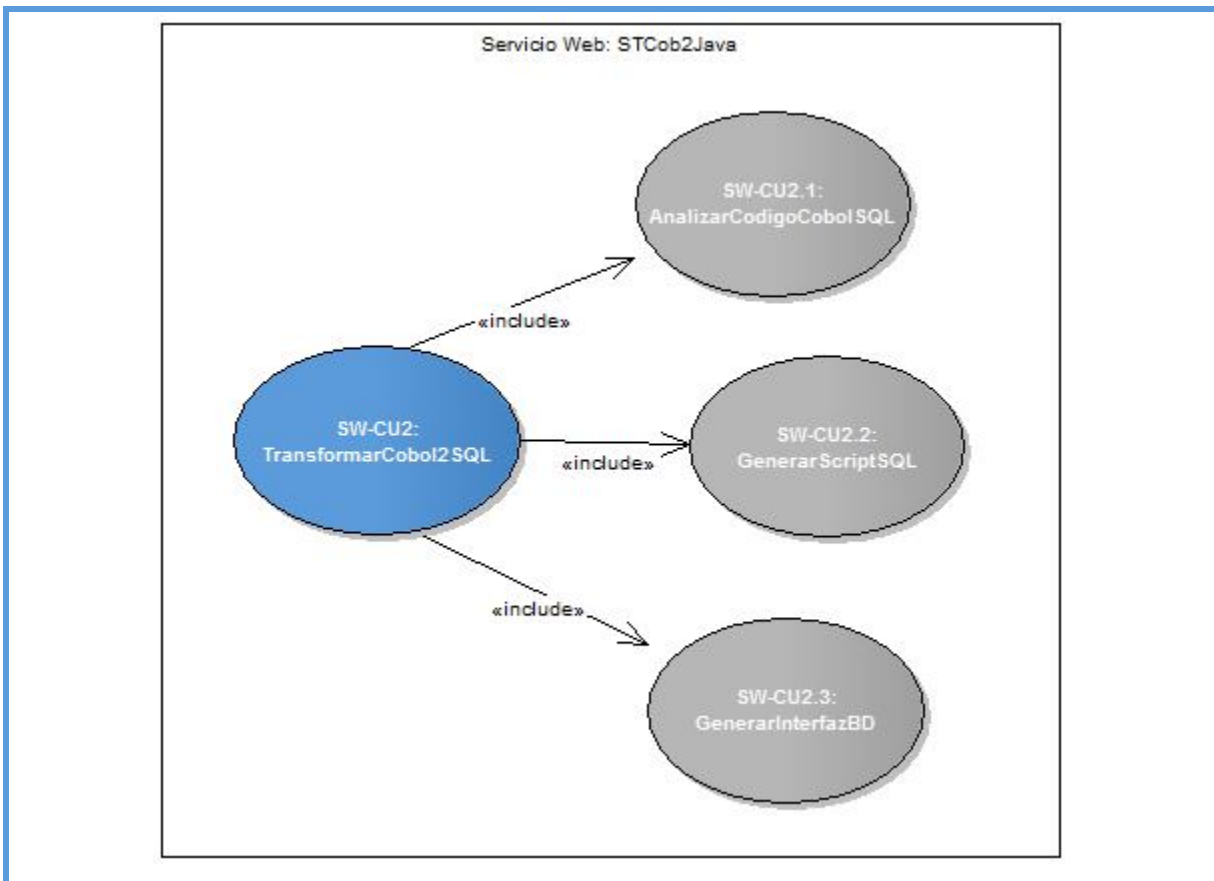


Figura 29. Detalle del caso de uso SW-CU2:TransformarCobol2SQL

El segundo caso de uso de la Figura 28, correspondiente a la traducción de estatutos lógicos incluye al caso de uso SW-CU3.1:AnalizarCodigoCobol el cuál a su vez extiende a los siguientes casos de uso:

- SW-CU3.2:CnvDatosSimples
- SW-CU3.3:CnvEstructuras2Clase
- SW-CU3.4:CnvSentenciasControl
- SW-CU3.5:CnvProcedimientos
- SW-CU3.6:CnvSntAccesoBD

Cada uno de ellos tiene la funcionalidad de convertir los diferentes tipos de instrucciones en COBOL (Ver Figura 30).

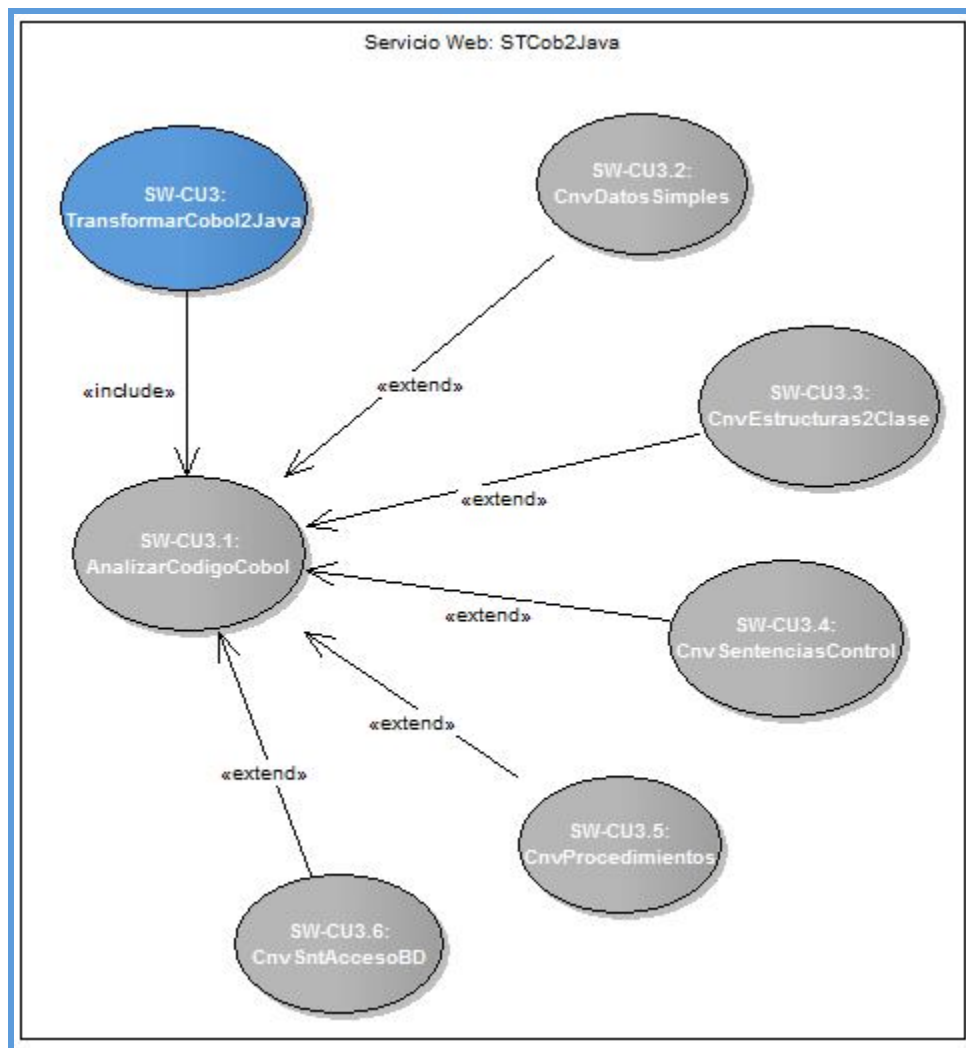


Figura 30. Detalle del caso de uso SW-CU3:TransformarCobol2Java

El detalle de los casos de uso tal como su descripción, objetivo, casos de uso relacionados, requisitos y escenarios son presentados en el Anexo B del presente documento.

4.1.2 Requerimientos de Diseño y Construcción

El diseño del sistema estará basado en el estándar de COBOL 85, el cual no está implementado al 100%. Por lo tanto en el diseño se consideraron posibles extensiones tanto de sentencias no consideradas en el estándar COBOL 85, como a otros dialectos de COBOL que también estén basados en el estándar de COBOL 85 y que no fueron cubiertos en este proyecto de tesis.

De la misma manera el diseño contempla diferentes comportamientos estratégicos para cada sentencia considerada en el analizador.

4.1.3 Requerimientos de Reusabilidad

Debido a que la herramienta STCob2Java fue generada mediante la adaptación de las herramientas de antecedente C2J y Cob2Java para propósitos específicos (traducir de lenguaje Cobol a Java), no se consideraron requerimientos de reusabilidad a nivel de módulos para otros sistemas. Sin embargo, sí se consideraron requerimientos de flexibilidad, para reusar clases y métodos ya implementados en la herramienta C2J y extender su funcionalidad.

4.1.4 Diagramas de Paquetes y Clases

A continuación se presenta el diagrama de paquetes diseñado y cada uno de los diagramas de clases de estos paquetes (Figura 31).

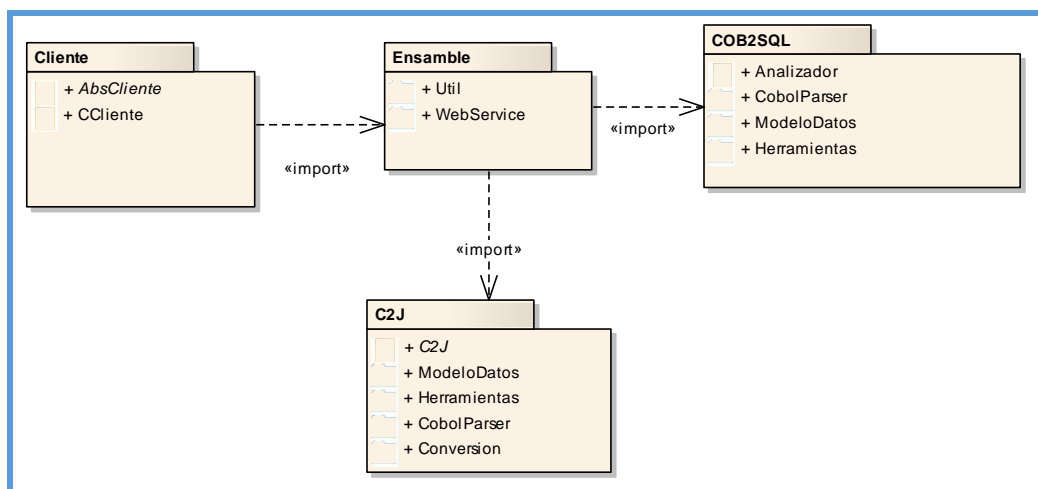


Figura 31. Diagrama general de paquetes del servicio web

Los paquetes presentados en la Figura 31 son descritos a continuación:

Tabla 9. Descripción de paquetes del servicio web

Nombre del paquete	Descripción
Cliente	Representa al cliente del servicio web, éste proporcionará el código en Cobol que será traducido a Java y SQL.
Ensamble	Contiene la clase encargada de unificar los paquetes encargados de realizar las diferentes traducciones de lenguaje COBOL a SQL o Java.
COBOL2SQL	Es el encargado de generar tanto el script de la base de datos como un conjunto de clases que permiten la interfaz entre la aplicación en Java y la base de datos generada por el script.
C2J	Define las clases que permiten realizar la conversión del código COBOL al lenguaje de programación Java.
Herramientas	Esta se encarga de realizar y controlar los procesos de escritura de los nuevos archivos que contienen el código traducido y comprimirlos para generar un solo archivo. Es usado por los paquetes COBOL2SQL y C2J.
ModeloDatos	Contiene las clases que almacenan información acerca de la estructura de las instrucciones de manipulación de los archivos descritas en COBOL, además de la estructura de los registros. Es usado por los paquetes COBOL2SQL y C2J.
CobolParser	Contiene la arquitectura del analizador y clases del manejador de Tokens, recuperador de errores, etc. Los paquetes COBOL2SQL y C2J contienen un archivo con el mismo nombre, pero los contenidos son diferentes ya que ambos provienen de la compilación con javacc de un archivo que contiene la gramática de Cobol y las acciones semánticas. El primero enfocado a acciones semánticas de traducción del sistema de archivos y el segundo a estatutos lógicos.
Conversion (Paquete interno de COBOL2JAVA)	Contiene el conjunto de clases concretas enfocadas a la conversión de cada uno de los verbos de cobol a java. Para el diseño de este paquete fue tomado como base el patrón de diseño “ Strategy ” [11].

En la Figura 32 se muestran las clases que están contenidas en el paquete Ensamble, así como algunos de sus atributos y métodos principales.

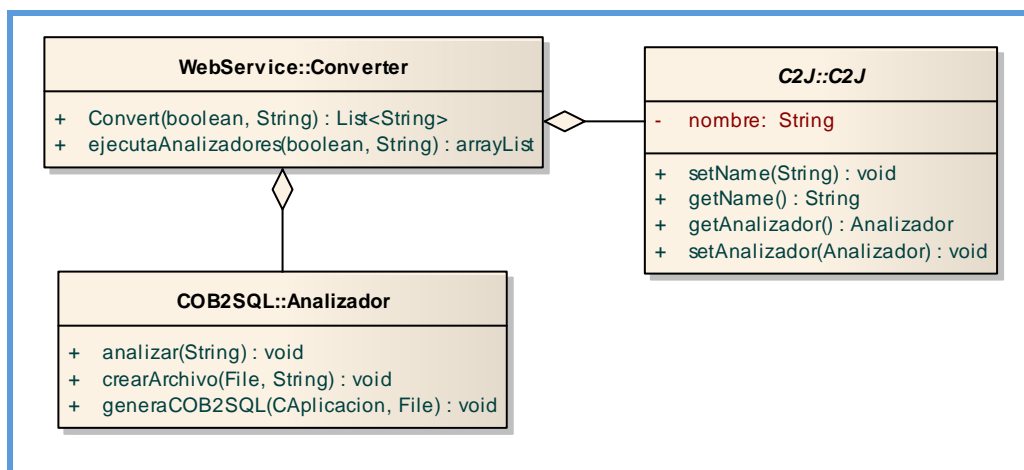


Figura 32. Clases principales del paquete Ensamble

De la misma manera la Figura 33 muestra las clases que permitan almacenar la información referente a la estructura de los archivos definidos en COBOL.

En la Figura 34 se observa el diagrama que corresponde a la transformación de código COBOL a Java. El módulo principal se compone de un analizador, una tabla de símbolos y un conjunto de clases que realizan la conversión de acuerdo al tipo de dato analizado y de acuerdo a un algoritmo previamente diseñado.

Para llevar a cabo el análisis del código, y posteriormente ser traducido al lenguaje Java, se usa la arquitectura de un compilador (Figura 35), el cuál debe contar con módulos de manejo de errores, control de tokens, etc.

La Figura 36 muestra el diagrama de clases “CBDSQL” con el conjunto de clases y la estructura de la nueva base de datos en SQL que es usada por el módulo “COBOL2JAVA”. La nueva estructura representa para el código de la aplicación Java la estructura de archivos que presenta el código COBOL original y permite manejar la base de datos en SQL generada a partir del código COBOL original. Este paquete no se especifica en el diagrama de paquetes del servicio Web, ya que es un paquete generado y devuelto como parte de las clases generadas a partir del código fuente en lenguaje COBOL. Todas las clases son comunes para todos los programas que tengan acceso a archivos, únicamente la clase CConstructorSQL es diferente, ya que mantiene la estructura de las tablas en la base de datos en SQL.

El detalle de la descripción de cada una de las clases, se encuentra en el Anexo C del presente documento de tesis.

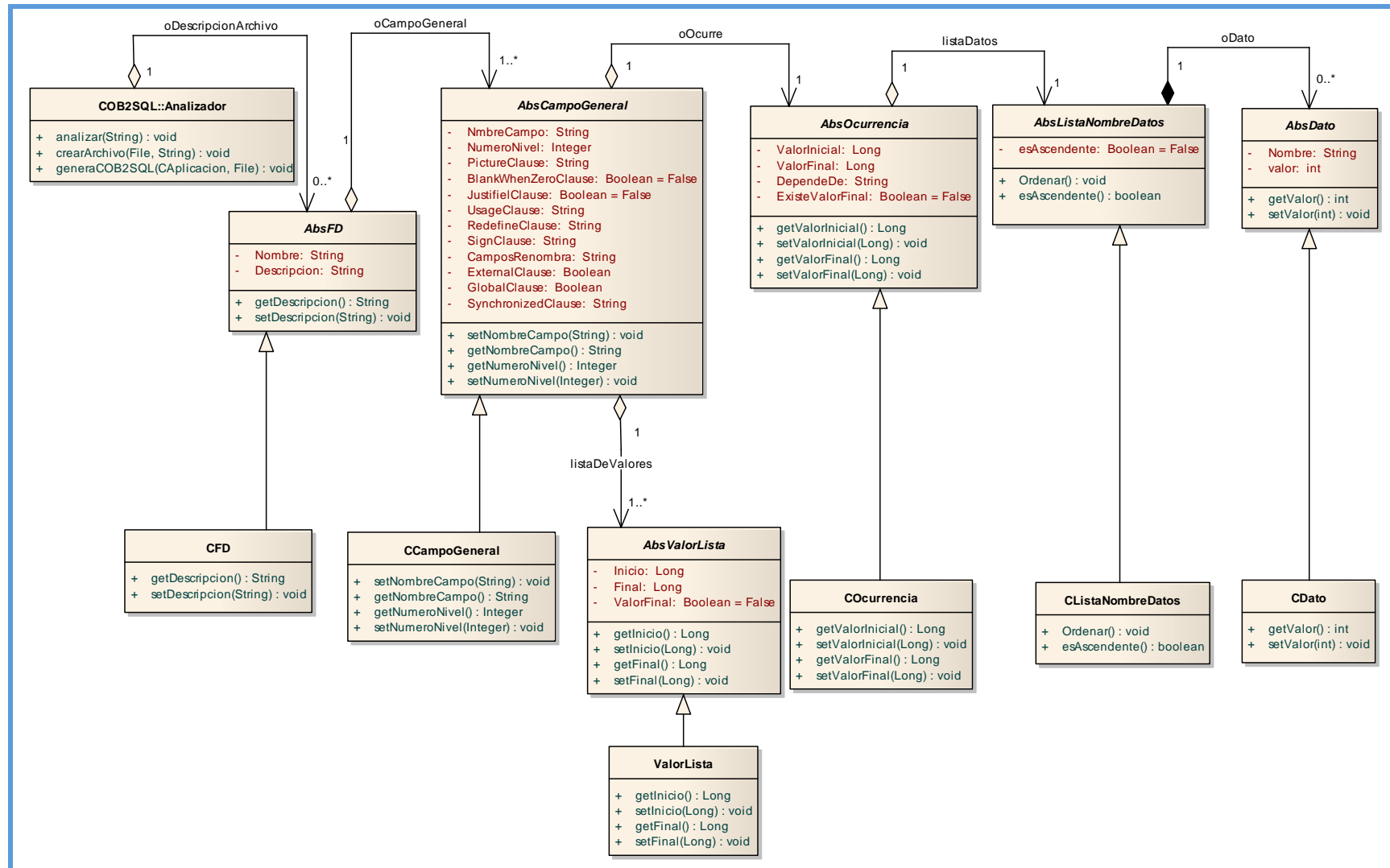


Figura 33. Diagrama de clases del paquete COB2SQL

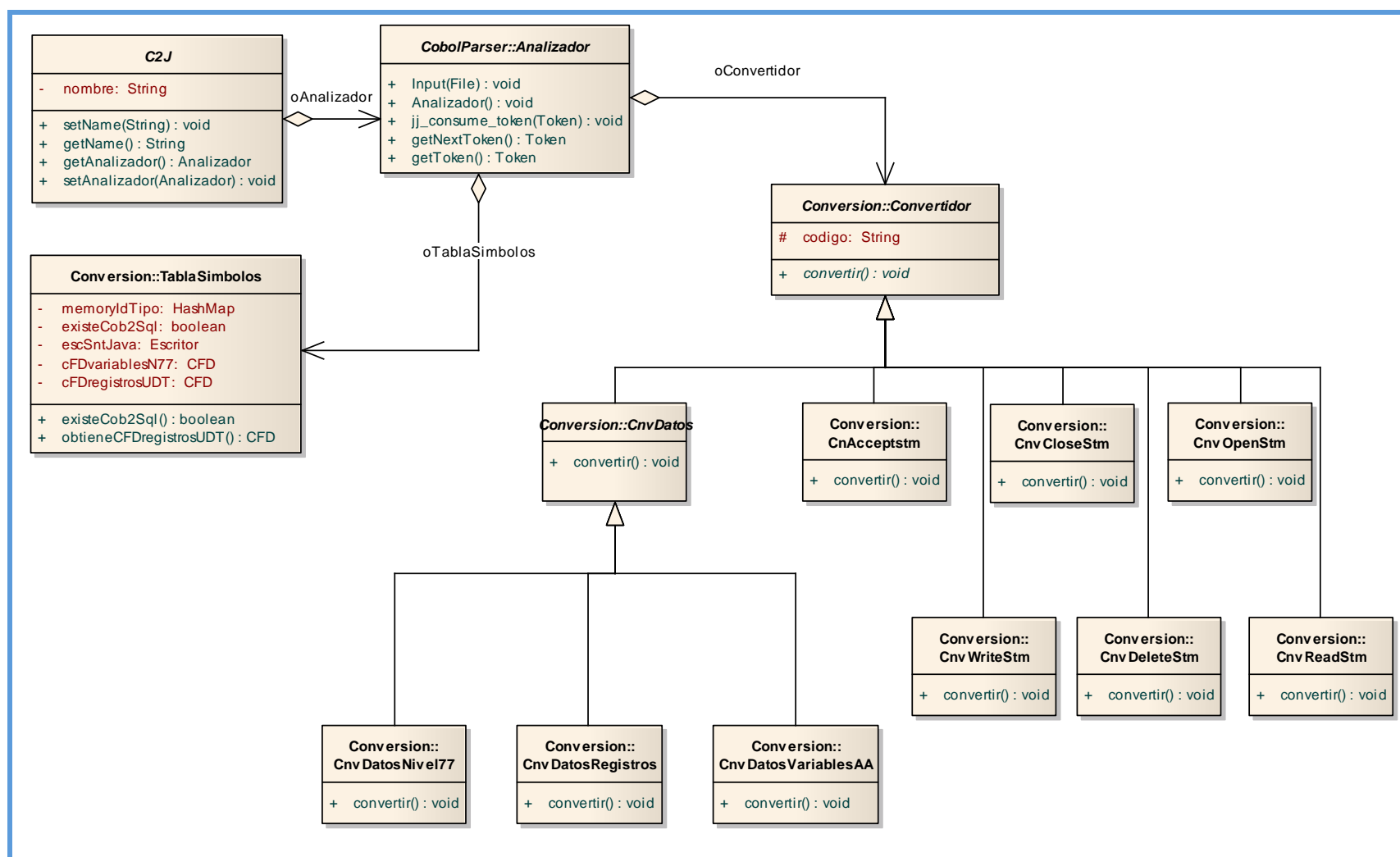


Figura 34. Diagrama de clases del paquete C2J

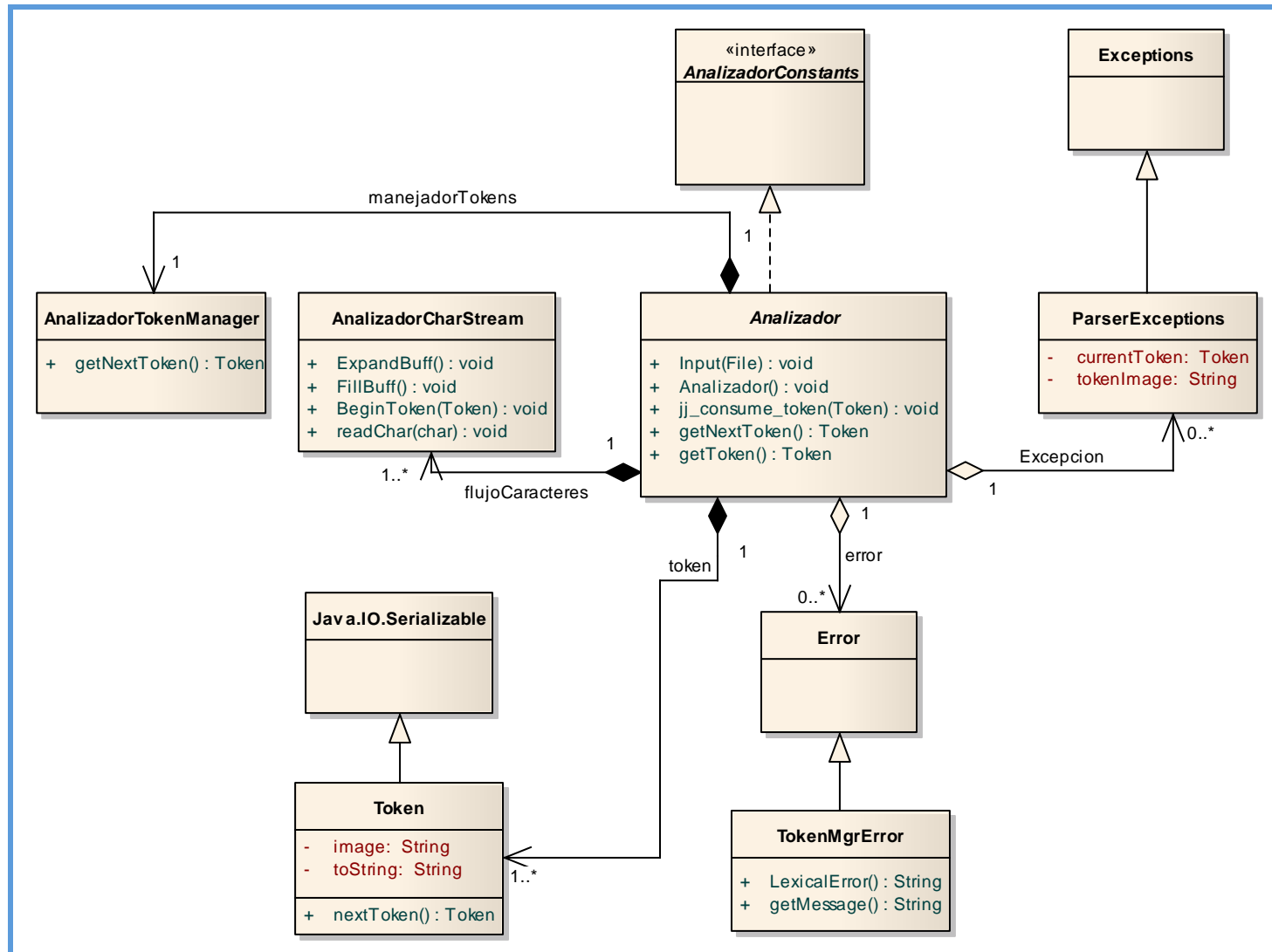


Figura 35. Diagrama de clases del paquete CobolParser

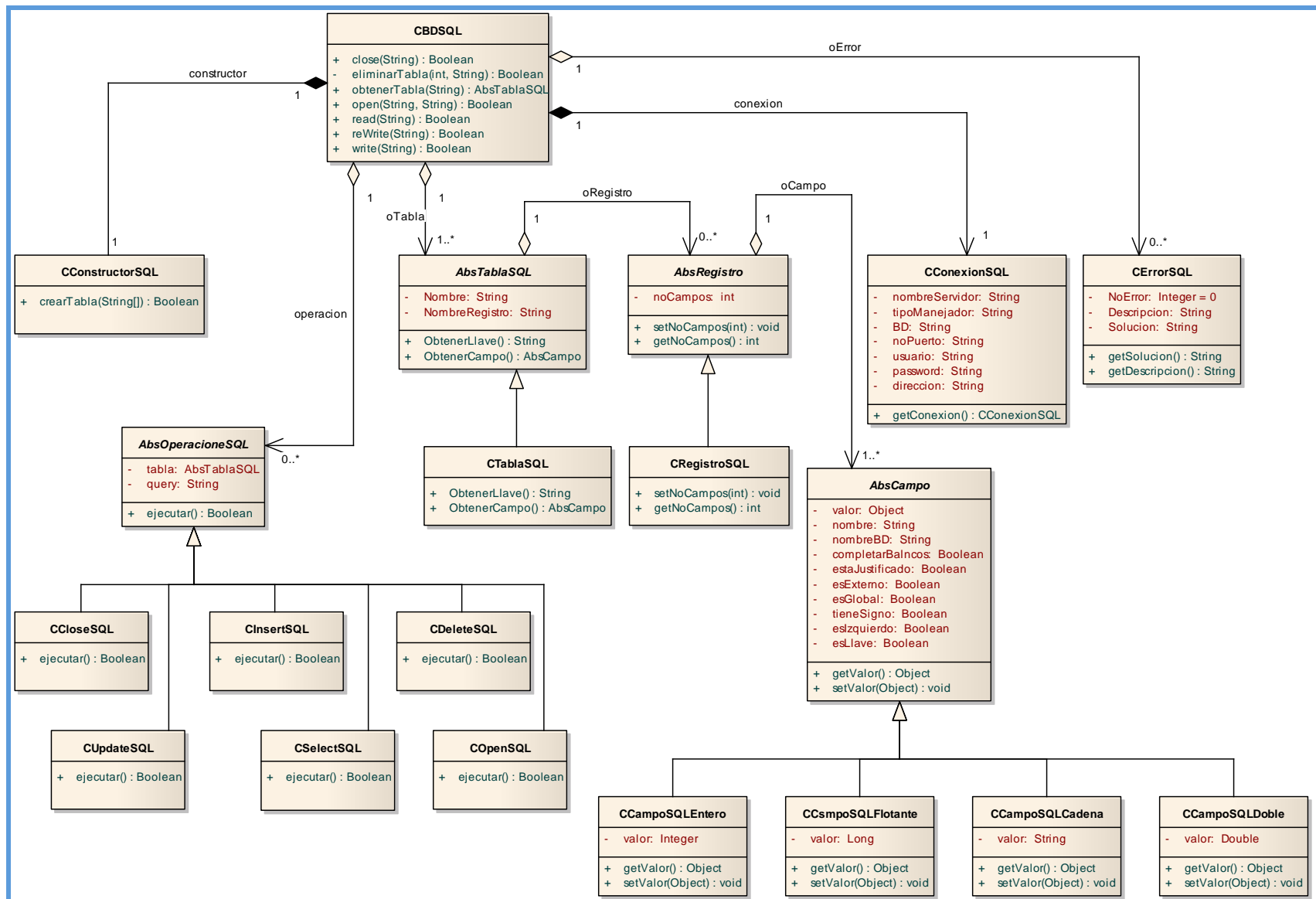


Figura 36. Diagrama de clases del paquete InterfazJava2SQL

4.1.5 Diagrama de Despliegue

Una vez analizadas las clases que participan dentro del Servicio Web, la Figura 37 muestra el diseño del diagrama de despliegue. En ella se muestran los componentes y las clases participantes en el sistema de traducción de COBOL a Java. Por otro lado se muestra el hardware considerado para la implementación.

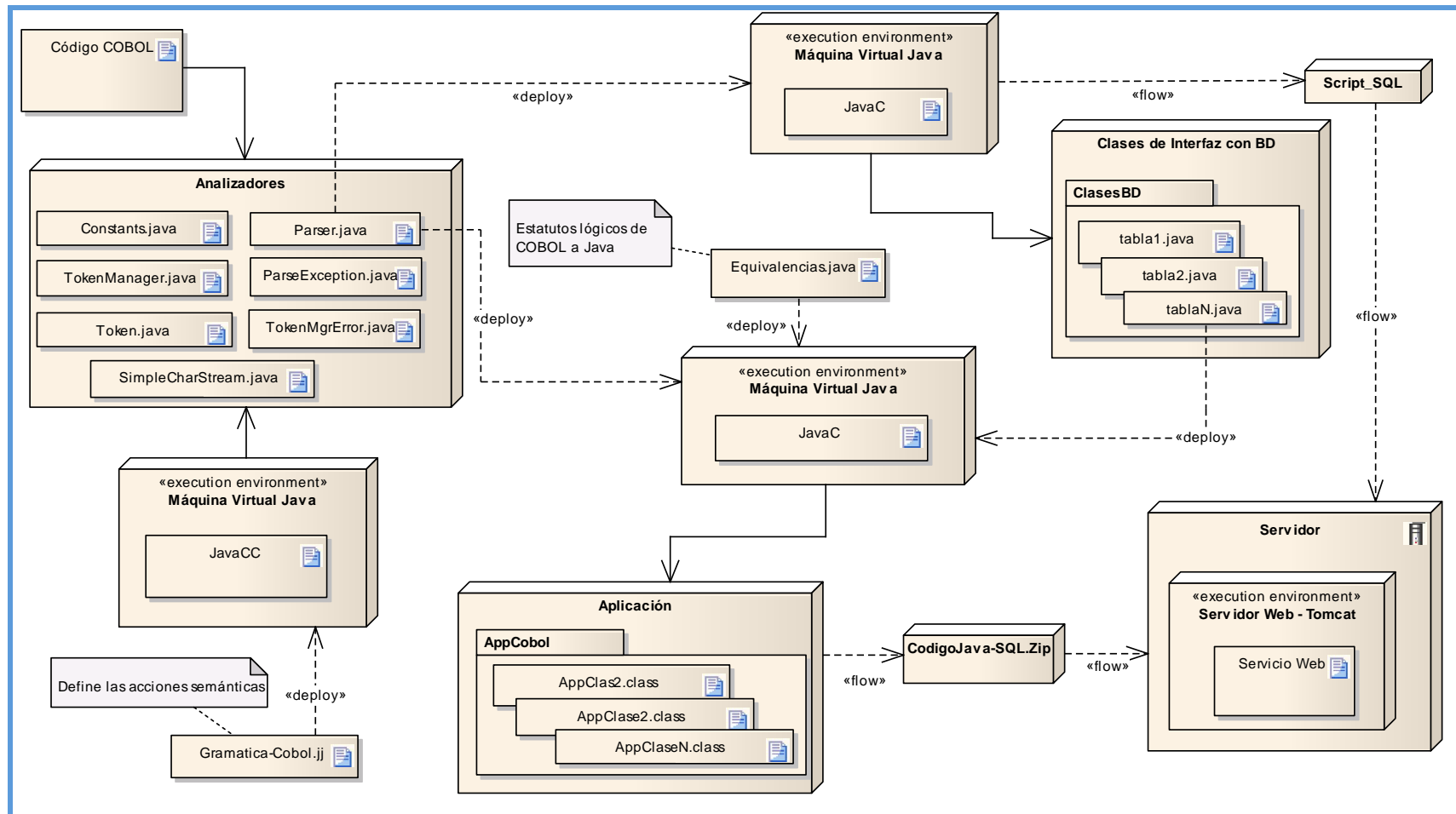


Figura 37. Diagrama de despliegue de la herramienta

4.1.6 Interfaz del Servicio Web

Para acceder al servicio Web de la herramienta STCob2Java es necesario contar con la siguiente información:

- **Dirección IP**

Dirección IP del servidor en el que se encuentra hospedado el servicio Web. Para obtener dicha dirección será necesario contactar al Dr. René Santaolaya Salgado, quien es el director general del proyecto a través del correo electrónico rene@cenidet.edu.mx. A la vuelta del correo se proporcionará la dirección del sitio.

- **Método**

El nombre del método al que se debe llamar para acceder a la herramienta es: **converter**.

- **Entradas**

En la Tabla 10 se muestran el nombre de los mensajes y los tipos de datos que son solicitados por el servicio Web como entrada. Es importante señalar para que el servicio Web funcione correctamente, que los dos parámetros son obligatorios.

Tabla 10. Parámetros de entrada al servicio web

Nombre	Tipo de Datos	Descripción	
CobolCode	Java.lang.String	Contiene el código escrito en lenguaje Cobol que se desea traducir.	
generarTCompleta	Boolean	True	Se hará una traducción completa, es decir, se traducirán a lenguaje Java tanto los Datos de Acceso a Archivos como los Estatutos Lógicos.
		False	Solamente se traducirán los Datos de Acceso a Archivos.

- **Salida**

La salida del servicio Web es un arreglo de cadenas (Strings) que contiene los diferentes archivos resultantes de la transformación del código escrito en Cobol al lenguaje Java.

Tabla 11. Parámetros de salida del servicio web

Tipo de Dato	Descripción	
ArrayList	Cada uno de los elementos de la lista contiene una cadena, en la que los primeros caracteres antes del salto de línea corresponden al nombre del archivo (incluyendo la extensión) y todo lo siguiente corresponde al contenido del archivo.	
	<p>Ejemplo:</p> <p>Cadena: EQUIPO.JAVA\nimport ObjetosEstructuras.*;\nimport COB2SQL.*;\nimport java.io.*;\nimport java.text.NumberFormat;\n...</p> <div> <div></div> <div>Nombre del Archivo</div> <div></div> <div>Contenido del archivo</div> </div>	

4.2 Análisis y Diseño del Cliente del Servicio Web

4.2.1 Requerimientos Funcionales: Modelo de Casos de Uso

El usuario podrá tener acceso a través de una aplicación cliente, la cuál es una aplicación Web. Esta aplicación cuenta con las siguientes funcionalidades:

1. **CL_STCob2Java_RF_01.** Esta funcionalidad lleva el control de los usuarios autorizados para realizar la traducción de los programas hechos en COBOL.
2. **CL_STCob2Java_RF_02.** Esta funcionalidad permite al usuario salir del sistema, no permitiendo el envío de archivos para traducir.
3. **CL_STCob2Java_RF_03.** Esta funcionalidad permite cargar el archivo a traducir, además proporciona la interfaz al usuario para seleccionar entre realizar una conversión completa o únicamente la traducción de instrucciones de acceso a archivo del programa escrito en COBOL.
4. **CL_STCob2Java_RF_04.** Esta funcionalidad permite traducir el archivo de Cobol a Java, captura el archivo escrito en COBOL y lo envía al Servicio Web para la traducción en su correspondiente código Java.
5. **CL_STCob2Java_RF_05.** Esta funcionalidad comprime los archivos generados por el Servicio Web, obteniendo un solo archivo para su descarga.
6. **CL_STCob2Java_RF_06.** Esta funcionalidad muestra toda la información correspondiente al desarrollo del sistema como:
 - Características
 - Autores
 - Ligas a los manuales disponibles
 - Etc.

A continuación se describe el análisis de casos de uso para llevar a cabo el desarrollo de la aplicación cliente. En la Tabla 12 se presentan los actores identificados, responsabilidades y privilegios.

Tabla 12. Diccionario de actores involucrados

Diccionario de Actores			
Actor	Descripción	Responsabilidad	Privilegios
Usuario	Es cualquier persona que usa el cliente Web para llevar a cabo la traducción.	Proporcionar al sistema un usuario y contraseña para acceder a la aplicación. Seleccionar un archivo escrito en Cobol. Elegir la opción de traducción completa o del sistema de archivos.	Puede descargar archivos traducidos a Java a partir de código COBOL.
SW-STCob2Java	Servicio Web encargado de la traducción de Cobol a lenguaje Java.	Llevar a cabo la traducción de Cobol a Java	-----

Los casos de uso identificados para el Cliente del Servicio Web son los siguientes:

- **CL-CU1:IngresarSistema**
- **CL-CU2:SalirSistema**
- **CL-CU3:CargarArchivo**
- **CL-CU4:ConvertirArchivo**
- **CL-CU5:ComprimirArchivos**

La Figura 38 muestra la interacción entre los actores y casos de uso definidos para el cliente. El caso de uso CL-CU4:ConvertirArchivo es el encargado de realizar la llamada al servicio web. Internamente el cliente ejecuta el caso de uso SW-CU1:ConvertirCobol2J de la Figura 28.

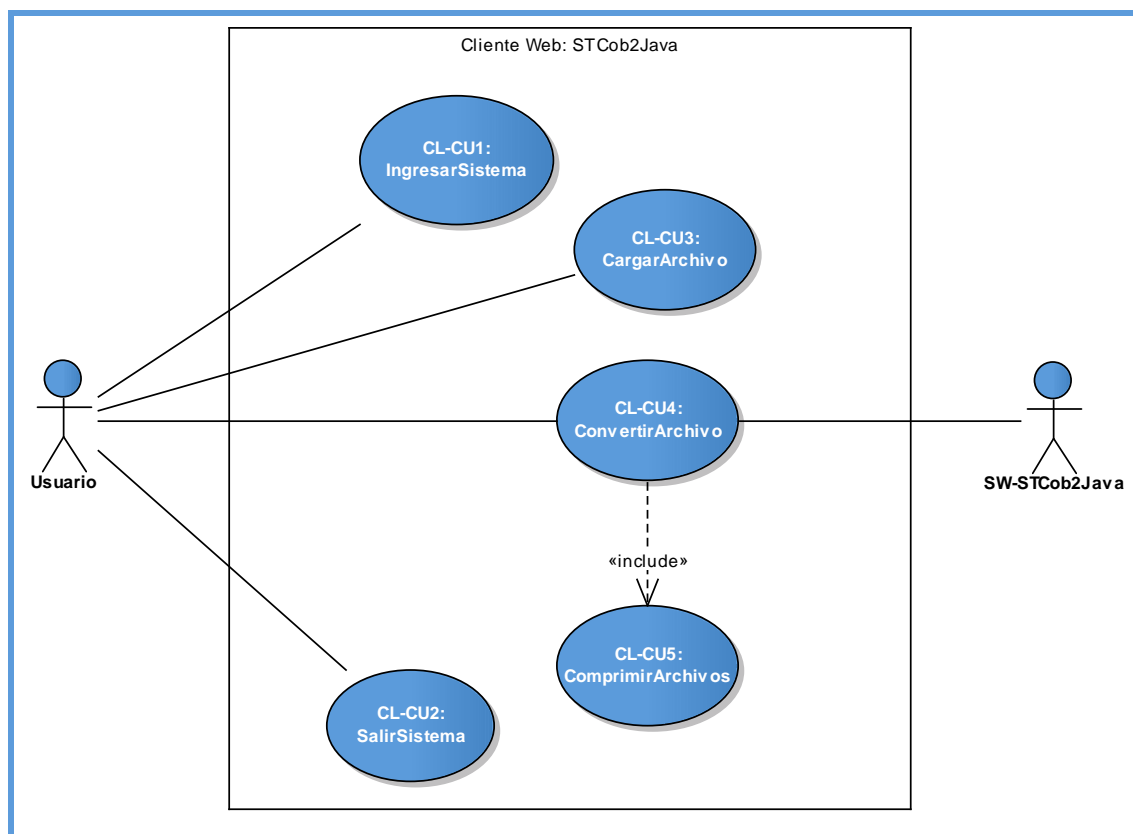


Figura 38. Diagrama general de casos de uso del cliente web: STCob2Java

4.2.2 Diagrama de Paquetes y Clases

La aplicación Web, básicamente cuenta con tres paquetes: *Servlet*, el cual contiene las clases involucradas con el manejo de la interfaz al usuario, *WebService*, contiene las clases involucradas con el llamado al servicio Web y el paquete *Util*, que contiene clases auxiliares para el procesamiento de la información. La Figura 39 muestra el diagrama general de paquetes.

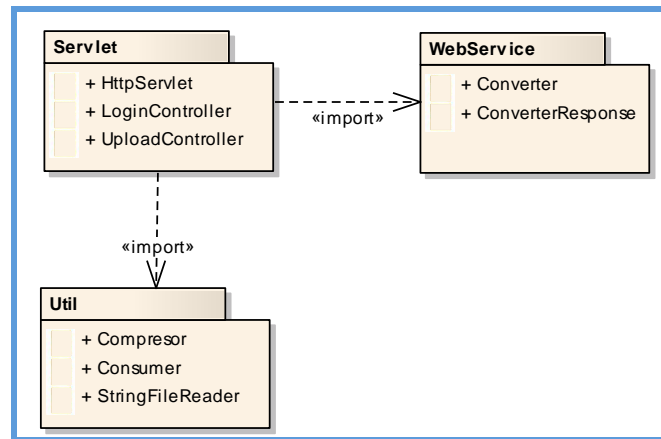


Figura 39. Diagrama de paquetes de la aplicación cliente

Debido a que la interfaz al usuario de la aplicación Web no es compleja, y no requiere de procesamiento complejo (el procesamiento de la traducción lo realiza el Servicio Web), se eligió el uso de Servlets, por ser una de las tecnologías mas sencillas para la generación de páginas dinámicas.

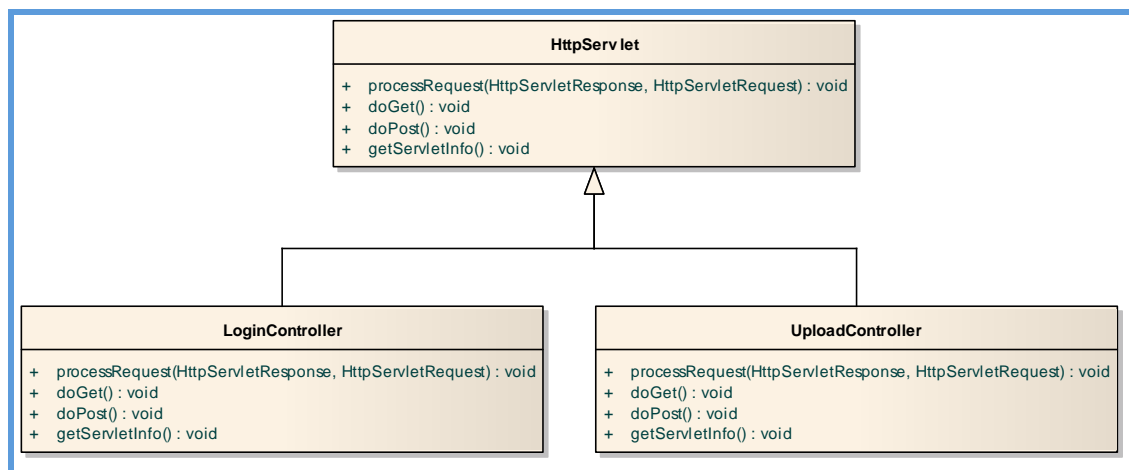


Figura 40. Diagrama de Clases del paquete Servlet de la aplicación cliente

La Figura 40 muestra las clases del paquete *Servlet*. La clase *Logincontroller* lleva el control de acceso a la aplicación, valida que el usuario y contraseña proporcionados por el usuario sean válidos. Por su parte, la clase *UploadController*, maneja el procesamiento del archivo que contiene el código COBOL.

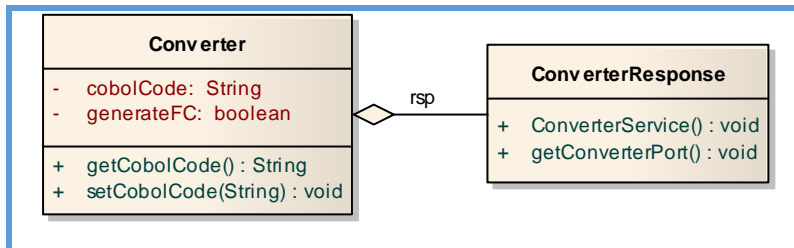


Figura 41. Diagrama de Clases del paquete WebService de la aplicación cliente

La clase *Converter* es una interfaz al Servicio Web (Figura 41).

La Figura 42 muestra las clases involucradas en el procesamiento de la información retornada por el servicio y también el llamado al servicio Web.

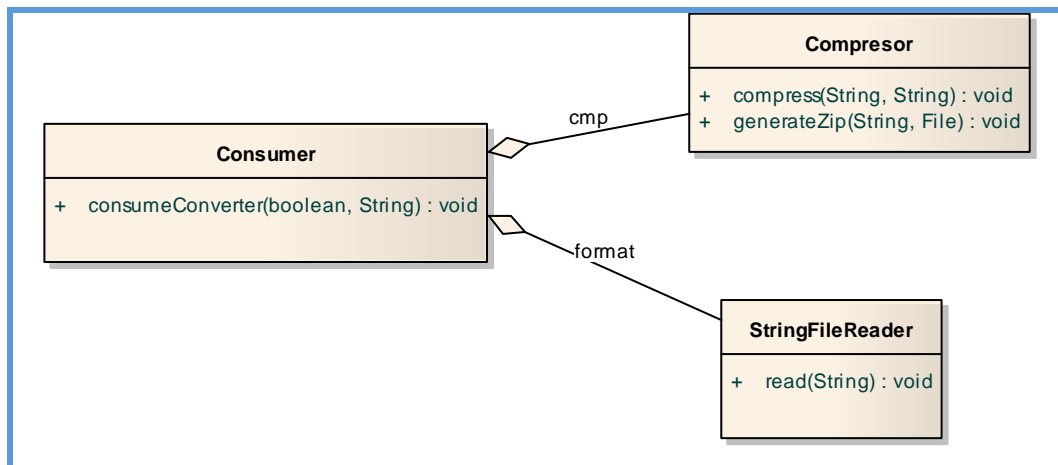


Figura 42. Diagrama de Clases del paquete Util de la aplicación cliente

4.2.3 Diagrama de Despliegue

Para fines de mantenimiento de la herramienta, se presenta el Diagrama de Despliegue en UML, en donde se indican los diferentes componentes involucrados y la interacción entre ellos.

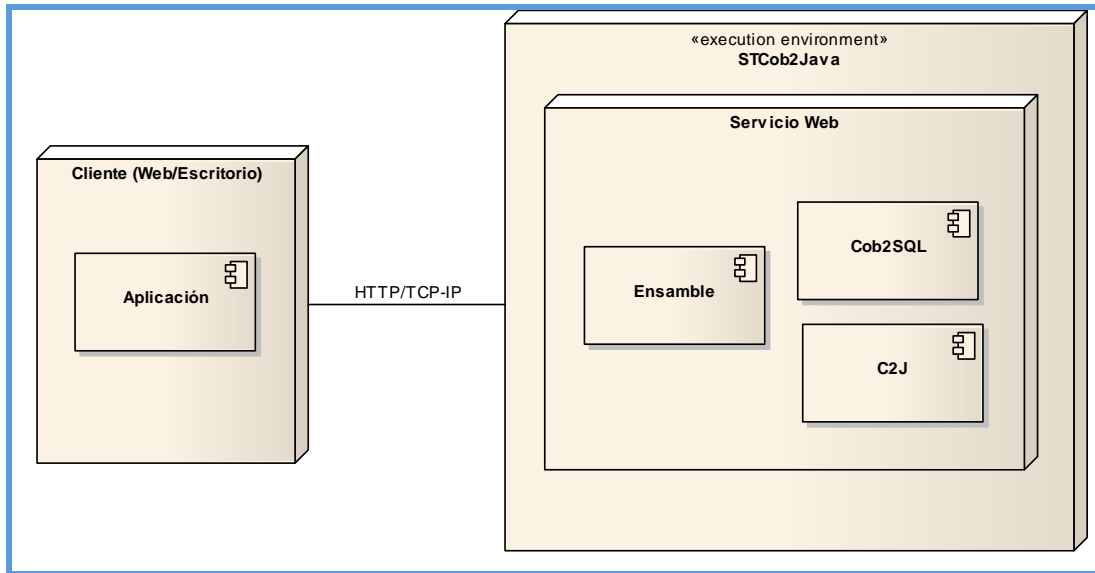


Figura 43. Diagrama de despliegue del STCob2Java.

4.2.4 Interfaz del Cliente Web

A continuación se proporciona un ejemplo general del uso de la herramienta STCob2Java. La aplicación Web consta de diversas páginas, las cuales contienen información general de la aplicación y una página donde permite la traducción de código en Cobol al lenguaje de programación Java, haciendo uso del servicio Web implementado. Una vez que la aplicación cliente recibe del usuario la información solicitada y es enviada al servicio Web, se procesa dicha información para generar una serie de archivos, agrupándolos dentro de un solo archivo con extensión .zip.

- **Acceso a la Aplicación Cliente**

Para acceder a la aplicación cliente es necesario entrar al sitio Web de STCob2Java, esto se hace mediante un navegador de internet, se recomienda el uso del navegador **Mozilla Firefox**. Una vez introducida la dirección del sitio Web en el navegador, se abrirá la página para ingresar a la aplicación Web STCob2Java, como se muestra en la Figura 44.



Figura 44. Página de ingreso a la aplicación web STCob2Java

- **Páginas de Información**

STCob2Java cuenta con un menú de cinco opciones, de la cuales cuatro proporcionan información general del sistema como:

- **Inicio.** Descripción general.
- **Características de la herramienta.** Requerimientos a los que atiende.
- **Autores.** Involucrados en el proyecto
- **Manuales.** Manuales disponibles para la aplicación.

- **Página de la Aplicación**

Antes de comenzar a utilizar STCob2Java se requiere de la autenticación del usuario que solicita usar esta aplicación. El usuario necesita tener la información necesaria para su autenticación, es decir, el nombre de usuario y su contraseña. En la Figura 45 se muestra la pantalla de captura de dichos datos.



Figura 45. Página de acceso a la aplicación STCob2Java

Una vez que el usuario ingresa a la aplicación, para efectos de prueba el usuario es “alba” y la contraseña es “albaquino08c”, éste deberá proporcionar el archivo con el código fuente en Cobol a ser convertido. En la Figura 46 y la Figura 47 se muestran las pantallas para realizar la carga de dicho archivo. Cabe mencionar que STCob2Java asume que el código en Cobol del archivo proporcionado, es ejecutable en algún ambiente de desarrollo de Cobol 85 y que no tiene estatutos fuera del estándar COBOL 85.

Además, el usuario podrá elegir convertir sólo la sección de acceso a archivos o la conversión completa (sistema de archivos y estatutos lógicos).

Para terminar el proceso de conversión, el usuario deberá pulsar el botón “Enviar Archivo” para invocar al servicio Web y realizar la conversión.

En la Figura 48 se muestra la pantalla final antes de invocar al servicio Web.



Figura 46. Cargar el archivo a analizar para la aplicación STCob2Java

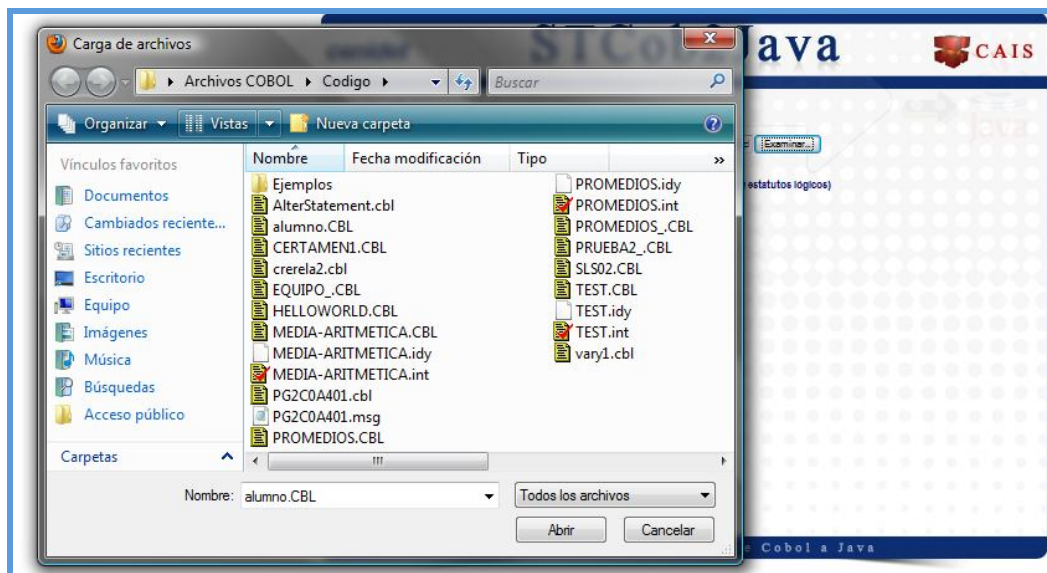


Figura 47. Navegación de directorios para buscar el archivo a analizar



Figura 48. Datos completos para la aplicación STCob2Java

STCob2Java crea un archivo con extensión .zip que contiene los archivos generados por el proceso de conversión.

En la Figura 49 se muestra la página de resultados de la aplicación. En dicha página se enlistan archivos y directorios generados por la aplicación. Para ver y manipular los archivos generados, se deben descargar en el equipo local.



Figura 49. Resultado de la ejecución de la aplicación STCob2Java

Capítulo 5. Diseño Experimental

El presente capítulo presenta el plan de pruebas integral programado para evaluar el funcionamiento del sistema desarrollado en el proyecto de tesis. El plan de pruebas contempla el diseño de pruebas de integración y funcionales involucrando a las dos herramientas generadas, el servicio Web de traducción y el cliente Web de acceso.

Además del plan de pruebas se presentan los resultados de la ejecución del mismo y un análisis de dichos resultados.

5.1 Plan de Pruebas

5.1.1 Elementos de Prueba

Los elementos de prueba que serán verificados para comprobar su correcto funcionamiento son los requerimientos funcionales definidos tanto para el servicio Web en específico, como la herramienta cliente del servicio Web, además de los casos de uso contemplados en ambas herramientas.

En la Tabla 13 se presentan todos los requerimientos funcionales evaluados para el plan de pruebas, así como todos los casos de uso considerados (Tabla 14).

Tabla 13. Elementos de prueba : Requerimientos funcionales

Requerimientos Funcionales	
Identificador	Descripción
SW_STCob2Java_RF_01	El servicio deberá permitir al cliente las opciones de realizar la conversión completa y la traducción únicamente de instrucciones de acceso a archivo del programa escrito en COBOL.
SW_STCob2Java_RF_02	El sistema deberá generar un script en lenguaje SQL estándar 92 que contenga el esquema de la base de datos y pueda ser ejecutado en algún manejador de Base de Datos que soporte en estándar SQL 92.
SW_STCob2Java_RF_03	El sistema debe ser capaz de generar un conjunto de clases de objetos que permitan realizar la comunicación entre aplicaciones en Java y la base de datos a partir de la definición de archivos en el programa COBOL fuente
SW_STCob2Java_RF_04	El sistema deberá realizar la traducción automática de estructuras de datos en lenguaje de programación COBOL a una serie de clases abstractas y derivadas en Java, las cuales contengan los datos y métodos relacionados con la estructura.
SW_STCob2Java_RF_05	El sistema realizará la traducción automática de las diferentes sentencias del lenguaje COBOL a Java, como sentencias de procedimientos, tipos de datos simples, y sentencias de control.
CL_STCob2Java_RF_01	El sistema deberá llevar el control de los usuarios autorizados para realizar la traducción de los programas hechos en COBOL
CL_STCob2Java_RF_02	El sistema permitirá al usuario salir del sistema, no permitiendo el envío de archivos para traducir
CL_STCob2Java_RF_03	El sistema permitirá cargar el archivo a traducir, además de permitir seleccionar al usuario entre realizar una conversión completa y la traducción únicamente de instrucciones de acceso a archivo del programa escrito en COBOL
CL_STCob2Java_RF_04	El sistema permitirá traducir el archivo de Cobol a Java, enviándolo al Servicio Web
CL_STCob2Java_RF_05	El sistema permitirá comprimir los archivos generados por el Servicio Web, obteniendo un solo archivo para su descarga.
CL_STCob2Java_RF_06	El sistema mostrará toda la información correspondiente al desarrollo del sistema como: <ul style="list-style-type: none"> • Características • Autores • Ligas a los manuales disponibles

Tabla 14. Elementos de prueba: Casos de uso

Casos de Uso	
Identificador	Descripción
SW-CU1:ConvertirCobol2J	El objetivo del caso de uso es la traducción de código escrito en COBOL a código Java.
SW-CU2:TransformarCobol2SQL	El objetivo del caso de uso es llevar a cabo la traducción de instrucciones de manipulación de archivos y definiciones de estructuras de archivos escritas en lenguaje del estándar de COBOL 85, hacia instrucciones de manipulación de datos escritas en lenguaje Java y la definición de una base de datos relacional usando el lenguaje SQL 92.
SW-CU3:TransformarCobol2Java	El objetivo del caso de uso es llevar a cabo la traducción de las instrucciones no relacionadas con la manipulación de archivos escrito en COBOL a código escrito en lenguaje Java.
SW-CU2.1:AnalizarCodigoCobolSQL	El objetivo del caso de uso es identificar en el código fuente escrito en COBOL las diferentes instrucciones referentes al manejo del sistema de archivos de la aplicación.
SW-CU2.2:GenerarScriptSQL	El objetivo del caso de uso es generar un script, escrito en el estándar SQL 92, que contenga el esquema de una base de datos equivalente a la declaración y manejo del sistema de archivos en COBOL.
SW-CU2.3:GenerarInterfazBD	El objetivo del caso de uso es generar una serie de clases que permitan la sustitución de instrucciones de acceso a archivos en COBOL a nuevas instrucciones de acceso a Base de Datos en SQL.
SW-CU3.1:AnalizarCodigoCobol	El objetivo del caso de uso es llevar a cabo el análisis de cada una de las sentencias escritas en COBOL para determinar posteriormente qué tipo de

	transformación será aplicada a cada una de ellas.
SW-CU3.2:CnvDatosSimples	El objetivo del caso de uso es llevar a cabo la traducción de datos simples en COBOL, se identifican como datos simples todos aquellos que tengan directamente un dato equivalente en el lenguaje Java.
SW-CU3.3:CnvEstructuras2Clase	El objetivo del caso de uso es transformar las estructuras identificadas en el código COBOL a clases en Java que contengan atributos y métodos de acceso a ellos.
SW-CU3.4:CnvSentenciasControl	El objetivo de este caso de uso es traducir las sentencias de control identificadas dentro del código fuente escrito en COBOL al lenguaje de programación Java.
SW-CU3.5:CnvProcedimientos	El objetivo de este caso de uso es aplicar una serie de pasos para llevar a cabo la traducción de procedimientos en COBOL al lenguaje de programación Java.
SW-CU3.6:CnvSntAccesoBD	El objetivo del caso de uso es llevar a cabo la transformación de las sentencias correspondientes al acceso al sistema de archivos en COBOL por instrucciones en Java de acceso a una Base de Datos.
CL-CU1:IngresarSistema	El objetivo del caso de uso es acceder al sistema por medio de una cuenta de usuario (nombre y contraseña) y permitir al usuario realizar las operaciones de traducción de código COBOL a código Java.
CL-CU2:SalirSistema	El objetivo del caso de uso es cerrar la sesión del usuario dentro del sistema.
CL-CU3:CargarArchivo	El objetivo del caso de uso es subir el archivo con el código fuente a memoria, con el fin de mantener lista la información para ser enviada al servicio Web y ser convertido.
CL-CU4:ConvertirArchivo	El objetivo del caso de uso es llevar a cabo la conversión de los archivos en Cobol a lenguaje Java,
CL-CU5:ComprimirArchivos	El objetivo del caso de uso es generar todos los archivos físicamente a partir de las cadenas devueltas como parámetros del servicio Web, y, posteriormente formar un solo archivo con extensión .zip.

5.1.2 Criterios de Aceptación

Para el diseño de cada uno de los casos de prueba, han sido considerados tanto los requerimientos definidos como los casos de uso identificados. De esta manera, los criterios de aceptación se basan en el cumplimiento exitoso de cada una de las instancias de prueba, para cada caso de prueba.

Se considera la aceptación del sistema si todos los casos de prueba con sus respectivas instancias son ejecutadas exitosamente y obteniendo los resultados esperados.

5.1.3 Criterios de Suspensión

Bajo ninguna circunstancia se deberán suspender las pruebas definidas en el presente plan de pruebas, ya que de estas dependerá la aceptación final del proyecto propuesto.

5.1.4 Actividades de Prueba

Las actividades requeridas para la preparación y ejecución de las pruebas de aceptación son las siguientes:

- Generación del plan de pruebas
- Diseño de casos de prueba
- Preparación del ambiente para las pruebas internas
- Ejecución interna de casos de prueba

- Generación de la bitácora de pruebas
- Generación de resumen final de incidentes

5.1.5 Condiciones de Ambiente

En la Figura 50 se muestra el ambiente necesario para la ejecución de pruebas.

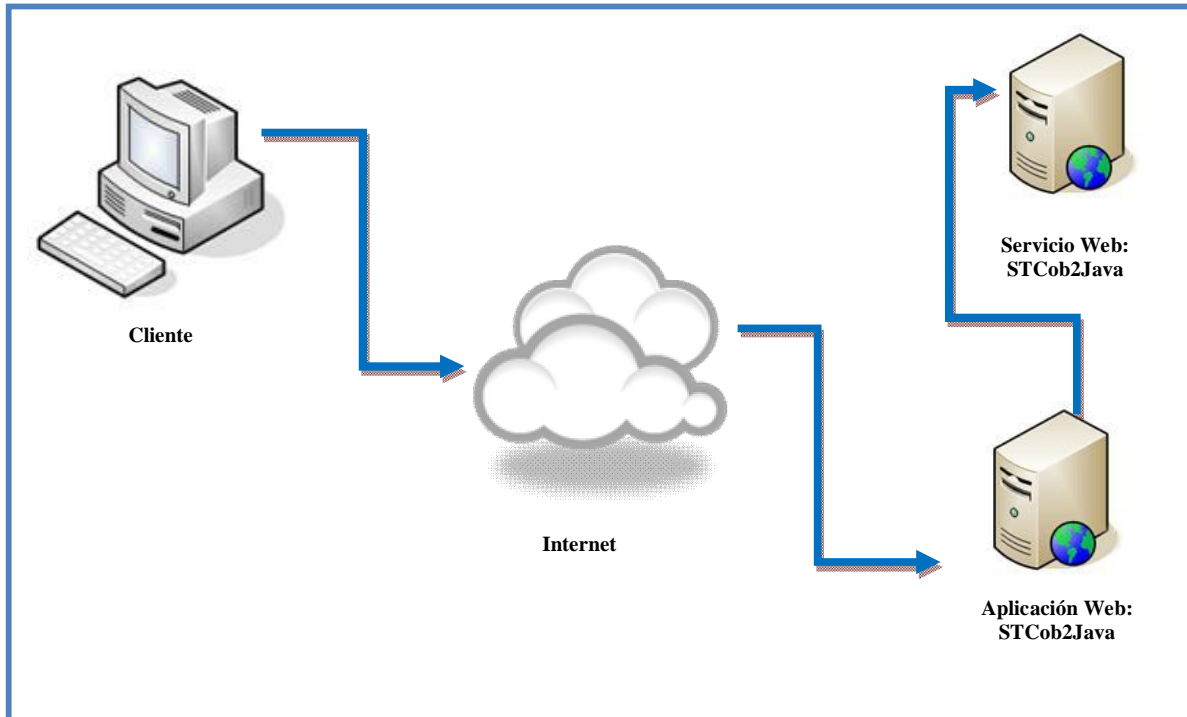


Figura 50. Condiciones del ambiente de pruebas

Por otro lado, es necesario contar con un compilador y ambiente de ejecución del lenguaje de programación Cobol y Java, para comprobar el funcionamiento equivalente de ambos programas. Además de un manejador de base de datos para SQL. A continuación se enlista el software necesario para llevar a cabo la ejecución de las pruebas:

- **Cobol**
Net Express 5.1 de Micro Focus
- **Java**
Máquina Virtual de Java: jdk1.6.0_07
IntelliJ IDEA 8.1.3 (IDE para Java)
- **SQL**
MySQL Server 5.0
Navicat 8.0 MySQL

5.1.6 Personal para Ejecución de Pruebas

Los roles involucrados en la ejecución del presente plan son los siguientes:

- **Evaluador Interno.** Es el responsable de evaluar las pruebas en general y generar el reporte de ejecución con la información generada.
- **Ejecutor.** Es el responsable de llevar a cabo los procedimientos de prueba aplicados a cada una de las instancias de prueba, adicionalmente serán los responsables de llevar a cabo el registro de las instancias de prueba que realice.

Es importante mencionar que todas las personas involucradas deberán tener conocimientos (de medio a avanzado) tanto del lenguaje de programación Cobol, como del lenguaje de programación Java.

5.1.7 Riesgos y contingencias

A continuación se enlistan los riesgos y contingencias para la ejecución del presente plan de pruebas.

- **Riesgo:** No disponibilidad del ambiente para la ejecución de pruebas.
Contingencia 1: Se pospone la ejecución de las pruebas hasta contar con el ambiente requerido.
Contingencia 2: Se ejecuta el plan de pruebas en un ambiente simulado.
- **Riesgo:** El personal disponible para la ejecución de las pruebas no cumple con el perfil y habilidades requeridas.
Contingencia 1: Se pospone la ejecución de las pruebas hasta tener el personal adecuado.
Contingencia 2: Brindar capacitación intensiva al personal disponible bajo el compromiso de ejecutar todas las pruebas requeridas.

5.2 Diseño de las pruebas

Las pruebas al sistema fueron divididas en dos grupos:

- Pruebas de Función
- Pruebas de Integración

Para cada grupo de pruebas se diseñaron un conjunto de casos de prueba, para el caso de las pruebas de integración, la base fueron los casos de uso identificados para el sistema y su

composición. Para el caso de las pruebas de función el enfoque fue basado en los requerimientos funcionales especificados.

5.2.1 Pruebas de Integración

La prueba de integración es el proceso de verificar que los componentes del sistema trabajan juntos conforme a lo descrito en las especificaciones de diseño del programa y del sistema [16].

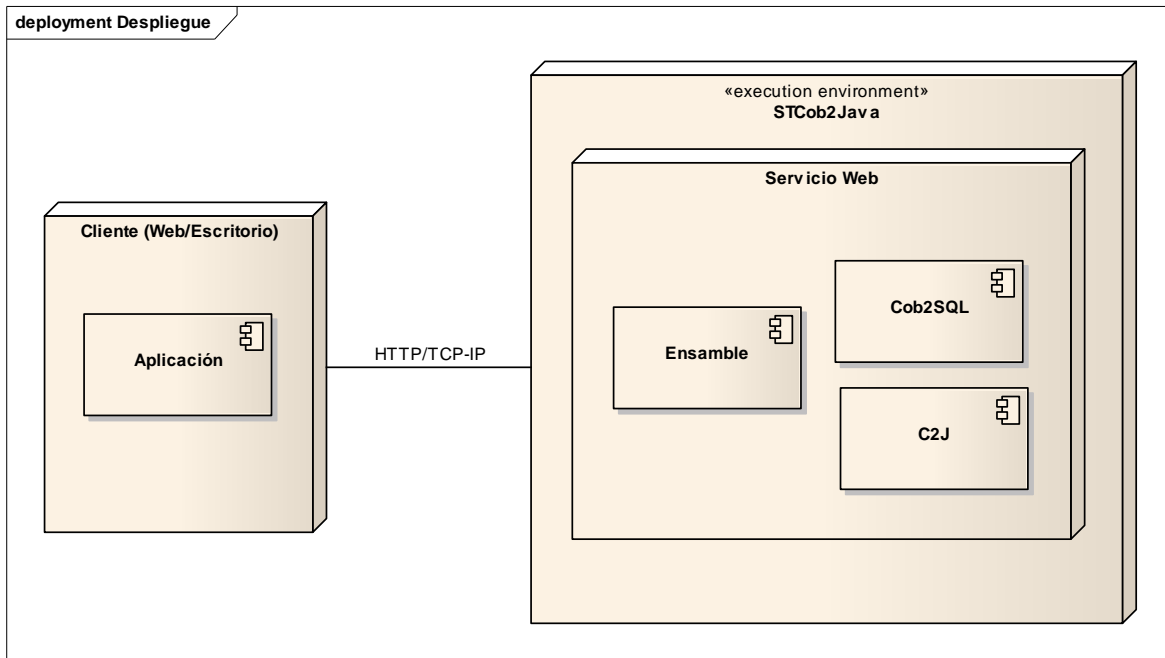


Figura 51. Diagrama de despliegue de las dos aplicaciones

Las pruebas de integración fueron basadas en la Figura 51. La prueba STCob2Java_PI_01 se refiere a la integración del cliente Web con el servicio Web, mientras que las pruebas STCob2Java_PI_02, STCob2Java_PI_03 y STCob2Java_PI_04 muestran la integración de los diferentes módulos dentro del servicio Web.

Nombre de la Prueba:	STCob2Java_PI_01	Autor:	Luisa Alba Aquino B.
Tipo de prueba	Prueba de Integración		
Requerimientos Funcionales:	CL_STCob2Java_RF_03		
Casos de Uso:	CL-CU4:ConvertirArchivo		
Objetivo:	El objetivo del caso de prueba es verificar la integración entre el cliente Web y el Servicio Web.		
Condiciones de Ambiente:	<ul style="list-style-type: none"> El usuario debe estar autenticado en el sistema El servicio Web debe estar en funcionamiento El archivo debe estar cargado en el sistema 		
Entrada :	NA		

Resultados Esperados:	La aplicación cliente muestra una liga para la descarga del archivo generado
Procedimiento de la prueba:	<ol style="list-style-type: none"> 1. El usuario presiona el botón "Convertir" 2. El sistema envía el archivo al Servicio Web 3. El sistema muestra la liga para descargar el archivo generado 4. Fin
Excepciones	
Excepción 1:	<ol style="list-style-type: none"> 3. El servicio Web no se encuentra disponible 4. El sistema muestra el mensaje de error 5. Fin
Observaciones:	NA

Nombre de la Prueba:	STCob2Java_PI_02	Autor:	Luisa Alba Aquino B.
Tipo de prueba	Prueba de Integración		
Requerimientos Funcionales:	<ul style="list-style-type: none">• SW_STCob2Java_RF_01		
Casos de Uso:	<ul style="list-style-type: none">• SW-CU1:ConvertirCobol2J• SW-CU2:TransformarCobol2SQL• SW-CU3:TransformarCobol2Java		
Objetivo:	El objetivo del caso de prueba es verificar que el Servicio Web lleve a cabo la ejecución correcta de sus dos módulos: <ul style="list-style-type: none">• Conversión del sistema de archivos• Conversión de estatutos lógicos		
Condiciones de Ambiente:	La prueba se realiza en un ambiente de desarrollo y pruebas con posibilidad de depurar errores.		
Entrada :	El archivo con el código fuente.		
Resultados Esperados:	Generación de los parámetros de salida.		
Procedimiento de la prueba:	<ol style="list-style-type: none">1. El ejecutor de la prueba ejecuta el servicio Web con condición de paro en el módulo de entrada2. El ejecutor revisa que el programa ejecute el módulo de conversión del sistema de archivos3. El ejecutor revisa que el programa ejecute el módulo de conversión de estatutos lógicos4. El ejecutor revisa la información del parámetro de salida		
Cursos Alternos			
Curso Alterno 1:	NA.		
Excepciones			
Excepción 1:	NA.		
Observaciones:	NA		

Nombre de la Prueba:	STCob2Java_PI_03	Autor:	Luisa Alba Aquino B.
Tipo de prueba	Prueba de Integración		
Casos de Uso:	SW-CU2.1:AnalizarCodigoCobolSQL SW-CU2.2:GenerarScriptSQL SW-CU2.3:GenerarInterfazBD		
Objetivo:	El objetivo de la prueba es verificar la integración de los módulos: análisis del código, generación del script en SQL y generación clases para interfaz con la base de datos dentro de el servicio Web		
Condiciones de Ambiente:	La prueba se realiza en un ambiente de desarrollo y pruebas con posibilidad de depurar errores.		
Entrada :	El archivo con el código fuente.		
Resultados Esperados:	Generación de los archivos correspondientes a los módulos evaluados.		

Procedimiento de la prueba:	<ol style="list-style-type: none"> 1. El ejecutor de la prueba ejecuta el servicio Web con condición de paro en el módulo de conversión del sistema de archivos. 2. El ejecutor revisa que el programa ejecute el módulo de análisis de manera correcta. 3. El ejecutor revisa que el programa ejecute el módulo de generación de script en SQL de forma correcta. 4. El ejecutor revisa que el programa ejecute el módulo de generación de las clases de interfaz con la base de datos. 5. El ejecutor revisa la información del parámetro de salida
Cursos Alternos	
Curso Alterno 1:	<ol style="list-style-type: none"> 2. Durante la ejecución del módulo de análisis se genera un error. 3. El sistema registra el error en la bitácora de error y continua el análisis 4. El ejecutor revisa el archivo de bitácora generado 5. El ejecutor continua con el paso 3 del procedimiento de la prueba
Excepciones	
Excepción 1:	NA.
Observaciones:	NA.

Nombre de la Prueba:	STCob2Java_PI_04	Autor:	Luisa Alba Aquino B.
Tipo de prueba	Prueba de Integración		
Casos de Uso:	SW-CU3.1:AnalizarCodigoCobol SW-CU3.2:CnvDatosSimples SW-CU3.3:CnvEstructuras2Clase SW-CU3.4:CnvSentenciasControl SW-CU3.5:CnvProcedimientos SW-CU3.6:CnvSntAccesoBD		
Objetivo:	El objetivo de la prueba es verificación la integración de los diferentes módulos de conversión de estatutos lógicos dentro del servicio Web.		
Condiciones de Ambiente:	La prueba se realiza en un ambiente de desarrollo y pruebas con posibilidad de depurar errores.		
Entrada :	El archivo con el código fuente.		
Resultados Esperados:	Generación de los archivos correspondientes a los módulos evaluados.		
Procedimiento de la prueba:	<div><div>1.</div><div>El ejecutor de la prueba ejecuta el servicio Web con condición de paro en el módulo de conversión de estatutos lógicos</div></div> <div><div>2.</div><div>El ejecutor revisa que el programa traduzca las instrucciones referentes a la conversión de datos simples mientras lleva a cabo el análisis.</div></div> <div><div>3.</div><div>El ejecutor revisa que el programa traduzca las instrucciones referentes a la conversión de estructuras a clases mientras lleva a cabo el análisis.</div></div> <div><div>4.</div><div>El ejecutor revisa que el programa traduzca las instrucciones referentes a la conversión de estructuras de control mientras lleva a cabo el análisis.</div></div> <div><div>5.</div><div>El ejecutor revisa que el programa traduzca las instrucciones referentes a la conversión de procedimientos o párrafos mientras lleva a cabo el análisis.</div></div> <div><div>6.</div><div>El ejecutor revisa que el programa traduzca las instrucciones referentes a la conversión de estatutos de acceso a archivos mientras lleva a cabo el análisis.</div></div> <div><div>7.</div><div>El ejecutor revisa la información del parámetro de salida</div></div>		
Cursos Alternos			
Curso Alterno 1:	<div><div>2.</div><div>El programa encuentra instrucciones no identificadas durante el análisis</div></div> <div><div>3.</div><div>El programa escribe en bitácora el mensaje de error</div></div> <div><div>4.</div><div>El ejecutor permite que el programa continúe con el análisis</div></div>		
Excepciones			
Excepción 1:	NA.		
Observaciones:	NA.		

5.2.2 Pruebas de Función

Una prueba funcional evalúa el sistema para determinar si las funciones descritas por la especificación de requerimientos son realmente ejecutadas por el sistema integrado [16].

Se diseñaron ocho pruebas de función atendiendo a los requerimientos funcionales definidos para la aplicación. El número de casos de prueba fue tan extenso como fue posible, es decir, por ejemplo para el requerimiento funcional SW_STCob2Java_RF_05, el cual se refiere a la conversión de verbos se realizó al menos un caso de prueba por cada verbo. Para otros casos, el número de casos de prueba por instrucción fue mayor a uno ya que el verbo puede tener diversas opciones alternativas.

Nombre de la Prueba:	STCob2Java_PF_01	Autor:	Luisa Alba Aquino B.
Tipo de prueba:	Prueba de Función		
Requerimientos Funcionales:	CL_STCob2Java_RF_01		
Objetivo:	El objetivo del caso de prueba es comprobar que el sistema permita el acceso solamente a usuarios registrados.		
Condiciones de Ambiente:	El sistema se encuentra disponible en la URL correspondiente.		
Entrada :	Nombre de usuario y contraseña.		
Resultados Esperados:	El ejecutor ingresa al sistema si cuenta con un usuario y contraseña válidos, en otro caso, el sistema no le permite ingresar.		
Procedimiento de la prueba:	1. Inicio 2. El ejecutor ingresa la URL del sistema en el navegador Web. 3. El ejecutor ingresa el nombre del usuario y contraseña del sistema. 4. El sistema muestra la interfaz principal de la herramienta. 5. Fin.		
Cursos Alternos			
Curso Alterno 1:	1. El usuario y/o la contraseña ingresados no son válidos. 2. El sistema muestra mensaje de datos incorrectos. 3. El ejecutor ingresa nuevamente los datos. 4. El sistema muestra la interfaz principal de la herramienta. 5. Fin.		
Curso Alterno 2:	4. Los datos para ingresar al sistema están incompletos. 5. El sistema muestra mensaje de datos incompletos. 6. El ejecutor ingresa los datos correspondientes. 7. El sistema muestra la interfaz principal de la herramienta. 8. Fin		
Excepciones			
Excepción 1:	Si el ejecutor decide no volver a intentar ingresar al sistema los datos, después de un mensaje de datos incorrectos/incompletos, el caso de prueba termina.		
Observaciones:	NA		

Nombre de la Prueba:	STCob2Java_PF_02	Autor:	Luisa Alba Aquino B.
Tipo de prueba	Prueba de Función		
Requerimientos Funcionales:	CL_STCob2Java_RF_02		
Objetivo:	Verificar que el sistema realiza correctamente el cierre de sesión de los usuarios.		
Condiciones de Ambiente:	El ejecutor deberá estar autenticado en el sistema.		
Entrada :	NA.		
Resultados Esperados:	El sistema cancela las operaciones pendientes y termina la sesión del usuario.		

Procedimiento de la prueba:	<ol style="list-style-type: none"> 1. Inicio 2. El ejecutor selecciona la opción "Cerrar Sesión" del sistema. 3. El sistema cierra la sesión. 4. El sistema muestra la interfaz principal del sistema. 5. Fin.
Cursos Alternos	
Curso Alterno 1:	NA
Excepciones	
Excepción 1:	NA
Observaciones:	NA

Nombre de la Prueba:	STCob2Java_PF_03	Autor:	Luisa Alba Aquino B.
Tipo de prueba	Prueba de Función		
Requerimientos Funcionales:	<ul style="list-style-type: none">CL_STCob2Java_RF_03CL_STCob2Java_RF_04CL_STCob2Java_RF_05		
Objetivo:	<p>El objetivo de la prueba es verificar:</p> <ul style="list-style-type: none">El archivo con el código fuente se cargue en el sistemaEl archivo se envíe al servicio Web para su conversiónEl servicio devuelva la informaciónEl sistema comprima la información en un archivoEl sistema muestre una lista con los archivos generadosEl sistema muestre la liga para descargar el archivo		
Condiciones de Ambiente:	El ejecutor deberá estar autenticado en el sistema		
Entrada :	El archivo con el código fuente		
Resultados Esperados:	<ul style="list-style-type: none">El sistema muestra una lista con los archivos generadosEl sistema muestra una liga para descargar el archivo		
Procedimiento de la prueba:	<ol style="list-style-type: none">El usuario selecciona “Cargar Archivo”El sistema muestra una ventana de diálogo para seleccionar el archivoEl usuario selecciona el archivo deseadoEl usuario completa la operación con el botón “Aceptar”El usuario selecciona el botón “Convertir”El sistema muestra una lista con los archivos generadosEl sistema muestra una liga para la descarga del archivoFin		
Cursos Alternos			
Curso Alterno 1:	<ol style="list-style-type: none">El usuario selecciona la opción “Conversión Completa”El usuario selecciona el botón “Convertir”El sistema muestra una lista con los archivos generadosEl sistema muestra una liga para la descarga del archivoFin		
Excepciones			
Excepción 1:	<ol style="list-style-type: none">El usuario no selecciona un archivo para la conversiónEl usuario selecciona la opción “Cancelar “Termina el caso de prueba		
Observaciones:	NA		

Nombre de la Prueba:	STCob2Java_PF_04	Autor:	Luisa Alba Aquino B.
Tipo de prueba	Prueba de Función		
Casos de Uso:	SW-CU3.3:CnvEstructuras2Clase		
Objetivo:	El objetivo de la prueba es comprobar la traducción de las estructuras en COBOL a clases en Java.		
Condiciones de Ambiente:	<ul style="list-style-type: none"> • La prueba se realiza en un ambiente de desarrollo y pruebas con posibilidad de depurar. • El ejecutor ha creado un proyecto con todas las clases generadas en la traducción. • El ejecutor abre el archivo con el código fuente en COBOL en un entorno de desarrollo para COBOL 		

Entrada :	<ul style="list-style-type: none"> El código del programa traducido en Java.
Resultados Esperados:	El código del programa traducido en Java libre de errores, verificando la correspondencia entre estructuras y clases.
Procedimiento de la prueba:	<ol style="list-style-type: none"> El ejecutor copia las clases generadas por el servicio en la carpeta de archivos fuentes del proyecto El ejecutor compara y revisa el código fuente con el código generado, verificando: <ul style="list-style-type: none"> Por cada dato compuesto en COBOL, se generó un archivo con la clase y el nombre del dato Si el dato contiene a su vez datos compuestos, se generan recursivamente archivos con las clases con el dato componente. Si el dato contiene datos primitivos, son agregados a la clase de acuerdo a los tipos de datos equivalentes en Java El ejecutor revisa la generación de los métodos auxiliares para las variables de tipo condicional. El ejecutor compila el proyecto en Java El ejecutor registra la prueba como exitosa Fin de la prueba
Cursos Alternos	
Curso Alterno 1:	<ol style="list-style-type: none"> El ejecutor encuentra inconsistencias entre las estructuras de COBOL y las clases en Java El ejecutor revisa los datos inconsistentes y registra los datos inconsistentes en la bitácora de pruebas El ejecutor regresa al paso 2 del procedimiento general de la prueba
Curso Alterno 2:	<ol style="list-style-type: none"> El ejecutor encuentra que los métodos auxiliares para las variables de tipo condicional no fueron generadas El ejecutor revisa y registra los datos para los cuales no fueron generados los métodos auxiliares El ejecutor regresa al paso 3 del procedimiento general de la prueba
Curso Alterno 3:	<ol style="list-style-type: none"> El ejecutor encuentra que el proyecto contiene errores de compilación (referentes a las clases en Java equivalentes a las estructuras en COBOL) El ejecutor encuentra y registra los errores El ejecutor revisa los errores y los corrige
Excepciones	
Excepción:	El ejecutor decide no corregir los errores encontrados durante la compilación Termina el caso de prueba
Observaciones:	NA

A partir del requerimiento funcional SW_STCob2Java_RF_05 se derivaron los casos de prueba STCob2Java_PF_04 al STCob2Java_PF_07. Se clasificaron los estatutos en estatutos de acceso a archivos, de acceso a base de datos, etc. Con el objetivo de tener un mejor panorama de los casos de prueba.

Nombre de la Prueba:	STCob2Java_PF_05	Autor:	Luisa Alba Aquino B.
Tipo de prueba	Prueba de Función		
Casos de Uso:	SW-CU3.2:CnvDatosSimples		
Objetivo:	El objetivo de la prueba es comprobar la correcta traducción de los datos simples a Java, además de los verbos simples.		
Condiciones de Ambiente:	<ul style="list-style-type: none"> La prueba se realiza en un ambiente de desarrollo y pruebas con posibilidad de depurar errores. El ejecutor ha creado un proyecto con todas las clases generadas en la traducción. El ejecutor abre el archivo con el código fuente en COBOL en un entorno de desarrollo para COBOL. 		
Entrada :	El código del programa traducido en Java.		

Resultados Esperados:	La funcionalidad de los programas escritos tanto en COBOL como en Java es la misma, es decir: <ul style="list-style-type: none">Mensajes en pantalla son los mismosNúmero de IteracionesValor de las variables	
Procedimiento de la prueba:	<ol style="list-style-type: none">El ejecutor copia las clases generadas por el servicio en la carpeta de archivos fuentes del proyectoEl ejecutor compara y revisa el código fuente con el código generado, verificando:<ul style="list-style-type: none">La presencia de una o más instrucciones en el código en Java por cada instrucción en COBOLLa declaración de todas las variables en Java definidas en el código en COBOL correspondientes a tipos de datos simplesSi la variable se encuentra inicializada en el código en COBOL, deberá ser inicializada en el código en JavaEl ejecutor depura ambos programas, validando:<ul style="list-style-type: none">Mensajes en pantallaContenido de las variablesEl ejecutor compila el proyecto en JavaEl ejecutor registra la prueba como exitosaFin de la prueba	
Cursos Alternos		
Curso Alterno 1:	<ol style="list-style-type: none">El ejecutor encuentra que una instrucción en COBOL no se encuentra definida en Java (verbo, dato o inicialización)El ejecutor registra la instrucción en la bitácoraEl ejecutor escribe el código en Java como debió ser traducidoEl ejecutor continúa con el paso 3 del procedimiento general de la prueba	
Curso Alterno 2:	<ol style="list-style-type: none">El ejecutor encuentra inconsistencias en ambas pantallas de los programas (COBOL y Java)El ejecutor registra el error y corrige la informaciónEl ejecutor continúa con el paso 4 del procedimiento general de la prueba	
Curso Alterno 3:	<ol style="list-style-type: none">El ejecutor encuentra que el proyecto contiene errores de compilación (referentes a la traducción de datos y verbos simples COBOL)El ejecutor encuentra y registra los erroresEl ejecutor revisa los errores y los corrige	
Excepciones		
Excepción:	El ejecutor decide no corregir los errores encontrados (instrucciones no definidas, inconsistencias en pantalla o durante la compilación) Termina el caso de prueba	
Items de Prueba		
Instancia:	COBOL	Resultado Esperado (Java)
Tipos de Datos Simples:	<ul style="list-style-type: none">NuméricoAlfanuméricoAlfabético	<ul style="list-style-type: none">StringIntFloatdoble
Verbos:	<ul style="list-style-type: none">AcceptAddAlterComputeDisplayDivideExitInitializeInspectMoveMultiplySetStringSubtract	Conjunto de una o más instrucciones y uno o más métodos equivalentes a los verbos en COBOL
Observaciones:	NA	

Nombre de la Prueba:	STCob2Java_PF_06	Autor:	Luisa Alba Aquino B.
Tipo de prueba	Prueba de Función		
Casos de Uso:	SW-CU3.4:CnvSentenciasControl		
Objetivo:	El objetivo de la prueba es comprobar la traducción de las instrucciones de control, verificando el comportamiento del programa en Java.		
Condiciones de Ambiente:	<ul style="list-style-type: none">La prueba se realiza en un ambiente de desarrollo y pruebas con posibilidad de depurar.El ejecutor ha creado un proyecto con todas las clases generadas en la traducción.En un entorno de desarrollo para COBOL, el ejecutor abre el archivo con el código fuente en COBOL.		
Entrada :	El código del programa traducido en Java.		
Resultados Esperados:	<ul style="list-style-type: none">El comportamiento de ambos programas (en COBOL y Java) es el mismo en los verbos que se refieren al control.Las clases generadas son compiladas y libres de errores para ser ejecutadas por la máquina virtual de Java.		
Procedimiento de la prueba:	<ol style="list-style-type: none">El ejecutor copia las clases generadas por el servicio en la carpeta de archivos fuentes del proyectoEl ejecutor compara y revisa el código fuente con el código generado, verificando:<ul style="list-style-type: none">La presencia de una o más instrucciones en el código en Java por cada instrucción en COBOL referente a verbos de controlEl ejecutor depura ambos programas, validando:<ul style="list-style-type: none">Mensajes en pantallaIteraciones en ambos programasEl ejecutor compila el proyecto en JavaEl ejecutor registra la prueba como exitosaFin de la prueba		
Cursos Alternos			
Curso Alterno 1:	<ol style="list-style-type: none">El ejecutor encuentra que una instrucción en COBOL no se encuentra definida en JavaEl ejecutor registra la instrucción en la bitácoraEl ejecutor escribe el código en Java como debió ser traducidoEl ejecutor continúa con el paso 3 del procedimiento general de la prueba		
Curso Alterno 2:	<ol style="list-style-type: none">El ejecutor encuentra inconsistencias en ambas pantallas de los programas (COBOL y Java)El ejecutor registra el error y corrige la informaciónEl ejecutor continúa con el paso 4 del procedimiento general de la prueba		
Curso Alterno 3:	<ol style="list-style-type: none">El ejecutor encuentra que el proyecto contiene errores de compilación (referentes a la traducción de datos y verbos simples COBOL)El ejecutor encuentra y registra los erroresEl ejecutor revisa los errores y los corrige		
Excepciones			
Excepción:	<ul style="list-style-type: none">El ejecutor decide no corregir los errores encontrados (instrucciones no definidas, inconsistencias en pantalla o durante la compilación)Termina el caso de prueba		
Items de Prueba			
Instancia:	COBOL		Resultado Esperado (Java)
Verbos:	<ul style="list-style-type: none">GotoContinueEvaluateGobackIf-else		<ul style="list-style-type: none">If-elseBreakMétodos
Observaciones:	NA		

Nombre de la Prueba:	STCob2Java_PF_07	Autor:	Luisa Alba Aquino B.
Tipo de prueba	Prueba de Función		
Casos de Uso:	SW-CU3.5:CnvProcedimientos		
Objetivo:	El objetivo de la prueba es comprobar la traducción de los párrafos en COBOL y la instrucción de invocación a ellos.		
Condiciones de Ambiente:	<ul style="list-style-type: none">La prueba se realiza en un ambiente de desarrollo y pruebas con posibilidad de depurar errores.El ejecutor ha creado un proyecto con todas las clases generadas en la traducción.En un entorno de desarrollo para COBOL, el ejecutor abre el archivo con el código fuente en COBOL.		
Entrada :	El código del programa traducido en Java.		
Resultados Esperados:	<ul style="list-style-type: none">Por cada párrafo en COBOL, es definido un método en Java.Por cada llamado a un párrafo en COBOL (perform), existe el llamado correspondiente a un método en Java.Las clases generadas son compiladas y libres de errores para ser ejecutadas por la máquina virtual de Java.		
Procedimiento de la prueba:	<ol style="list-style-type: none">El ejecutor copia las clases generadas por el servicio en la carpeta de archivos fuentes del proyectoEl ejecutor compara y revisa el código fuente con el código generado, verificando que:<ul style="list-style-type: none">Por cada párrafo en COBOL existe un método en JavaPor cada llamado a un párrafo en COBOL (perform), existe el llamado a un método en JavaEl ejecutor depura ambos programas, validando:<ul style="list-style-type: none">Mensajes en pantallaSecuencias de iteración entre párrafos y métodosEl ejecutor compila el proyecto en JavaEl ejecutor registra la prueba como exitosaFin de la prueba		
Cursos Alternos			
Curso Alterno 1:	<ol style="list-style-type: none">El ejecutor encuentra que una instrucción en COBOL no se encuentra definida en JavaEl ejecutor registra el error en la bitácoraEl ejecutor escribe el código en Java como debió ser traducidoEl ejecutor continúa con el paso 3 del procedimiento general de la prueba		
Curso Alterno 2:	<ol style="list-style-type: none">El ejecutor encuentra inconsistencias en ambas pantallas de los programas (COBOL y Java)El ejecutor registra el error y corrige la informaciónEl ejecutor continúa con el paso 4 del procedimiento general de la prueba		
Curso Alterno 3:	<ol style="list-style-type: none">El ejecutor encuentra que el proyecto contiene errores de compilación (referentes a la traducción de datos y verbos simples COBOL)El ejecutor encuentra y registra los erroresEl ejecutor revisa los errores y los corrige		
Excepciones			
Excepción:	<ul style="list-style-type: none">El ejecutor decide no corregir los errores encontrados (instrucciones no definidas, inconsistencias en pantalla o durante la compilación)Termina el caso de prueba		
Ítems de Prueba			
Instancia:	COBOL	Resultado Esperado (Java)	
Verbos:	<ul style="list-style-type: none">Definición de párrafosVerbo Perform	<ul style="list-style-type: none">Declaración y definición de métodosInvocación a métodos	
Observaciones:	NA		

Nombre de la Prueba:	STCob2Java_PF_08	Autor:	Luisa Alba Aquino B.
Tipo de prueba	Prueba de Función		
Casos de Uso:	SW-CU3.6:CnvSntAccesoBD		
Objetivo:	El objetivo de la prueba es comprobar la traducción de las instrucciones de acceso a archivo, verificando el comportamiento del programa en Java.		
Condiciones de Ambiente:	<ul style="list-style-type: none">La prueba se realiza en un ambiente de desarrollo y pruebas con posibilidad de depurar errores.El ejecutor ha creado un proyecto con todas las clases generadas en la traducción.En un entorno de desarrollo para COBOL, el ejecutor abre el archivo con el código fuente en COBOL.		
Entrada :	El código del programa traducido en Java.		
Resultados Esperados:	<ul style="list-style-type: none">El comportamiento de ambos programas (en COBOL y Java) es el mismo, cuando se usan programas con instrucciones de acceso a archivos.Las clases generadas son compiladas y libres de errores para ser ejecutadas por la máquina virtual de Java.		
Procedimiento de la prueba:	<ol style="list-style-type: none">El ejecutor copia las clases generadas por el servicio en la carpeta de archivos fuentes del proyectoEl ejecutor compara y revisa el código fuente con el código generado, verificando:<ul style="list-style-type: none">La presencia de una o más instrucciones en el código en Java por cada instrucción en COBOL referente a verbos de acceso a archivosEl ejecutor depura ambos programas, validando:<ul style="list-style-type: none">Mensajes en pantallaInformación en los archivos de escritura y base de datosEl ejecutor compila el proyecto en JavaEl ejecutor registra la prueba como exitosaFin de la prueba		
Cursos Alternos			
Curso Alterno 1:	<ol style="list-style-type: none">El ejecutor encuentra que una instrucción en COBOL no se encuentra definida en JavaEl ejecutor registra la instrucción en la bitácoraEl ejecutor escribe el código en Java como debió ser traducidoEl ejecutor continúa con el paso 3 del procedimiento general de la prueba		
Curso Alterno 2:	<ol style="list-style-type: none">El ejecutor encuentra inconsistencias en ambas pantallas de los programas (COBOL y Java) y/o inconsistencias en los datos almacenados en los archivos vs base de datosEl ejecutor registra el error y corrige la informaciónEl ejecutor continúa con el paso 4 del procedimiento general de la prueba		
Curso Alterno 3:	<ol style="list-style-type: none">El ejecutor encuentra que el proyecto contiene errores de compilación (referentes a la traducción de datos y verbos simples COBOL)El ejecutor encuentra y registra los erroresEl ejecutor revisa los errores y los corrige		
Excepciones			
Excepción:	<ul style="list-style-type: none">El ejecutor decide no corregir los errores encontrados (instrucciones no definidas, inconsistencias en pantalla o durante la compilación)Termina el caso de prueba		
Items de Prueba			
Instancia:	COBOL	Resultado Esperado (Java)	
Verbos:	<ul style="list-style-type: none">CloseDeleteOpenReadWriteRewrite	Control de acceso a las tablas Conjunto de instrucciones que permitan insertar, borrar, actualizar y consultar la base de datos equivalente al sistema de archivos en SQL.	
Observaciones:	NA		

5.3 Resultados de la Ejecución del Plan de Pruebas

Se llevó a cabo la ejecución tanto de las pruebas de integración como las pruebas de función. Para las pruebas de integración, el enfoque fue que para todos los programas (código fuente en COBOL) la integración entre los módulos y la comunicación entre el cliente Web y el servicio Web se llevara a cabo correctamente.

En las pruebas de función el enfoque se centró en la equivalencia entre instrucciones y/o comparar ambas funcionalidad para determinar la equivalencia entre los programas.

5.3.1 Pruebas de Integración

Se ejecutaron los 4 casos de prueba correspondientes a las pruebas de integración con 10 programas diferentes para verificar la integración entre las aplicaciones implementadas, así como los módulos dentro de ellas. La Tabla 15 muestra el resumen de resultados para las pruebas realizadas.

Tabla 15. Programas evaluados en las pruebas de integración

Programa	Nombre de la Prueba			
	STCob2Java_PI_01	STCob2Java_PI_02	STCob2Java_PI_03	STCob2Java_PI_04
Programa 1	Exitoso	Exitoso	Exitoso	Exitoso
Programa 2	Exitoso	Exitoso	Exitoso	Exitoso
Programa 3 Servicio Web no disponible	Fallido	Fallido	Fallido	Fallido
Programa 4 El código fuente contiene instrucciones no identificadas	Exitoso	Exitoso	Exitoso	Exitoso
Programa 5	Exitoso	Exitoso	Exitoso	Exitoso

Uno de los problemas que se encontró fue que la aplicación cliente no consideraba mostrar el mensaje de error cuando el servicio no se encuentra disponible, y simplemente mostraba una lista vacía en lugar de los archivos generados. El error fue corregido y el caso de prueba exitoso.

5.3.2 Pruebas de Función

Para las pruebas de función se llevó a cabo la prueba de 28 programas divididos en cuatro grupos de prueba como lo muestra la Tabla 16.

Tabla 16. Programas evaluados en las pruebas función

Descripción	Cuenta con:	
	Instrucciones de acceso a archivos	Instrucciones lógicas
Los casos de prueba de la herramienta C2J	No	Si
Los casos de prueba de la herramienta COB2SQL	Si	Si
Programas con instrucciones de acceso a archivos	Si	Si
Programas con instrucciones lógicas en general	No	Si

A continuación se muestran el detalle de cada uno de los casos de prueba para los resultados más significativos.


Caso de prueba: STCob2Java_PF_01

Instancia del caso de prueba	Resultado
1. Usuario y contraseña correcta	Se llevó a cabo la validación y el usuario accedió al sistema
2. Usuario y/o contraseña incorrecta	El sistema validó el usuario y contraseña y envió el mensaje de error
Estatus del caso de prueba: Exitoso	

Caso de prueba: STCob2Java_PF_02





Instancia del caso de prueba	Resultado
1. Cerrar sesión	El sistema cierra la sesión del usuario y muestra la página de inicio de la aplicación.
Estatus del caso de prueba: Exitoso	

Caso de prueba: STCob2Java_PF_03

Instancia del caso de prueba	Resultado
1. El usuario carga el archivo con el código fuente, el sistema realiza la traducción y devuelve el resultado	

<p>2. El usuario elige llevar a cabo la conversión completa</p>	
<p>2. El usuario no selecciona un archivo para llevar a cabo la conversión</p>	
<p>Estatus del caso de prueba: Exitoso</p>	

Caso de prueba: STCob2Java_PF_04

Instancia del caso de prueba	Resultado
<p>1. Código en Cobol con estructuras de datos simples y compuestas (sección "WORKING-STORAGE SECTION")</p> <pre> WORKING-STORAGE SECTION. 01 REGISTRO-TRANSAC. 02 CODIGO-TRANS PIC 9(04). 88 VAL-COD-TRANS VALUE 1 THRU 9999. 02 TIPO-TRANS PIC X. 88 INSERTAR VALUE 'I' 'i'. 88 ELIMINAR VALUE 'E' 'e'. 88 MODIFICAR VALUE 'M' 'm'. 88 SALIR VALUE 'S' 's'. 02 FECHA-NAVE. 05 DIA-NAVE PIC 9(02). 88 VAL-DIA-NAV VALUE 1 THRU 31. 05 MES-NAVE PIC 9(02). 88 VAL-MES-NAV VALUE 1 THRU 12. 05 AÑO-NAVE PIC 9(04). 88 VAL-AÑO-NAV VALUE 1999 THRU 9999. 05 NOMBRE PIC X(15). 02 NOMBRE PIC X(15). 02 NOMBRE-CAP-TRANS PIC X(15). 02 NACION-TRANS PIC X(15). 02 FECHA-TRANS. 05 DIA-TRANS PIC 9(02). 88 VAL-DIA-TRANS VALUE 1 THRU 31. 05 MES-TRANS PIC 9(02). 88 VAL-MES-TRANS VALUE 1 THRU 12. 05 AÑO-TRANS PIC 9(04). 88 VAL-AÑO-TRANS VALUE 1999 THRU 9999. </pre>	<p>Archivos Generados:</p> <p> REGISTRO_TRANSAC.java ×</p> <pre> package DataClasses; import ... public class REGISTRO_TRANSAC { public int CODIGO_TRANS; public String TIPO_TRANS = ""; public FECHA_NAVE FECHA_NAVE = new FECHA_NAVE(); public String NOMBRE = ""; public String NOMBRE_CAP_TRANS = ""; public String NACION_TRANS = ""; public FECHA_TRANS FECHA_TRANS = new FECHA_TRANS(); </pre> <p> FECHA_NAVE.java ×</p> <pre> package DataClasses; import ... public class FECHA_NAVE { public int DIA_NAVE; public int MES_NAVE; public int AÑO_NAVE; public String NOMBRE = ""; </pre> <p> FECHA_TRANS.java ×</p> <pre> package DataClasses; import ... public class FECHA_TRANS { public int DIA_TRANS; public int MES_TRANS; public int AÑO_TRANS; </pre>
<p>2. Código en Cobol con estructuras de datos simples y compuestas (sección "FILE SECTION")</p>	<p>Archivos Generados:</p> <p> C25ql_ALUMNO.java ×</p>

```

FILE SECTION.
FD ALUMNO.
01 REG-ALUM.
  02 COD-ALUM      PIC 9(6).
  88 COD          VALUE 1 THRU 999999.
  02 NOMBRES.
    03 NOMBRE      PIC X(15).
    03 AP-PAT      PIC X(15).
    03 AP-MAT      PIC X(15).
  02 SEXO          PIC A.
  88 SEX          VALUE 'F' 'M'.
  02 EDAD          PIC 9(2).
  88 EDA          VALUE 17 THRU 65.
  02 EST-CIVIL     PIC 9.
  88 E-CIV        VALUE 1 THRU 5.

```

```
package DataClasses;
```

```
import ...
```

```
public class C2Sql_ALUMNO
```

```
{
    public int COD_ALUM;
    public C2Sql_NOMBRES C2Sql_NOMBRES= new C2Sql_NOMBRES();
    public String SEXO = "";
    public int EDAD;
    public int EST_CIVIL;

```

 C2Sql_NOMBRES.java ×

```
public class C2Sql_NOMBRES
```

```
{
    public String NOMBRE = "";
    public String AP_PAT = "";
    public String AP_MAT = "";

```

Estatus del caso de prueba: Exitoso

Caso de prueba: STCob2Java_PF_05

Instancia del caso de prueba	Resultado
<p>1. Tipos de Datos Simples:</p> <pre> DATA DIVISION. WORKING-STORAGE SECTION. 77 NC PIC 9(10). 77 NOMBRE PIC A(30). 77 CALIF1 PIC 9(3). 77 CALIF2 PIC 9(3). 77 CALIF3 PIC 9(3). 77 X PIC 9(2) VALUE 8. 77 I PIC 99 VALUE 1. 77 PROMEDIO PIC 9(5). 77 PROMEDIO1 PIC 999.99. 77 A PIC X(78) VALUE "&". 77 FECHA PIC 9(6). 77 CONT PIC 99 VALUE 1. 77 SUMA PIC 9(3) VALUE ZERO. </pre>	<pre> public int NC; public String NOMBRE = ""; public int CALIF1; public int CALIF2; public int CALIF3; public int X = 8; public int I = 1; public int PROMEDIO; public float PROMEDIO1; public String A = "&"; public int FECHA; public int CONT = 1; public int SUMA = 0; </pre>

<p>2. Instrucción "Accept"</p> <p>Tipo de dato numérico y alfanumérico:</p> <p>PROMEDIOS.</p> <pre> ACCEPT NC ACCEPT NOMBRE ACCEPT CALIF1 ACCEPT CALIF2 ACCEPT CALIF3 </pre>	<pre> public void _PROMEDIOS() { try { NC = Integer.parseInt(auxiliaryBReader.readLine()); } catch (Exception e) { System.out.println("Error al capturar el valor de: NC"); } try { NOMBRE = auxiliaryBReader.readLine(); } catch (Exception e) { System.out.println("Error al capturar el valor de: NOMBRE"); } try { CALIF1 = Integer.parseInt(auxiliaryBReader.readLine()); } catch (Exception e) { System.out.println("Error al capturar el valor de: CALIF1"); } } </pre>
<p>3. Instrucción "Add"</p> <pre> ADD 1 TO I ADD 1 TO X. </pre>	<pre> I += 1; X += 1; </pre>

<p>4. Instrucción "Alter"</p> <pre> METODO-UNO. GO TO METODO-DOS. METODO-DOS. DISPLAY "METODO-DOS" ALTER METODO-UNO TO FIN-PROGRAMA. PERFORM METODO-UNO. </pre>	<pre> public void _METODO_UNO() { if(true) { _METODO_DOS(); isGoToExecuted = true; return; } //Siguiente método a ser ejecutado de acuerdo a la secuencia if (!isGoToExecuted) followMethodToExecute = 2; } public void _METODO_DOS() { System.out.println("METODO-DOS"); _FIN_PROGRAMA(); //Siguiente método a ser ejecutado de acuerdo a la secuencia if (!isGoToExecuted) followMethodToExecute = 3; } </pre>
<p>5. Instrucción "Compute"</p> <pre> COMPUTE SUMA=(CALIF1+CALIF2+CALIF3) COMPUTE PROMEDIO = SUMA / 3 </pre>	<pre> SUMA = (CALIF1 + CALIF2 + CALIF3); PROMEDIO = SUMA / 3; PROMEDIO1 = PROMEDIO; </pre>
<p>6. Instrucción "Display"</p> <pre> DISPLAY A DISPLAY "NO.CONTROL" DISPLAY "NOMBRE" DISPLAY "CALIF1" DISPLAY "CALIF2" DISPLAY "CALIF3" DISPLAY "PROMEDIO" DISPLAY "APROBADO" </pre>	<pre> System.out.println(A); System.out.println("NO.CONTROL"); System.out.println("NOMBRE"); System.out.println("CALIF1"); System.out.println("CALIF2"); System.out.println("CALIF3"); System.out.println("PROMEDIO"); System.out.println("APROBADO"); </pre>

<p>7. Instrucción “Divide”</p> <pre> DIVIDE 100 BY 10 GIVING CALIF1 REMAINDER CALIF2 </pre>	<pre> CALIF1 = 100 / 10; CALIF2 = 100 % 10; </pre>
<p>8. Instrucción “Evaluate”</p> <pre> MOVE 1 TO PLANET-NUMBER. EVALUATE PLANET-NUMBER WHEN 1 MOVE "Mercury" TO PLANET-NAME WHEN 2 MOVE "Venus " TO PLANET-NAME WHEN 3 MOVE "Earth " TO PLANET-NAME WHEN 4 MOVE "Mars " TO PLANET-NAME WHEN 5 MOVE "Jupiter" TO PLANET-NAME WHEN 6 MOVE "Saturn " TO PLANET-NAME WHEN 7 MOVE "Uranus " TO PLANET-NAME WHEN 8 MOVE "Neptune" TO PLANET-NAME WHEN 9 MOVE "Pluto " TO PLANET-NAME WHEN OTHER MOVE " " TO PLANET-NAME END-EVALUATE. </pre>	<pre> PLANET.PLANET_NUMBER = 1; //Inicio Evaluate if(PLANET.PLANET_NUMBER == 1) { PLANET.PLANET_NAME = "Mercury"; } else if(PLANET.PLANET_NUMBER == 2) { PLANET.PLANET_NAME = "Venus "; } else if(PLANET.PLANET_NUMBER == 3) { PLANET.PLANET_NAME = "Earth "; } else if(PLANET.PLANET_NUMBER == 4) { PLANET.PLANET_NAME = "Mars "; } else if(PLANET.PLANET_NUMBER == 5) { PLANET.PLANET_NAME = "Jupiter"; } else if(PLANET.PLANET_NUMBER == 6) { PLANET.PLANET_NAME = "Saturn "; } </pre>

	<pre> else if(PLANET.PLANET_NUMBER == 7) { PLANET.PLANET_NAME = "Uranus "; } else if(PLANET.PLANET_NUMBER == 8) { PLANET.PLANET_NAME = "Neptune"; } else if(PLANET.PLANET_NUMBER == 9) { PLANET.PLANET_NAME = "Pluto "; } else { PLANET.PLANET_NAME = " "; } //Fin Evaluate </pre>
<p>9. Instrucción "Initialize"</p> <pre> FD SupplierFile. 01 SupplierRecord. 02 SupplierCode PIC 99. 02 SupplierName PIC X(3) . 02 SupplierAddress PIC X(50) . 01 PrnSupplierRecord. 02 PrnSupplierCode PIC BB99. 02 PrnSupplierName PIC BBX(20) . 02 PrnSupplierAddress PIC BBX(50) . </pre>	<div data-bbox="1291 779 1638 812">C2Sql_SUPPLIERFILE.java ×</div> <pre> public void initialize() { SUPPLIERCODE = 0; SUPPLIERNAME = ""; SUPPLIERADDRESS = ""; } </pre> <div data-bbox="1291 1047 1638 1079">PRNSUPPLIERRECORD.java ×</div> <pre> public void initialize() { PRNSUPPLIERCODE = ""; PRNSUPPLIERNAME = ""; PRNSUPPLIERADDRESS = ""; } </pre>

<p>10. Instrucción “Inspect”</p> <pre>INSPECT CAT-TYPE replacing characters by "p" before 'O'. DISPLAY CAT-TYPE. INSPECT DOG-TYPE TALLYING Z-COUNT FOR ALL 'Z' BEFORE "R" AFTER "U" 'C'. DISPLAY Z-COUNT.</pre>	<pre>System.out.println(PLANET.PLANET_NAME); CAT_TYPE = inspectReplacing(CAT_TYPE, " " , "p", "O", null, 0, 15); System.out.println(CAT_TYPE); Z_COUNT = inspectTallying(DOG_TYPE, "Z", "R", "U", 0) + inspectTallying(DOG_TYPE, "C", null, null, 0); System.out.println(Integer.toString(Z_COUNT));</pre>
<p>11. Instrucción “Move”</p> <pre>MOVE PROMEDIO TO PROMEDIO1 DISPLAY PROMEDIO1</pre>	<pre>PROMEDIO = SUMA / 3; PROMEDIO1 = PROMEDIO; System.out.println(Double.toString(PROMEDIO1));</pre>
<p>12. Instrucción “Multiply”</p> <pre>MULTIPLY 10 BY SUMA GIVING CALIF1 CALIF2 CALIF3 MULTIPLY 10 BY SUMA CALIF1 CALIF2</pre>	<pre>CALIF1 = 10 * SUMA; CALIF2 = 10 * SUMA; CALIF3 = 10 * SUMA; CALIF1 = 10 * CALIF1; CALIF2 = 10 * CALIF2; SUMA = 10 * SUMA;</pre>

<p>13. Instrucción "Set"</p> <pre> READ SupplierFile NEXT RECORD AT END SET EndOfFile TO TRUE END-READ </pre>	<pre> //Inicio "Read" sqlInterface.ObtenerTabla("SUPPLIERFILE").setLlaveRelativa(SUPPLIERKEY); // 0. Acceso secuencial 1.Acceso aleatorio if(sqlInterface.Read("SUPPLIERFILE",0)) { SUPPLIERSTATUS = CCodigoEstadoSQL.getCodigoEstado(); SUPPLIERKEY = sqlInterface.ObtenerTabla("SUPPLIERFILE").getLlaveRelativa(); C2Sql_SUPPLIERFILE.copyTableToSqlObj(sqlInterface.ObtenerTabla("SUPPLIERFILE")); if(CCodigoEstadoSQL.getCodigoEstado().equals("11")) { setConditionName("ENDOFFILEVALUE", "ENDOFFILE"); } } //Fin "Read" </pre>
<p>14. Instrucción "String"</p> <pre> MAIN-STREET. MOVE 'WILLIAM' TO FIRST-NAME. MOVE 'THOMAS' TO MIDDLE-NAME. MOVE 'ROGERS' TO LAST-NAME. STRING FIRST-NAME DELIMITED BY SPACE ' ' DELIMITED BY SIZE MIDDLE-NAME DELIMITED BY SPACE ' ' DELIMITED BY SIZE LAST-NAME DELIMITED BY SPACE INTO WHOLE-NAME. DISPLAY WHOLE-NAME. </pre>	<pre> FIRST_NAME = "WILLIAM"; MIDDLE_NAME = "THOMAS"; LAST_NAME = "ROGERS"; WHOLE_NAME = FIRST_NAME.substring(0,posDelimitador(FIRST_NAME," ")) + " " + MIDDLE_NAME.substring(0,posDelimitador(MIDDLE_NAME," ")) + " " + LAST_NAME.substring(0,posDelimitador(LAST_NAME," ")); System.out.println(WHOLE_NAME); </pre>
<p>15. Instrucción "Subtract"</p> <pre> SUBTRACT CALIF2 FROM CALIF3 SUMA </pre>	<pre> SUMA -= (CALIF2); CALIF3 -= (CALIF2); </pre>
<p>Estatus del caso de prueba: Exitoso</p>	

Caso de prueba: STCob2Java_PF_06

Instancia del caso de prueba	Resultado
<p>1. Instrucción "IF"</p> <pre> IF (PROMEDIO > 70) DISPLAY "SI" END-IF IF (PROMEDIO < 70) DISPLAY "NO" END-IF </pre>	<pre> if ((PROMEDIO > 70)) { //llave de apertura de if System.out.println("SI"); } //llave de cierre de if if ((PROMEDIO < 70)) { //llave de apertura de if System.out.println("NO"); } //llave de cierre de if </pre>
<p>2. Instrucción "IF" con más de una condición</p> <pre> IF A = B AND B = C MOVE "Los tres son iguales" TO MENJE </pre>	<pre> if (A == B && B == C) { //llave de apertura de if MENJE = "Los tres son iguales"; } //llave de cierre de if </pre>

<p>3. Instrucción “IF-ELSE”</p> <pre> IF OPCION EQUAL 1 PERFORM LLENAR-NAVES ELSE IF OPCION EQUAL 2 PERFORM LLENAR-TRANSACCION UNTIL SEGUIR-NO ELSE IF OPCION EQUAL 5 PERFORM SALIR-PROG END-IF. </pre>	<pre> if (OPCION.equals(1)) { //llave de apertura de if _LLENAR_NAVES(); } //llave de cierre de if else { //llave de apertura de else if (OPCION.equals(2)) { //llave de apertura de if while (!(evaluateCondition("SEGUIR", "SEGUIR_NO"))) { //inicia bloque a ejecutar _LLENAR_TRANSACCION(); } //termina bloque a ejecutar } //llave de cierre de if else { //llave de apertura de else if (OPCION.equals(5)) { //llave de apertura de if _SALIR_PROG(); } //llave de cierre de if } //llave de cierre de else } //llave de cierre de else </pre>
<p>4. Simulación de Instrucción “WHILE”</p> <pre> PERFORM LLENAR-TRANSACCION UNTIL SEGUIR-NO </pre>	<pre> while (!(evaluateCondition("SEGUIR", "SEGUIR_NO"))) { //inicia bloque a ejecutar _LLENAR_TRANSACCION(); } //termina bloque a ejecutar </pre>
<p>5. Simulación de Instrucción “FOR” con condición en variable</p> <pre> PERFORM VARYING I FROM 2 BY 1 UNTIL I > 50 MOVE 1 TO RES PERFORM VARYING J FROM 2 BY 1 UNTIL J >= I COMPUTE DIV = I / J </pre>	<pre> for (I = 2; !(I > 50); I = I + 1) { //inicia bloque a ejecutar RES = 1; for (J = 2; !(J >= I); J = J + 1) { //inicia bloque a ejecutar DIV = I / J; RESIDUO = I - (DIV * J); } } </pre>

<p>6. Simulación de Instrucción “FOR” un número determinado de veces</p> <pre> MOVE SUELDO TO SMAX, SMIN PERFORM 4 TIMES DISPLAY "NOMBRE " ACCEPT NOMBRE DISPLAY "SUELDO "</pre>	<pre> for (int i=0; i < 4; i++) { //inicia bloque a ejecutar System.out.println("NOMBRE "); try { NOMBRE = auxiliaryBReader.readLine(); } catch (Exception e) { System.out.println("Error al capturar el valor de: NOMBRE"); } System.out.println("SUELDO "); }</pre>
Estatus del caso de prueba: Exitoso	

Caso de prueba: STCob2Java_PF_07

Instancia del caso de prueba	Resultado
<p>1. Llamado a método simple</p> <pre> PERFORM MOSTRAREDAD PERFORM VALEDAD UNTIL EDA PERFORM BORRAREDAD</pre>	<pre> _MOSTRAREDAD(); while (!(C2Sql_ALUMNO.evaluateCondition("EDAD", "EDA"))) { //inicia bloque a ejecutar _VALEDAD(); } //termina bloque a ejecutar _BORRAREDAD();</pre>
<p>2. Llamado a un método más de una vez</p> <pre> MOVE 0 TO OPCION PERFORM LLENAR UNTIL OPCION = 4 STOP RUN.</pre>	<pre> _MOSTRAREDAD(); while (!(C2Sql_ALUMNO.evaluateCondition("EDAD", "EDA"))) { //inicia bloque a ejecutar _VALEDAD(); } //termina bloque a ejecutar _BORRAREDAD();</pre>
Estatus del caso de prueba: Exitoso	

Caso de prueba: STCob2Java_PF_08

Instancia del caso de prueba	Resultado
<p>1. Instrucción: "Open" Organización: Secuencial y Relativa</p> <pre> PROCEDURE DIVISION. BEGIN. OPEN I-O SupplierFile. </pre>	<pre> //Tipo de Apertura: "INPUT", "OUTPUT", "IO", "EXTEND" if (sqlInterface.Open("SUPPLIERFILE","IO")) { SUPPLIERSTATUS = CCodigoEstadoSQL.getCodigoEstado(); } </pre>
<p>2. Instrucción: "Close" Organización: Secuencial y Relativa</p> <pre> CLOSE SupplierFile. STOP RUN. </pre>	<pre> if (sqlInterface.Close("SUPPLIERFILE")); { SUPPLIERSTATUS = CCodigoEstadoSQL.getCodigoEstado(); } </pre>
Archivo Secuencial	
<pre> INPUT-OUTPUT SECTION. FILE-CONTROL. SELECT SupplierFile ASSIGN TO "RELSUPP.DAT" ORGANIZATION IS RELATIVE ACCESS MODE IS DYNAMIC RELATIVE KEY IS SupplierKey FILE STATUS IS Supplierstatus. </pre>	
Instancia del caso de prueba	Resultado
<p>3. Instrucción: "Delete" Organización: Secuencial</p>	<pre> sqlInterface.ObtenerTabla("SUPPLIERFILE").setLlaveRelativa(SUPPLIERKEY); //Es eliminado el registro actual leído if (sqlInterface.Delete("SUPPLIERFILE")) { if (CCodigoEstadoSQL.getCodigoEstado().equals("09")) . </pre>

<pre> READ SupplierFile NEXT RECORD AT END SET EndOfFile TO TRUE NOT AT END PERFORM DisplayRecord DELETE SupplierFile RECORD INVALID KEY DISPLAY "EL REGISTRO NO EXISTE: ", SupplierKey NOT INVALID KEY DISPLAY "REGISTRO ELIMINADO " END-READ. </pre>	<pre> System.out.println("Error en el borrado:" + CCodigoEstadoSQL.getDescripcion()); System.out.println("EL REGISTRO NO EXISTE: " + Integer.toString(SUPPLIERKEY)); } if(CCodigoEstadoSQL.getCodigoEstado().equals("00")) { System.out.println("REGISTRO ELIMINADO "); } } //Fin "Delete" </pre>
<p>4. Instrucción: "Read" Organización: Secuencial</p> <pre> READ SupplierFile NEXT RECORD AT END SET EndOfFile TO TRUE NOT AT END PERFORM DisplayRecord END-READ. </pre>	<pre> //Inicio "Read" sqlInterface.ObtenerTabla("SUPPLIERFILE").setLlaveRelativa(SUPPLIERKEY); // 0. Acceso secuencial 1.Acceso aleatorio if(sqlInterface.Read("SUPPLIERFILE",0)) { SUPPLIERSTATUS = CCodigoEstadoSQL.getCodigoEstado(); SUPPLIERKEY = sqlInterface.ObtenerTabla("SUPPLIERFILE").getLlaveRelativa(); C2Sql_SUPPLIERFILE.copyTableToSqlObj(sqlInterface.ObtenerTabla("SUPPLIERFILE")); if(CCodigoEstadoSQL.getCodigoEstado().equals("11")) { setConditionName("ENDOFFILEVALUE", "ENDOFFILE"); } if(CCodigoEstadoSQL.getCodigoEstado().equals("12")) { _DISPLAYRECORD(); } } } //Fin "Read" </pre>

<p>5.</p> <p>Instrucción: "Write"</p> <p>Organización: Secuencial</p> <pre> WRITE SupplierRecord INVALID KEY DISPLAY "EL REGISTRO FUE AGREGADO" NOT INVALID KEY DISPLAY "EL REGISTRO NO FUE AGREGADO". </pre>	<pre> sqlInterface.Write("SUPPLIERFILE"); { SUPPLIERSTATUS = CCodigoEstadoSQL.getCodigoEstado(); if(CCodigoEstadoSQL.getCodigoEstado().equals("09")) { System.out.println("Error en la escritura:" +CCodigoEstadoSQL.getDescripcion()); System.out.println("EL REGISTRO FUE AGREGADO"); } if(CCodigoEstadoSQL.getCodigoEstado().equals("00")) { System.out.println("EL REGISTRO NO FUE AGREGADO"); } } //Fin "Write" </pre>
<p>6.</p> <p>Instrucción: "Rewrite"</p> <p>Organización: Secuencial</p> <pre> READ SupplierFile NEXT RECORD AT END SET EndOfFile TO TRUE NOT AT END DISPLAY "ESCRIBA LA NUEVA DIRECCION --> " WITH NO ADVANCING ACCEPT SupplierAddress REWRITE SupplierRecord INVALID KEY DISPLAY "EL REGISTRO NO EXISTE: ", SupplierKey NOT INVALID KEY DISPLAY "REGISTRO ACTUALIZADO" END-READ. </pre>	<pre> if(sqlInterface.Rewrite("SUPPLIERFILE")) { SUPPLIERSTATUS = CCodigoEstadoSQL.getCodigoEstado(); if(CCodigoEstadoSQL.getCodigoEstado().equals("09")) { System.out.println("Error en la actualización:" + CCodigoEstadoSQL.getDescripcion()); System.out.println("EL REGISTRO NO EXISTE: " + Integer.toString(SUPPLIERKEY)); } if(CCodigoEstadoSQL.getCodigoEstado().equals("00")) { System.out.println("REGISTRO ACTUALIZADO"); } } //Fin "Rewrite" </pre>
Archivo Relativo	

<pre> FILE-CONTROL. SELECT SupplierFile ASSIGN TO "RELSUPP.DAT" ORGANIZATION IS RELATIVE ACCESS MODE IS DYNAMIC RELATIVE KEY IS SupplierKey FILE STATUS IS Supplierstatus. </pre>	
Instancia del caso de prueba	Resultado
<p>7.</p> <p>Instrucción: "Delete"</p> <p>Organización: Relativo</p> <pre> DISPLAY "ESCRIBA EL CODIGO" , " DE LA LLAVE --> " WITH NO ADVANCING ACCEPT SupplierKey DELETE SupplierFile RECORD INVALID KEY DISPLAY "EL REGISTRO NO EXISTE: " , SupplierKey NOT INVALID KEY DISPLAY "REGISTRO ELIMINADO ". </pre>	<pre> C2Sql_SUPPLIERFILE.SUPPLIERCODE = 23; C2Sql_SUPPLIERFILE.SUPPLIERNAME = "BOLAÑOS"; C2Sql_SUPPLIERFILE.SUPPLIERADDRESS = "HUAJUAPAN"; C2Sql_SUPPLIERFILE.copySqlObjToTable(sqlInterface.ObtenerTabla("SUPPLIERFILE")); sqlInterface.ObtenerTabla("SUPPLIERFILE").setLlaveRelativa(SUPPLIERKEY); sqlInterface.Write("SUPPLIERFILE"); { SUPPLIERSTATUS = CCodigoEstadoSQL.getCodigoEstado(); if(CCodigoEstadoSQL.getCodigoEstado().equals("09")) { System.out.println("Error en la escritura:" + CCodigoEstadoSQL.getDescripcion()); System.out.println("EL REGISTRO FUE AGREGADO"); } if(CCodigoEstadoSQL.getCodigoEstado().equals("00")) { System.out.println("EL REGISTRO NO FUE AGREGADO"); } } //Fin "Write" </pre>

<p>8.</p> <p>Instrucción: "Read"</p> <p>Organización: Relativo</p> <pre> DISPLAY "Enter supplier code key" (2 digits) --> " WITH NO ADVANCING ACCEPT SupplierKey READ SupplierFile RECORD INVALID KEY DISPLAY "SUPP STATUS :-" , SupplierStatus END-READ PERFORM DisplayRecord. </pre>	<pre> //Inicio "Read" sqlInterface.ObtenerTabla("SUPPLIERFILE").setLlaveRelativa(SUPPLIERKEY); // 0. Acceso secuencial 1.Acceso aleatorio if(sqlInterface.Read("SUPPLIERFILE",1)) { SUPPLIERSTATUS = CCodigoEstadoSQL.getCodigoEstado(); SUPPLIERKEY = sqlInterface.ObtenerTabla("SUPPLIERFILE").getLlaveRelativa(); C2Sql_SUPPLIERFILE.copyTableToSqlObj(sqlInterface.ObtenerTabla("SUPPLIERFILE")); if(CCodigoEstadoSQL.getCodigoEstado().equals("09")) { System.out.println("Error en la lectura:" + CCodigoEstadoSQL.getDescripcion()); System.out.println("SUPP STATUS :-" + SUPPLIERSTATUS); } } //Fin "Read" </pre>
<p>9.</p> <p>Instrucción: "Write"</p> <p>Organización: Relativo</p> <pre> MOVE 23 TO SupplierCode MOVE "BOLAÑOS" TO SupplierName MOVE "HUAJUAPAN" TO SupplierAddress WRITE SupplierRecord INVALID KEY DISPLAY "EL REGISTRO FUE AGREGADO" NOT INVALID KEY DISPLAY "EL REGISTRO NO FUE AGREGADO". </pre>	<pre> C2Sql_SUPPLIERFILE.SUPPLIERCODE = 23; C2Sql_SUPPLIERFILE.SUPPLIERNAME = "BOLAÑOS"; C2Sql_SUPPLIERFILE.SUPPLIERADDRESS = "HUAJUAPAN"; C2Sql_SUPPLIERFILE.copySqlObjToTable(sqlInterface.ObtenerTabla("SUPPLIERFILE")); sqlInterface.ObtenerTabla("SUPPLIERFILE").setLlaveRelativa(SUPPLIERKEY); sqlInterface.Write("SUPPLIERFILE"); { SUPPLIERSTATUS = CCodigoEstadoSQL.getCodigoEstado(); if(CCodigoEstadoSQL.getCodigoEstado().equals("09")) { System.out.println("Error en la escritura:" + CCodigoEstadoSQL.getDescripcion()); System.out.println("EL REGISTRO FUE AGREGADO"); } } </pre>

	<pre> if(CCodigoEstadoSQL.getCodigoEstado().equals("00")) { System.out.println("EL REGISTRO NO FUE AGREGADO"); } } //Fin "Write" </pre>
<p>10. Instrucción: "Rewrite" Organización: Relativo</p> <pre> DISPLAY "ESCRIBA EL CODIGO", "DE LA LLAVE --> " WITH NO ADVANCING ACCEPT SupplierKey DISPLAY "ESCRIBA LA NUEVA", " DIRECCION --> " WITH NO ADVANCING ACCEPT SupplierAddress REWRITE SupplierRecord INVALID KEY DISPLAY "EL REGISTRO ", "NO EXISTE: ", SupplierKey NOT INVALID KEY DISPLAY "REGISTRO ACTUALIZADO". </pre>	<pre> sqlInterface.ObtenerTabla("SUPPLIERFILE").setLlaveRelativa(SUPPLIERKEY); C2Sql_SUPPLIERFILE.copySqlObjToTable(sqlInterface.ObtenerTabla("SUPPLIERFILE")); if(sqlInterface.Rewrite("SUPPLIERFILE")) { SUPPLIERSTATUS = CCodigoEstadoSQL.getCodigoEstado(); if(CCodigoEstadoSQL.getCodigoEstado().equals("09")) { System.out.println("Error en la actualización:" + CCodigoEstadoSQL.getDescripcion()); System.out.println("EL REGISTRO " + "NO EXISTE: " + Integer.toString(SUPPLIERKEY)); } if(CCodigoEstadoSQL.getCodigoEstado().equals("00")) { System.out.println("REGISTRO ACTUALIZADO"); } } //Fin "Rewrite" </pre>
Archivo Secuencial – Reporte	
<pre> INPUT-OUTPUT SECTION. FILE-CONTROL. SELECT StudentFile ASSIGN TO "STUDENTS.DAT" ORGANIZATION IS LINE SEQUENTIAL. SELECT ReportFile ASSIGN TO "STUDENTS.RPT" ORGANIZATION IS LINE SEQUENTIAL. </pre>	
Instancia del caso de prueba	Resultado

11.

Instrucción: "Write"

Organización: Secuencial

Fuente: Variable de Impresión

```

MOVE StudentCount TO PrnStudentCount
MOVE MaleCount    TO PrnMaleCount
MOVE FemaleCount  TO PrnFemaleCount

```

```

WRITE PrintLine FROM HeadingLine
    AFTER ADVANCING PAGE
WRITE PrintLine FROM StudentTotalLine
    AFTER ADVANCING 2 LINES
WRITE PrintLine FROM MaleTotalLine
    AFTER ADVANCING 2 LINES
WRITE PrintLine FROM FemaleTotalLine
    AFTER ADVANCING 2 LINES.

```

```

STUDENTTOTALLINE.PRNSTUDENTCOUNT =
    Integer.toString(WORKTOTALS.STUDENTCOUNT);
MALETOTALLINE.PRNMALECOUNT =
    Integer.toString(WORKTOTALS.MALECOUNT);
FEMALETOTALLINE.PRNFEMALECOUNT =
    Integer.toString(WORKTOTALS.FEMALECOUNT);

{
    REPORTFILE.imprimir(HEADINGLINE);
} //Fin "Write"
{
    REPORTFILE.imprimir(STUDENTTOTALLINE.display());
} //Fin "Write"
{
    REPORTFILE.imprimir(MALETOTALLINE.display());
} //Fin "Write"
{
    REPORTFILE.imprimir(FEMALETOTALLINE.display());
} //Fin "Write"

```

Estatus del caso de prueba: Exitoso

5.4 Análisis de los Resultados

Se llevaron a cabo un total de 47 casos de prueba agrupados en pruebas de integración y pruebas de función.

Para las pruebas de integración no se encontraron errores importantes, ya que sólo se probó la integración entre los diferentes componentes de la aplicación.

A las pruebas de función corresponden aproximadamente 42 casos de prueba, de los cuales el 100% fueron exitosas, ya que los programas (código fuente en lenguaje COBOL) usados durante las pruebas, fueron seleccionados cuidadosamente, con el fin de evaluar la herramienta dentro de los alcances que fueron definidos previamente.

Para el caso de los programas con instrucciones fuera del alcance solamente se verificó que no generaran algún problema durante la ejecución del programa, obteniendo un resultado satisfactorio.

COBOL es considerado uno de los lenguajes de programación más complejos, por su gran número de palabras reservadas y complejidad sintáctica. Debido a la falta de tiempo suficiente, en este proyecto de tesis se seleccionó un conjunto limitado de casos de prueba, no abarcando todas las posibles combinaciones en los programas escritos en el lenguaje de programación COBOL. Sin embargo se considera suficiente para validar el correcto funcionamiento de la herramienta.

Finalmente, se propone lanzar una primera versión beta de la herramienta, para que los usuarios comiencen a usarla y continuar validando la misma. La herramienta como Servicio Web facilita la accesibilidad a ella para una gran cantidad de usuarios.

Capítulo 6. Conclusiones

6.1 Conclusiones

En el caso de los lenguajes de programación, el proceso de reingeniería es una tarea muy extensa, donde el análisis de cada uno de sus estatutos o instrucciones y su comportamiento y capacidades juega un papel primordial en la reingeniería.

No existe una sola manera de realizar la reingeniería, depende de las herramientas y conocimientos con que se cuente, ya que se pueden usar técnicas tan complejas de Inteligencia Artificial o simplemente tablas de conversión, todo depende del análisis, visión y creatividad de quien este llevando a cabo la traducción.

Para llevar a cabo la traducción de un lenguaje a otro se requiere estudiar las capacidades tanto del lenguaje fuente como destino y determinar si es posible llevar a cabo dicha traducción, o para que casos. Y considerar alternativas de solución.

Una de las partes fundamentales en la traducción es contar con la gramática del lenguaje fuente, a partir de ahí se puede conocer y analizar las características del lenguaje.

Existen diferentes herramientas denominadas “Meta compiladores”, usadas para llevar a cabo la reingeniería, y que especifican la gramática de un lenguaje de programación y facilitan la incorporación de acciones semánticas, generando automáticamente código para llevar a cabo el análisis de programas en el lenguaje especificado. La selección de dicho “meta-compilador” depende de las necesidades específicas de cada tipo de traducción, por lo que es importante conocer las capacidades del “meta-compilador”.

El uso del meta-compilador Javacc fue muy asertivo ya que por un lado las herramientas de antecedente de este trabajo de tesis usaron este meta-compilador y por otro dentro de la documentación de Javacc ya se contaba con la definición completa del estándar de COBOL 85.

Hubo casos en que la traducción no fue directa, es decir no había una instrucción en Java exactamente equivalente a la instrucción en COBOL, por lo que fue necesario implementar métodos y/o conjuntos de instrucciones que permitieran obtener la misma funcionalidad.

Aunque en un principio se pensó en solamente integrar los sistemas realizados como trabajo de antecedente fue necesario llevar a cabo una serie de cambios para poder acoplarlos.

Dentro de las limitaciones podemos mencionar que no fue posible lograr un 100% de la traducción de los estatutos, sin embargo, si se logró un porcentaje considerable, además de tener una herramienta con una arquitectura que facilita la extensión a los estatutos faltantes.

Como se especificó en los resultados de las pruebas, se requiere un mayor número de casos de prueba, sin embargo la herramienta hasta el momento se considera estable y funcional. Se propone la instalación de la herramienta en un servidor de aplicaciones web y con salida a Internet para que diferentes usuarios puedan hacer uso de ella y generar retroalimentación de su funcionamiento.

Finalmente se puede concluir que se logró el objetivo que se definió al generar una herramienta de traducción automática de código legado en lenguaje COBOL hacia código Java.

6.2 Aportaciones

Se desarrolló una herramienta de traducción automática de programas escritos en lenguaje de programación COBOL al lenguaje Java basado en el estándar de COBOL 85, que aunque no contempla el 100% de los estatutos, si proporciona un alto porcentaje de ellos.

La herramienta fué diseñada con una arquitectura con habilidades deseables para todo software como la extensibilidad, robustez, reusabilidad y flexibilidad. Es por ello que pueden agregarse los estatutos no implementados en este trabajo de tesis, además de poder extender a otros estándares o dialectos de COBOL.

Aunque existen herramientas en el mercado que llevan a cabo la traducción de programas en el lenguaje COBOL al lenguaje Java, ninguna de ellas es de origen nacional, por lo que en este trabajo de tesis se considera como una aportación a la tecnología nacional.

6.3 Trabajo Futuro

Como trabajo futuro, se propone terminar de implementar el 100% de los estatutos del estándar de COBOL 85, uno de los pasos mas fuertes es considerar a los arreglos durante la traducción, a partir de ahí se pude llevar a cabo un número significativo de verbos relacionados con los arreglos.

Los programas con varios archivos fuente es otro paso importante en el trabajo futuro de este proyecto de tesis, ya que se deberá buscar una estrategia para incorporarlos, y ésta puede ser:

- Agregar el código fuente de los sub-programas al programa principal y llevar a cabo la traducción.
- O traducir el sub-programa, generando una nueva clase y generar y agregar un objeto de dicha clase a la clase principal.

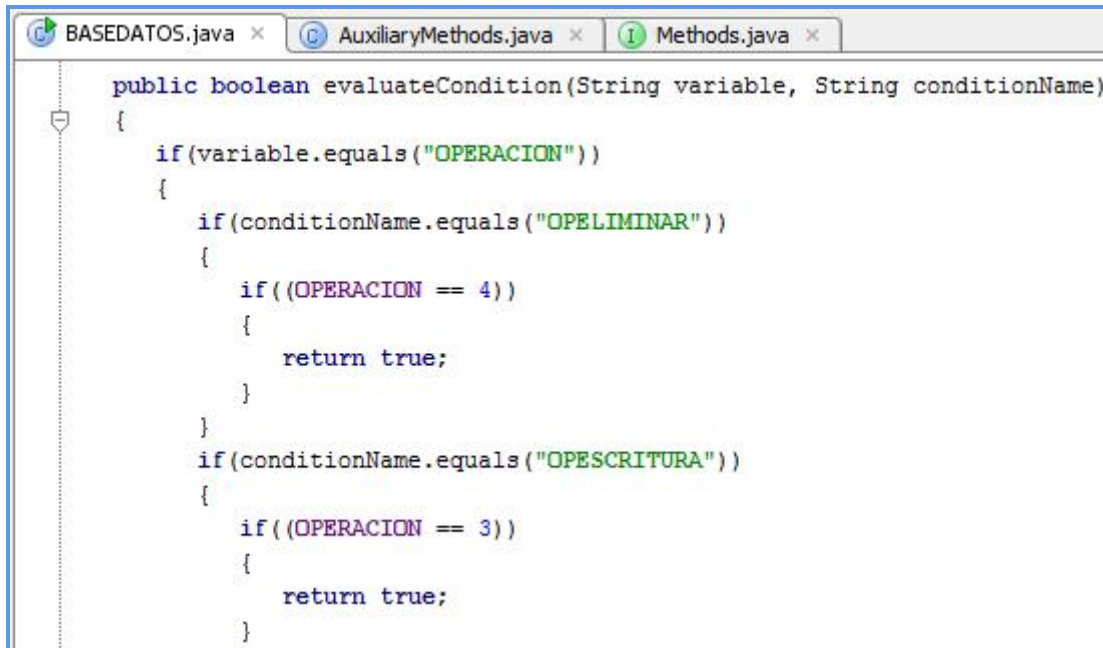
Anexo A. Capacidades del Sistema

A continuación se enlistan y detallan cada una de las capacidades del Sistema de Transformación de Cobol a Java:

1. **El Sistema fue implementado como servicio Web, además del cliente Web para el acceso.**
2. **El Sistema esta basado en el Estándar de COBOL 85.**
3. **Los comentarios aceptados por el Sistema deben iniciar con un asterisco seguido de un signo de mayor que (*>).**
4. **El código de todas las clases generadas (traducidas), esta completamente identadas.**
5. **Los tipos de datos traducidos son: alfabéticos, numéricos y alfanuméricos.**

6. Implementación de las variables de condición en Java.

El sistema implementa dos métodos para simular las variables de condición, el primero realiza la comparación de las variables asociadas y devuelve verdadero si la variable asociada tiene algún valor definido en la variable de condición. Esto con el fin de usar las variables en las condiciones. Ejemplo Figura 52.



```

public boolean evaluateCondition(String variable, String conditionName)
{
    if(variable.equals("OPERACION"))
    {
        if(conditionName.equals("OPELIMINAR"))
        {
            if((OPERACION == 4))
            {
                return true;
            }
        }
        if(conditionName.equals("OPESCRITURA"))
        {
            if((OPERACION == 3))
            {
                return true;
            }
        }
    }
    return false;
}

```

Figura 52. Evaluación de las variables de condición

Donde:

- **Variable:** Es la variable, la cuál es evaluada para verificar si su valor esta dentro de los valores definidos en la variable de condición.
- **ConditionName:** Es la variable de condición, la cuál define el rango de valores o valores permitidos para la variable

El método es implementado en cada clase donde sean definidas las variables de condición. Primero, se evalúa cuál variable se esta evaluando, después se evalúa cuál variable de condición, para posteriormente verificar si el valor de la variable corresponde a alguno definido dentro de la variable de condición, si fuera el caso el método devuelve "true", en cualquier otro "false".

El segundo método asigna el primer valor definido en la variable de condición a la variable asociada o asigna el valor mínimo dentro del rango de valores permitidos por la variable de condición, si fuera el caso. De igual manera que el primer método con el fin de simular la asignación de "TRUE" a la variable de condición. Ejemplo Figura 53.

```

public void setConditionName(String variable, String conditionName)
{
    if (variable.equals("OPERACION"))
    {
        if (conditionName.equals("OPELIMINAR"))
        {
            OPERACION = 4;
        }
        if (conditionName.equals("OPESCRITURA"))
        {
            OPERACION = 3;
        }
    }
}

```

Figura 53. Asignación de "TRUE" a las variables de condición

7. Verbos traducidos correspondiente a estatutos lógicos

La siguiente tabla muestra una lista de los verbos en COBOL traducidos por el Sistema, especificando la funcionalidad particular sobre la cuál se ha enfocado la traducción.

Verbo	Funcionalidad Particular	Estatus de Implementación
Accept	Lectura de teclado, datos de tipo: numéricos, alfanuméricos y tipo fecha.	Implementada
	Lectura de datos de tipo estructura	Implementada
Add	Sumas de variables o constantes numéricas a otra variable, guardándose el resultado en esta variable.	Implementada
	Sumas de variables o constantes numéricas, almacenando el resultado en otra variable sin considerar el valor inicial de ésta.	Implementada
	Considera el redondeo.	Implementada
Alter	Modificación de párrafos en los cuales sólo es ejecutada una instrucción "GO TO", y modifica el párrafo que es invocado.	Implementada
Call	NA	Fuera de alcance
Cancel	NA	Fuera de alcance
Compute	Operaciones aritméticas.	Implementada
	Considera el redondeo.	Implementada
Continue	Rompe la secuencia de la instrucción if.	Implementada
Display	Impresión en pantalla de variables o constantes de tipo numérico o alfanumérico.	Implementada
	Impresión en pantalla de variables de tipo estructura.	Implementada
Divide	Divide una variable o constante numérica entre otra variable.	Implementada
	Divide una variable o constante numérica entre otra, almacenando el resultado en una variable diferente a las dos primeras.	Implementada
	Almacena el valor del residuo en otra variable.	Implementada
Evaluate	Evalúa más de una condición.	Implementada
	Evalua variables de condición.	Implementada
Exit	Detiene la ejecución del programa.	Implementada
Exit Program	NA	Fuera de alcance

Goback	NA	Fuera de alcance
Goto	Salto no controlados a métodos.	Implementada
If-else	Instrucciones if-else anidadas.	Implementada
	Considera variables de condición.	
	Considera más de una condición.	
Initialize	Inicializa variables numéricas, alfanuméricas y alfabéticas.	Implementada
	Inicializa variables de tipo registro.	Implementada
Inspect	Contador de caracteres	Implementada
	Contador de palabras dentro de otra palabra : <ul style="list-style-type: none"> ■ Las primeras ■ Todas ■ En una sub-cadena 	Implementada
	Reemplaza todos los caracteres por otro caracter.	Implementada
	Reemplazar y contar caracteres.	Implementada
Merge	NA	Fuera de alcance
Move	Asignación de variables o constantes a otras.	Implementada
	Conversión de tipos (alfanuméricos, numéricos).	Implementada
Multiply	Multiplica una variable o constante numérica con otra variable, guardándose el resultado en la última variable.	Implementada
	Multiplica una variable o constante numérica por otra, almacenando el resultado en una variable diferente a las dos primeras.	Implementada
Perform	Invocación a métodos.	Implementada
	Simulación de instrucciones for . Ejecución de bloques de código un número constante de veces o dependiendo de una variable, dando un incremento.	Implementada
	Simulación de instrucciones while . Ejecución de bloques de código dependiendo del valor de una condición.	Implementada
	Simulación de instrucciones do-while . Ejecución de bloques de código, verificando la condición al final del bloque.	Implementada
Release	NA	Fuera de alcance
Return	NA	Fuera de alcance
Search	NA	Fuera de alcance
Set	Asignación de un valor a una variable	Implementada
	Asignación del valor "true" a una variable de condición.	Implementada
Sort	NA	Fuera de alcance
Start	NA	Fuera de alcance
Stop	Fin de la secuencia de métodos.	Implementada
String	Concatenación de cadenas (variables o literales)	Implementada
	Concatenación de cadenas delimitadas por: <ul style="list-style-type: none"> ■ Tamaño ■ Variable ■ Cadena 	Implementada
	Asignación de la cadena concatenada a otra cadena en una posición específica.	Implementada
Sustract	Resta de variables o constantes numéricas a otra variable, guardándose el resultado en la última variable.	Implementada
	Resta de variables o constantes numéricas, almacenando el resultado en otra	Implementada

	variable sin considerar el valor inicial de ésta.	
	Considera el redondeo.	Implementada
Unstring	Divide una cadena y almacena el contenido de ellas en otras variables considerando un delimitador.	Implementada

8. Métodos auxiliares para igualar funcionalidad

En algunos casos, fue necesario llevar a cabo la implementación de uno o más métodos en Java, que en su conjunto permitan igualar la funcionalidad en COBOL. A continuación se presenta una tabla con los verbos para los cuales se tuvo que generar métodos auxiliares para la traducción. En algunos casos los métodos son utilizados para más de un verbo.

Verbo/Instrucción	Método
Accept	accept. Asigna valores a todos los atributos de instancias de la clase, solicitando los valores al usuario. En caso que el atributo sea de referencia, entonces se invocará el método accept del objeto correspondiente.
Display	display. Muestra en pantalla el valor de todos los atributos de instancias de la clase. En caso que el atributo sea de referencia, entonces se invocará el método display del objeto correspondiente.
Initialize	initialize. Inicializa todos los atributos de la clase a 0 si el atributo es numérico o "" si el atributo es de tipo alfanumérico. En caso que el atributo sea de referencia, entonces se invocará el método initalize del objeto correspondiente.
	initializeReplacing. Inicializa todos los atributos de la clase a un valor determinado por el usuario. En caso que el atributo sea de referencia, entonces se invocará el método initalizeReplacing del objeto correspondiente, con los mismos valores de inicialización.
Read Write Rewrite Delete	copySqlObjToTable. Copia la información de un objeto obtenida de una tabla en SQL a los objetos que definen una estructura de datos en Java.
	copyTableToSqlObj. Copia la información contenida en los objetos en Java (registros en COBOL) a un objeto que define la estructura de los datos en SQL para guardar la información en una base de datos en SQL.
String	delimitedPosition. Contenida en la clase AuxiliaryMethods , regresa la posición real en una cadena a partir de la cual se llevará a cabo una concatenación, partiendo de un delimitador.
Inspect	inspectTallying. Contenida en la clase AuxiliaryMethods , regresa el número de veces que aparece una subcadena en una cadena.
Inspect	inspectConverting. Contenida en la clase AuxiliaryMethods , regresa una cadena reemplazando ciertos caracteres por otros.
Inspect	inspectReplacing. Contenida en la clase AuxiliaryMethods , regresa una cadena reemplazando subcadenas en la cadena original: una, las primeras o todas las subcadenas encontradas.

9. El método principal “Main” en Java

El método main es el encargado de llevar a cabo la orquestación del flujo del proceso de un programa en Java e inicia las secuencias interactivas a través de la ejecución del programa. La variable **followMethodToExecute** es un índice e indica el método que será ejecutado. Sólo

aquellos párrafos definidos antes de declarar la instrucción “STOP RUN” tienen asignado un número de secuencia para su ejecución. Como lo muestra la Figura 54a, de el código en Lenguaje COBOL de lado derecho, el párrafo “**Begin**” contiene la instrucción “STOP RUN”, lo cuál indica que en el programa traducido en Java, en el método principal “**main**”, únicamente se llevará a cabo el llamado al método “**Begin**”.

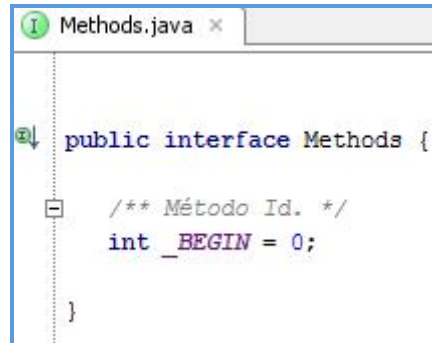
El método “**main**” mostrado en la Figura 54b contiene los siguientes elementos:

- Variable **CobolCode**. Mantiene la referencia al objeto que contiene la clase principal de la aplicación.
- Variable **followMethodToExecute**. La cuál indica el siguiente método a ejecutarse, a cada método es asignado un número entero empezando por 0 e incrementándose en uno conforme son reconocidos durante el análisis del código fuente en COBOL. El valor de la variable es incrementado al final de la ejecución de las instrucciones de cada método.
- Instrucción **while**. Se lleva a cabo un ciclo de ejecuciones de llamados a métodos, mientras el valor de **followMethodToExecute** sea menor al número de párrafos encontrados durante el análisis (antes de la instrucción “STOP RUN”).
- Instrucción **switch**. Evalúa el método a ejecutarse, dependiendo del valor de **followMethodToExecute**.

COBOL	JAVA
<pre> PROCEDURE DIVISION. Begin. DISPLAY "Elegir Operacion: " DISPLAY "1. Lectura" DISPLAY "2. Escritura" DISPLAY "3. Eliminar " DISPLAY "4. Actualizar -->" WITH NO ADVANCING. ACCEPT Operacion. IF OpcSeqWrite PERFORM SeqWrite END-IF. IF OpcGetStudentDetails PERFORM GetStudentDetails END-IF. IF OpcSeqReadNo88 PERFORM SeqReadNo88 END-IF. IF OpcSeqRead PERFORM SeqRead END-IF. STOP RUN. </pre>	<pre> public static void main(String[] args) { BASEDATOS CobolCode = new BASEDATOS(); CobolCode.followMethodToExecute = _BEGIN; while(CobolCode.followMethodToExecute < 1) { switch(CobolCode.followMethodToExecute) { case _BEGIN: CobolCode._BEGIN(); break; } CobolCode.isGoToExecuted = false; } } </pre>
a)	b)

Figura 54. Control de un programa en Java

Como auxiliar en la ejecución del programa en Java, existe la **interface** “*Methods*” (Ver Figura 55), la cual asigna un índice a cada nombre de párrafo encontrado en el código en COBOL. Esto permite en el método **main** manejar como índices los nombres de los métodos.



```

public interface Methods {

    /** Método Id. */
    int _BEGIN = 0;

}

```

Figura 55. Interface “*Methods*”

10. Párrafos

Para llevar a cabo la implementación de los párrafos en Java, se definió una correspondencia entre cada párrafo con un método, el nombre del método será el mismo del párrafo y no tendrá valor de retorno ni paso de parámetros, ya que como fue mencionado en la sección 3.1, en Cobol sólo existen variables globales. Por lo que no es necesario ni el envío de parámetros ni valores de retorno que corresponden a variables locales.

11. GO TO

La instrucción “GO TO”, significa hacer un salto de un párrafo a otro párrafo, rompiendo la forma secuencial de la ejecución, esto es porque una vez que termine de ejecutarse las instrucciones correspondientes al párrafo destino, no regresa al punto de origen donde fue invocado el salto, y el proceso o secuencia continúa con el siguiente párrafo que se encuentre. Para implementar esta instrucción se trabajó con la variable que indica el método siguiente a ser ejecutado (**followMethodToExecute**), en caso de haberse encontrado la instrucción “GO TO”, hace caso omiso a la instrucción de asignación del método siguiente a ejecutar. En la Figura 57 se muestra un ejemplo de implementación del código de Figura 56.

COBOL	
	<pre> METODO-UNO. DISPLAY "Ejemplo del GOTO". GO TO FIN-PROGRAMA. DISPLAY "No se ejecuta". </pre>

Figura 56. Ejemplo de instrucción “GO TO” en COBOL

JAVA

```

public void _METODO_UNO()
{
    System.out.println("Ejemplo del GOTO");
    if(true)
    {
        _FIN_PROGRAMA();
        isGoToExecuted = true;
        return;
    }
    System.out.println("No se ejecuta");

    //Siguiendo método a ser ejecutado de acuerdo a la secuencia
    if (!isGoToExecuted) followMethodToExecute = 2;
}

```

Figura 57. Implementación de la instrucción "GO TO" en Java

12. Generación de clases para estructuras de datos

Como fué explicado en la sección 3.4 para cada una de las estructuras en COBOL, existe una clase en Java, que define como atributos de la clase cada uno de los componentes o campos de la estructura original. El nombre de dichas clases es el nombre de las estructuras.

Para las estructuras definidas en la sección **File Section**, el nombre de las clases y objetos tienen el prefijo **C2Sql_**, de esa manera se distinguen de las estructuras definidas en la sección **Working-Storage Section**.

13. Verbos de Acceso a Archivos Secuenciales

Los archivos secuenciales son el tipo más sencillo de los tipos de archivos manejados por COBOL, ya que no dependen de ningún índice o llave a la hora de acceder a los registros, todo depende del orden en que son almacenados.

Para los archivos secuenciales, el único tipo de acceso es el secuencial.

De acuerdo al análisis descrito en la sección §3.8, los verbos implementados son los siguientes: **Open, Close, Read, Write, Rewrite, Delete**.

Con el tipo de archivo como secuencia, es posible llevar a cabo la escritura de reportes, esta funcionalidad también está implementada en el presente proyecto de tesis.

14. Verbos de Acceso a Archivos Relativos

En los archivos relativos, el tipo de acceso puede ser secuencial, aleatorio o dinámico. El acceso aleatorio significa que dependiendo de los parámetros definidos en cada instrucción de acceso a archivos (Ejem. Write, Read, etc.) el acceso será secuencial o aleatorio, dependiendo del caso.

Los verbos implementados de acuerdo a la sección §3.8 son los siguientes: **Open, Close, Read, Write, Rewrite, Delete.**

15. Verbos de Acceso a Archivos Indexados

Al igual que para los archivos indexados, en los archivos existe de la misma manera el acceso secuencial, aleatorio o dinámico. De acuerdo a la sección §3.8, se llevó a cabo la implementación de todos los verbos definidos para el acceso a archivos: **Open, Close, Read, Write, Rewrite, Delete.**

16. Variables definidas por el Sistema: STCob2Java

Durante la traducción de los programas en COBOL es necesario contar con diferentes variables auxiliares que permitan ayudar a la traducción. A continuación se muestran las variables comunes en todos los programas traducidos a Java y que son definidas con un propósito en específico.

Variable	Tipo de Dato (Java)	Descripción
followMethodToExecute	int	Indica el índice del siguiente método que será ejecutado.
isGoToExecuted	boolean	Si es verdadero indica que la invocación proviene de una instrucción go to y por lo tanto no considerará el orden en la declaración de los métodos.
auxiliaryInputSR	InputStreamReader	Variable auxiliar para lectura desde el teclado.
auxiliaryBReader	BufferedReader	Variable auxiliar para almacenar una cadena y posteriormente formatearla de acuerdo a las necesidades específicas.
auxiliaryNumberFormat	NumberFormat	Variable auxiliar para formatear números en caso que se indique un número de decimales, por ejemplo.
auxiliaryString	String	Variable auxiliar para ayudar a igualar la funcionalidad entre los programas en COBOL y Java.
auxiliaryDate	Date	Variable auxiliar para manejar fechas, cuando así se requiera.
auxiliarySimpleDF	SimpleDateFormat	Variable auxiliar para formatear datos de entrada con formato de fecha.
fileAccessType	String	Variable que almacena el tipo de archivo al que se está refiriendo en la instrucción correspondiente.

sqlInterface	CBDSQL	Define el acceso a las clases de acceso a la base de datos en SQL.
---------------------	--------	--

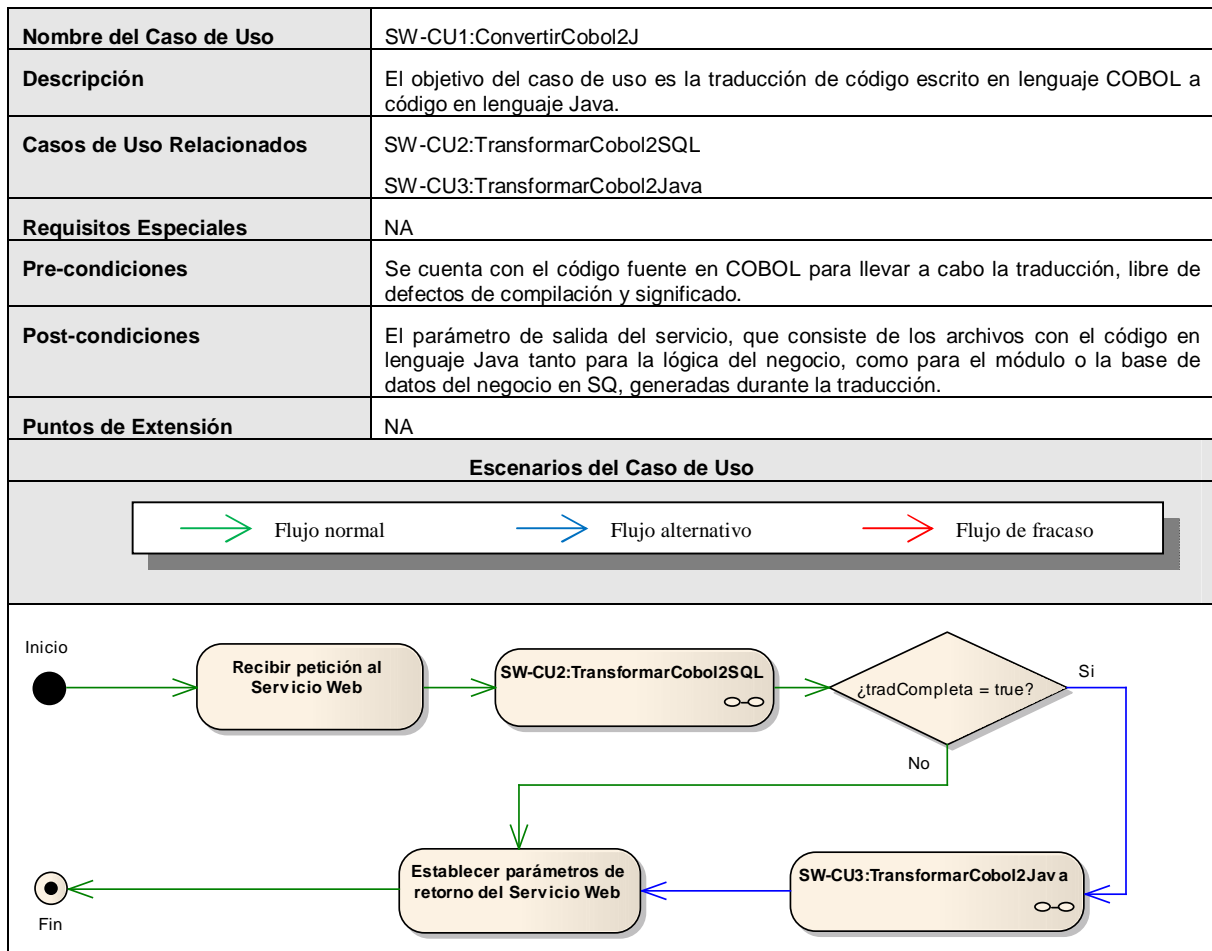
17. Arreglos

En COBOL es posible definir arreglos con la instrucción “OCCURS” en la definición de la variable. Con esto se hace posible el manejo de tablas, teniendo arreglos bidimensionales. Debido a la carencia de tiempo suficiente, en este trabajo de tesis, no fue posible la implementación de los arreglos, pero si se llevo a cabo el análisis para poder realizar la traducción.

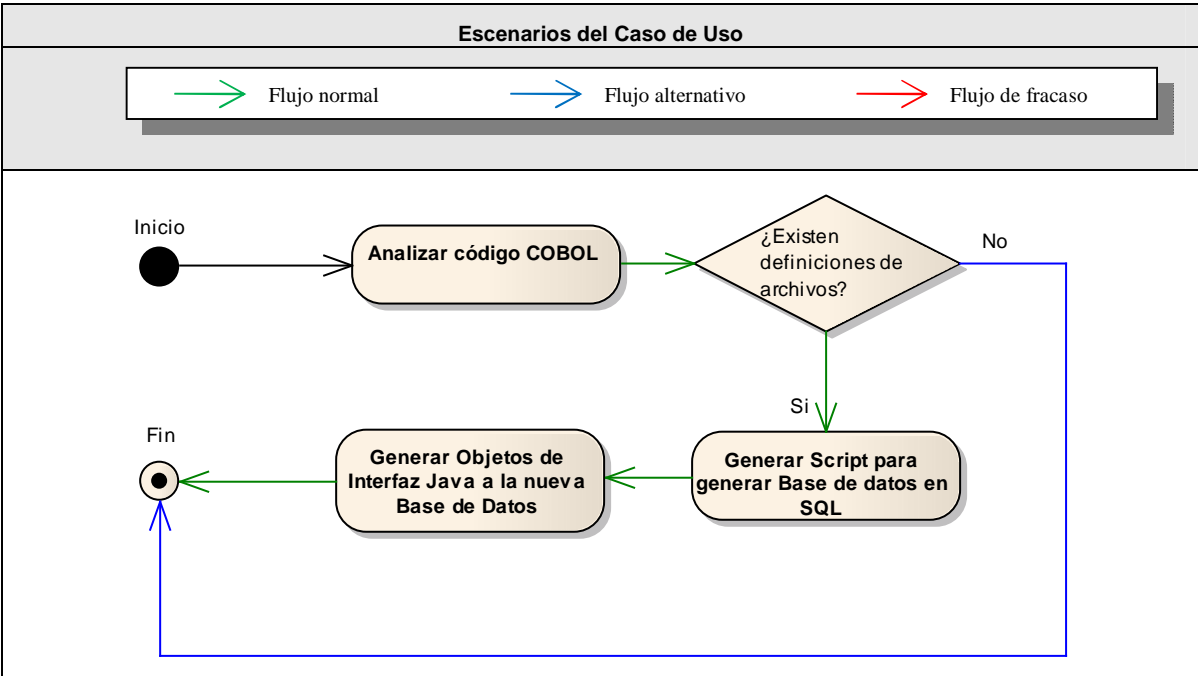
Para el caso de los datos que no son registros, simplemente se deberá definir como arreglos de tipo “String”, “int”, etc. Para las variables de tipo registro, será necesario definir un tipo de dato “ArrayList” en Java, los cuales contendrán objetos del tipo de la clase que lo esté definiendo, se tendrá que crear un constructor que inicialice los objetos y las clases.

Anexo B. Detalle de los de Casos de Uso de la Herramienta

En esta sección se describen a detalle cada uno de los casos de uso mencionados en la sección 4.1.1 y 4.2.1. Para cada caso de uso se enuncia una breve descripción de su objetivo, los casos de uso relacionados, requisitos y lo más importante, se ilustran los escenarios identificados mediante diagramas de actividad.

❖ **Casos de Uso del Servicio Web**

Nombre del Caso de Uso	SW-CU2:TransformarCobol2SQL
Descripción	El objetivo del caso de uso es llevar a cabo la traducción de instrucciones de manipulación de archivos y definiciones de estructuras de archivos escritas en lenguaje del estándar de COBOL 85, hacia instrucciones de manipulación de datos escritas en lenguaje Java y la definición de una base de datos relacional usando el lenguaje SQL 92.
Casos de Uso Relacionados	<ul style="list-style-type: none"> SW-CU1:ConvertirCobol2J SW-CU2.1:AnalizarCodigoCobolSQL SW-CU2.2:GenerarScriptSQL SW-CU2.3:GenerarInterfazBD
Requisitos Especiales	<ul style="list-style-type: none"> NA
Pre-condiciones	<ul style="list-style-type: none"> El sistema cuenta con el código escrito en lenguaje COBOL. Dicho código es ejecutable en algún compilador de COBOL que soporte la versión estándar COBOL 85. El código esta libre de defectos de compilación y de significado.
Post-condiciones	<ul style="list-style-type: none"> Es generado un script que corresponde al esquema de la base de datos. Son generadas una serie de clases (código escrito en lenguaje Java) que representan la interfaz entre cualquier aplicación escrita en Java y la nueva Base de Datos definida en el punto anterior.
Puntos de Extensión	NA



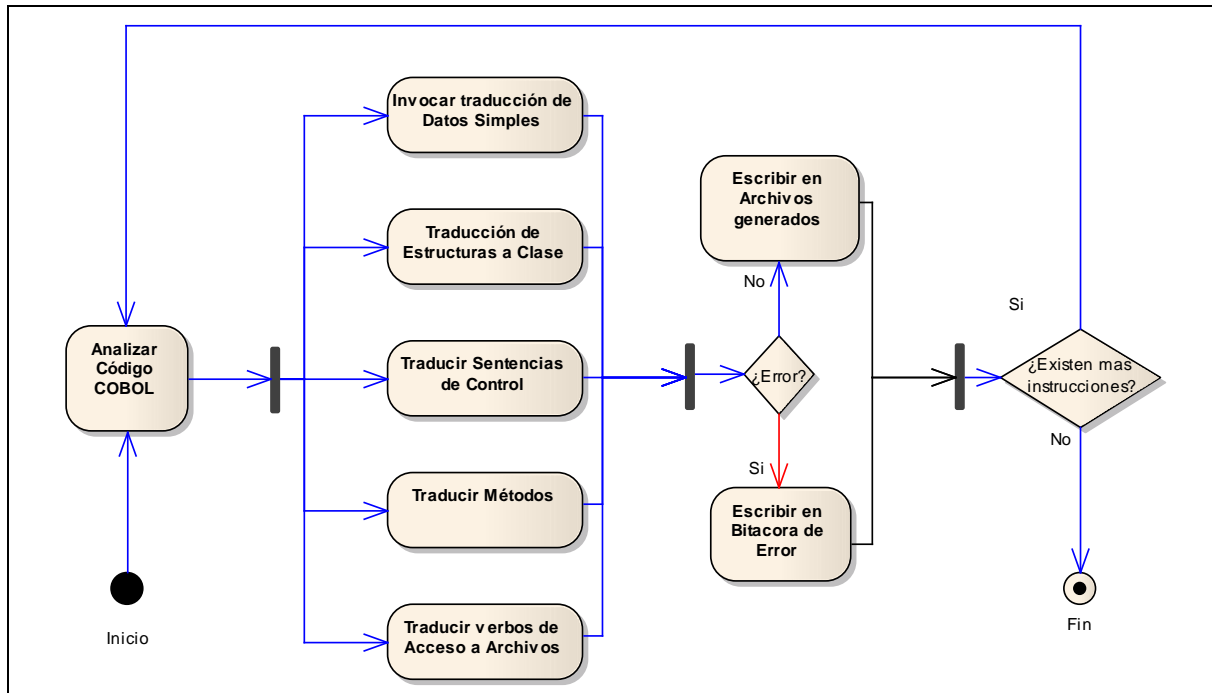
Nombre del Caso de Uso	SW-CU3:TransformarCobol2Java
Descripción	El objetivo del caso de uso es llevar a cabo la traducción de las instrucciones no relacionadas con la manipulación de archivos escrito en COBOL a código escrito en lenguaje Java.
Casos de Uso Relacionados	<ul style="list-style-type: none">SW-CU1:ConvertirCobol2JSW-CU3.1:AnalizarCodigoCobol
Requisitos Especiales	<ul style="list-style-type: none">NA
Pre-condiciones	<ul style="list-style-type: none">El sistema cuenta con el código escrito en lenguaje COBOL. Dicho código es ejecutable en algún compilador de COBOL que soporte la versión estándar COBOL 85.El código esta libre de defectos de compilación y de significado.
Post-condiciones	Son generados una serie de archivos correspondientes a la traducción de código COBOL a Java, junto con un archivo de bitácora de error donde se especifica los errores encontrados durante la traducción.
Puntos de Extensión	NA

Escenarios del Caso de Uso

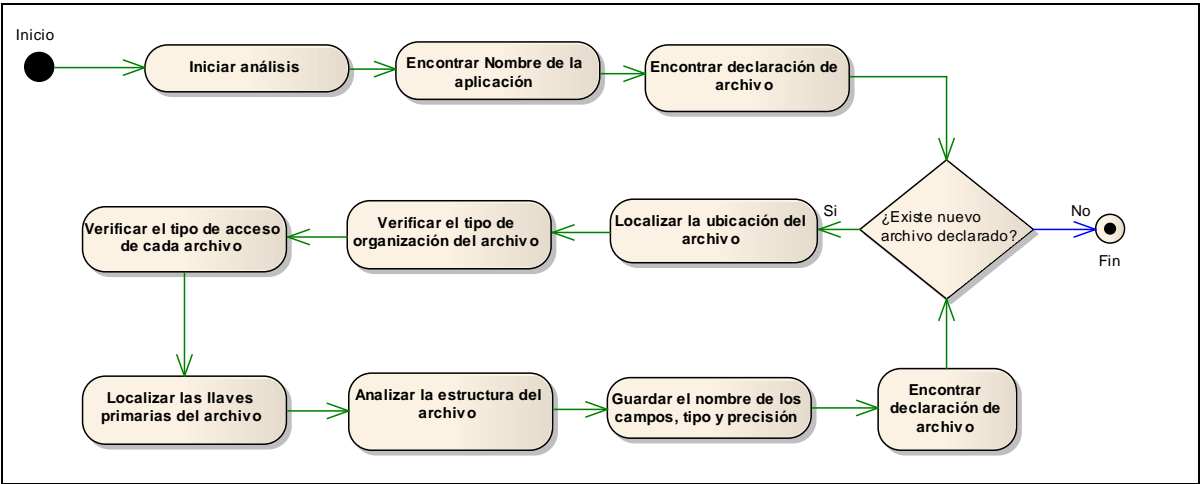
→ Flujo normal

→ Flujo alternativo

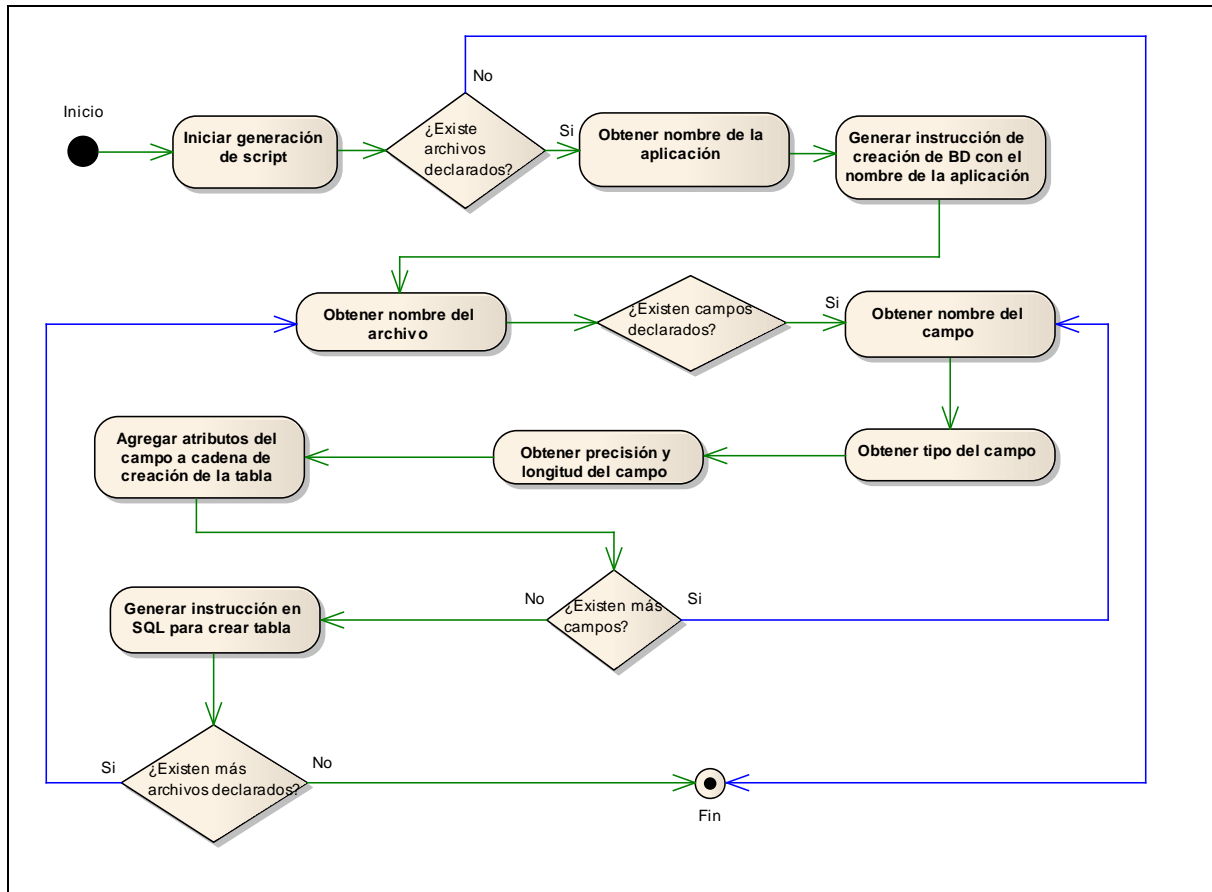
→ Flujo de fracaso



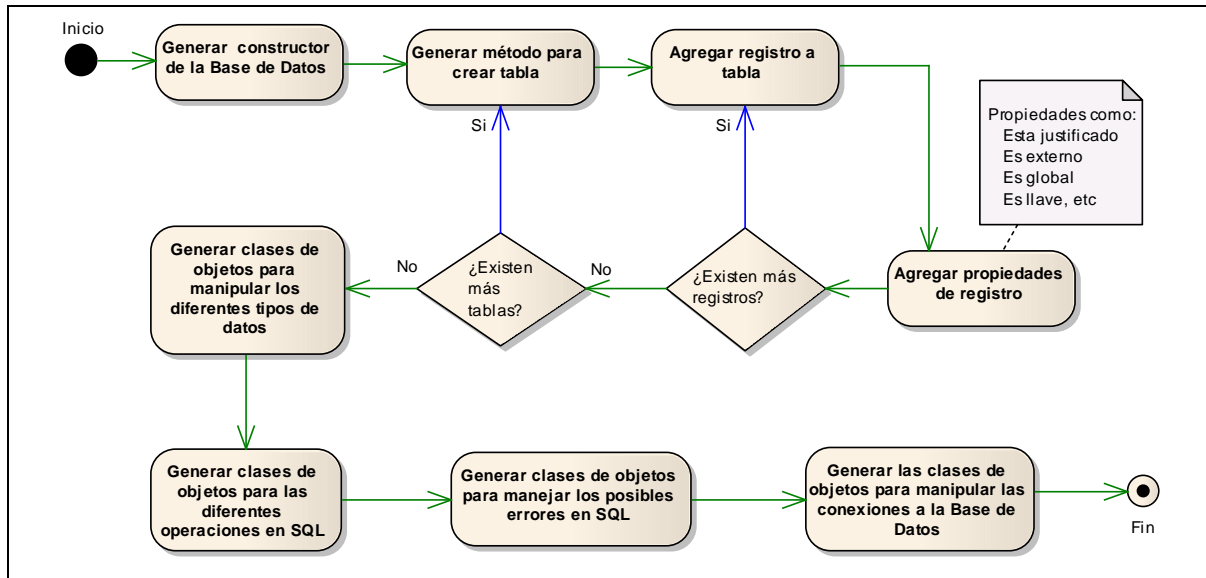
Nombre del Caso de Uso	SW-CU2.1:AnalizarCodigoCobolSQL
Descripción	El objetivo del caso de uso es desde el código fuente escrito en lenguaje COBOL identificar las diferentes instrucciones referentes al manejo del sistema de archivos de la aplicación.
Casos de Uso Relacionados	<ul style="list-style-type: none"> SW-CU2:TransformarCobol2SQL
Requisitos Especiales	<ul style="list-style-type: none"> NA
Pre-condiciones	<ul style="list-style-type: none"> El archivo con el código fuente a traducir se encuentra cargado en el sistema. El código fuente se encuentra libre de defectos de compilación y significado.
Post-condiciones	Las instrucciones referentes al sistema de archivos se encuentran almacenadas en tablas o archivos de trabajo para su futura transformación.
Puntos de Extensión	NA
Escenarios del Caso de Uso	
<div style="display: flex; align-items: center; justify-content: space-around;"> <div style="display: flex; align-items: center;"> → Flujo normal </div> <div style="display: flex; align-items: center;"> → Flujo alternativo </div> <div style="display: flex; align-items: center;"> → Flujo de fracaso </div> </div>	



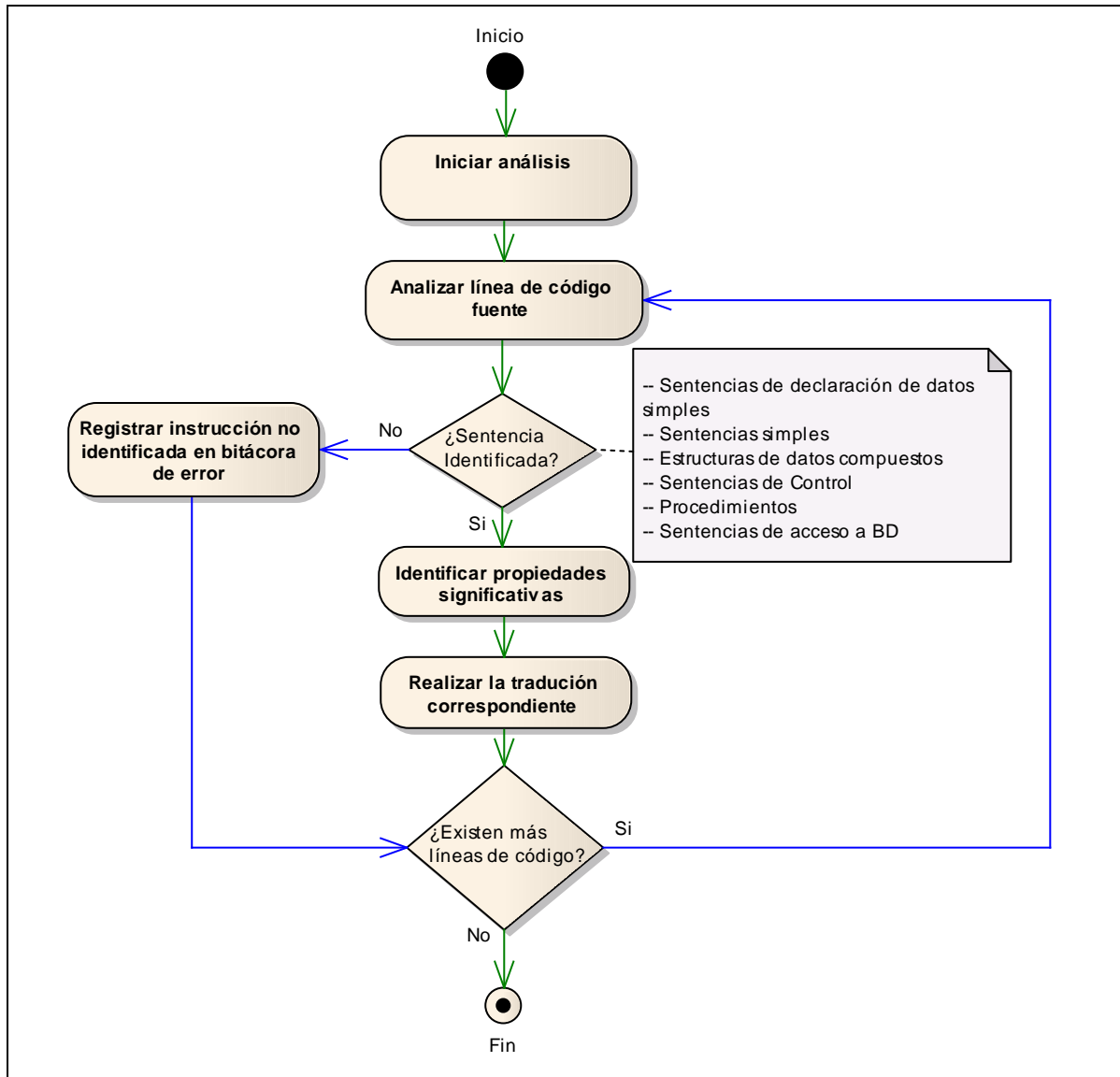
Nombre del Caso de Uso	SW-CU2.2:GenerarScriptSQL
Descripción	El objetivo del caso de uso es generar un script, escrito en el estándar SQL 92, que contenga el esquema de una base de datos equivalente a la declaración y manejo del sistema de archivos en COBOL.
Casos de Uso Relacionados	<ul style="list-style-type: none">SW-CU2:TransformarCobol2SQL
Requisitos Especiales	<ul style="list-style-type: none">NA
Pre-condiciones	<ul style="list-style-type: none">El análisis del código fuente fue realizado previamente.
Post-condiciones	Es generado un archivo con información del esquema de la base de datos en formato de script para ser ejecutado sobre algún manejador de base de datos compatible con el estándar SQL 92.
Puntos de Extensión	NA
Escenarios del Caso de Uso	
<div><div> Flujo normal</div><div> Flujo alternativo</div><div> Flujo de fracaso</div></div>	



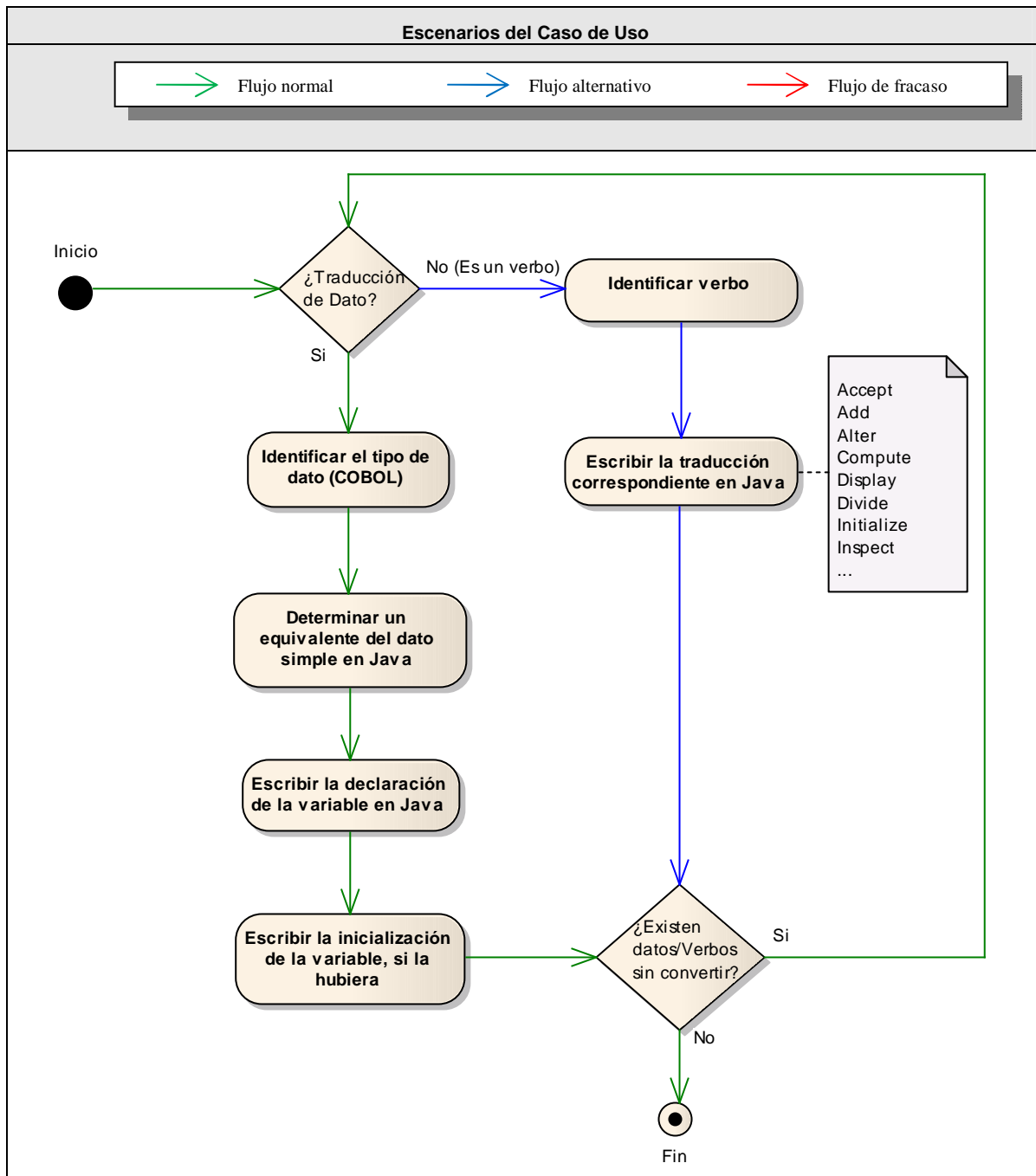
Nombre del Caso de Uso	SW-CU2.3:GenerarInterfazBD
Descripción	El objetivo del caso de uso es generar una serie de clases (clases interfaz entre la base de datos y el código traducido en lenguaje Java) que permitan llevar a cabo las operaciones de inserción, borrado, lectura, escritura y actualización desde el código en Java a la Base de Datos en SQL.
Casos de Uso Relacionados	<ul style="list-style-type: none"> SW-CU2:TransformarCobol2SQL
Requisitos Especiales	<ul style="list-style-type: none"> NA
Pre-condiciones	<ul style="list-style-type: none"> El análisis del código fuente fue realizado previamente. La base de datos (tablas en SQL) ha sido generada.
Post-condiciones	Se generan una serie de clases de objetos en lenguaje Java para acceder a la nueva base de datos.
Puntos de Extensión	NA
Escenarios del Caso de Uso	
<div style="display: flex; justify-content: space-around; align-items: center;"> <div style="display: flex; align-items: center;"> → Flujo normal </div> <div style="display: flex; align-items: center;"> → Flujo alternativo </div> <div style="display: flex; align-items: center;"> → Flujo de fracaso </div> </div>	



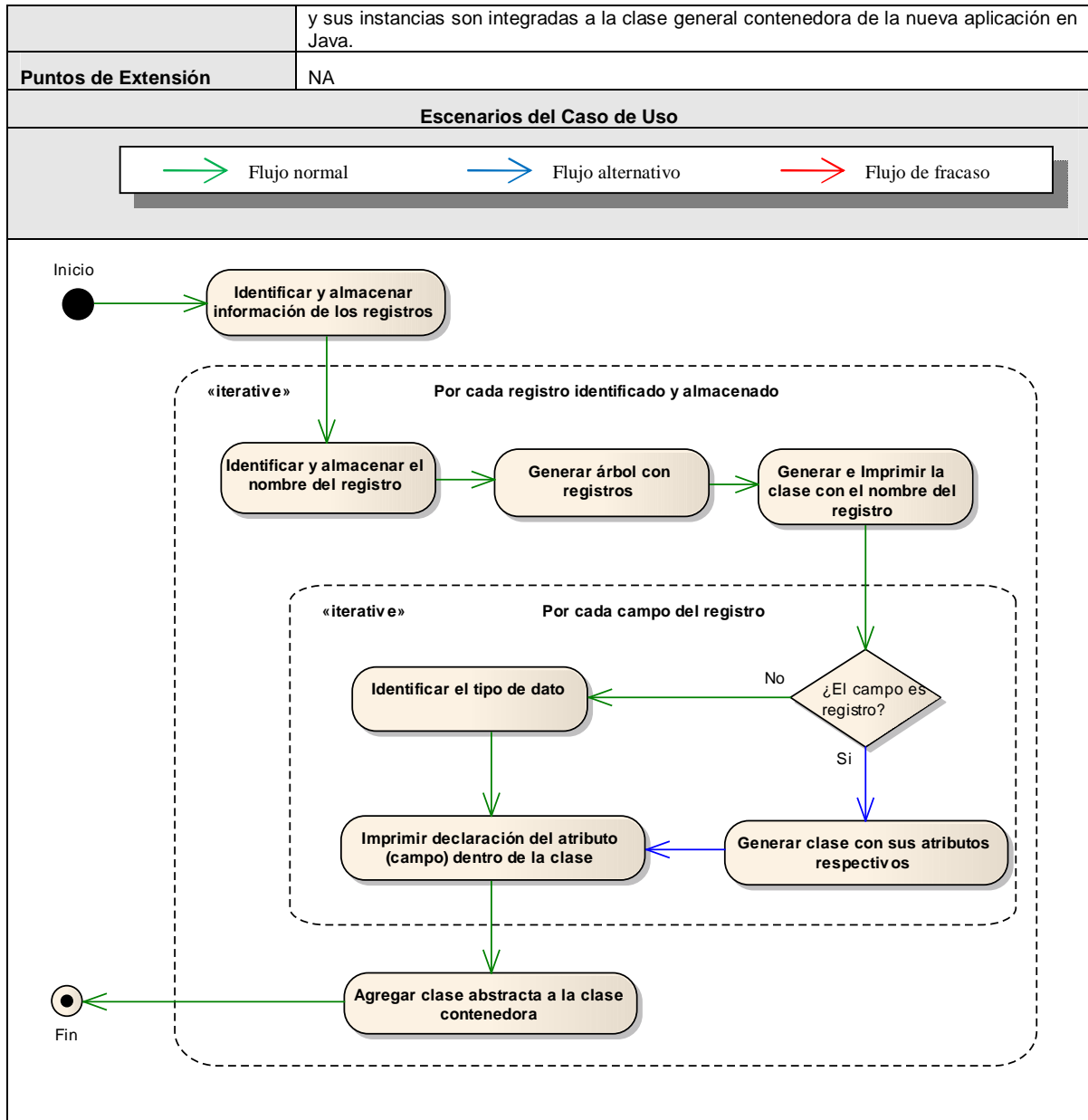
Nombre del Caso de Uso	SW-CU3.1:AnalizarCodigoCobol
Descripción	El objetivo del caso de uso es llevar a cabo el análisis de cada una de las sentencias escritas en COBOL para determinar posteriormente qué tipo de transformación será aplicada a cada una de ellas.
Casos de Uso Relacionados	<ul style="list-style-type: none"> SW-CU3:TransformarCobol2Java
Requisitos Especiales	<ul style="list-style-type: none"> NA
Pre-condiciones	<ul style="list-style-type: none"> El archivo con el código fuente a traducir se encuentra cargado en el sistema. El código fuente se encuentra libre de defectos de compilación y significado.
Post-condiciones	La información de las instrucciones identificadas se encuentran almacenadas para su futura transformación, de acuerdo a la instrucción.
Puntos de Extensión	<ul style="list-style-type: none"> SW-CU3.2:CnvDatosSimples SW-CU3.3:CnvEstructuras2Clase SW-CU3.4:CnvSentenciasControl SW-CU3.5:CnvProcedimientos SW-CU3.6:CnvSntAccesoBD
Escenarios del Caso de Uso	
<div style="display: flex; justify-content: space-around; align-items: center;"> <div style="display: flex; align-items: center;"> ➔ Flujo normal </div> <div style="display: flex; align-items: center;"> ➔ Flujo alternativo </div> <div style="display: flex; align-items: center;"> ➔ Flujo de fracaso </div> </div>	



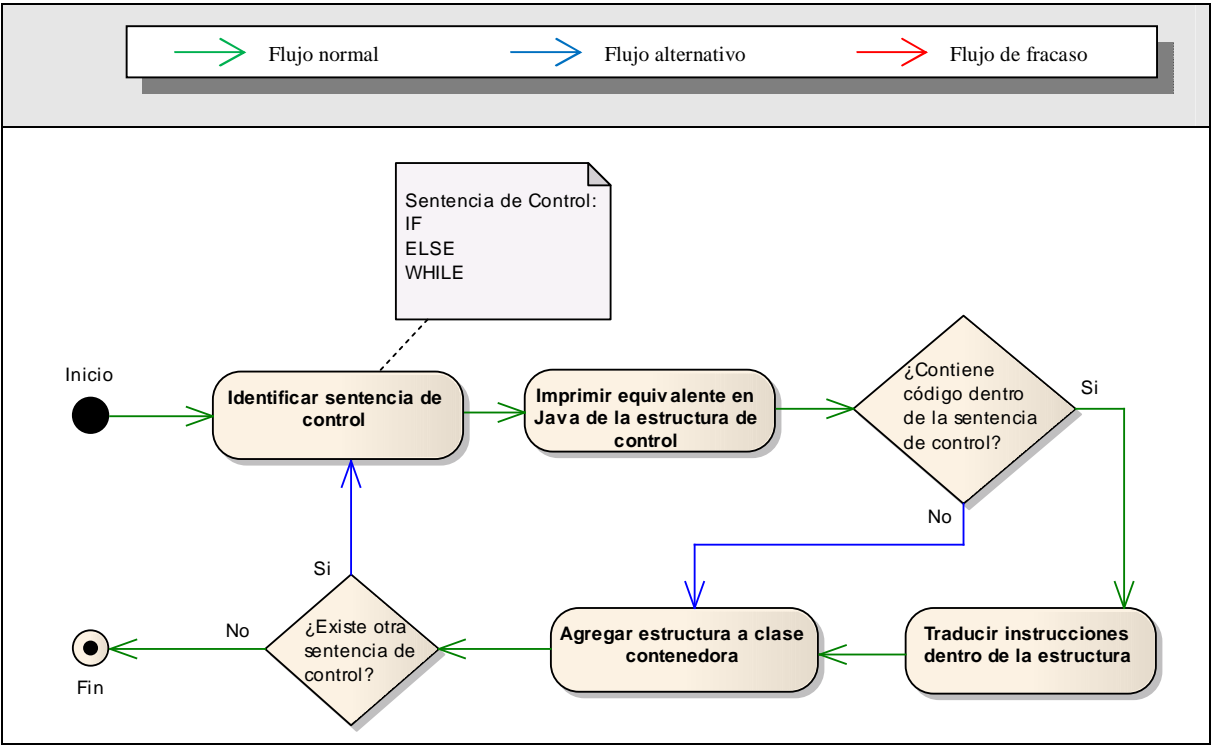
Nombre del Caso de Uso	SW-CU3.2:CnvDatosSimples
Descripción	El objetivo del caso de uso es llevar a cabo la traducción de datos y sentencias simples en COBOL, se identifican como datos y sentencias simples todos aquellos que tengan directamente una instrucción equivalente en el lenguaje Java. Por ejemplo, las instrucciones "Add", "Subtract", "Multiply", "Divide", etc. Además de los tipos de datos numéricos, alfabéticos y alfanuméricos.
Casos de Uso Relacionados	<ul style="list-style-type: none"> SW-CU3.1:AnalizarCodigoCobol
Requisitos Especiales	<ul style="list-style-type: none"> NA
Pre-condiciones	<ul style="list-style-type: none"> La información extraída del análisis (como nombre de las variables y sus características)se encuentra disponible en tablas o registros para llevar a cabo la traducción.
Post-condiciones	Los datos y las sentencias simples identificados en la aplicación de COBOL son traducidos e integrados a la clase general contenedora de la nueva aplicación en Java.
Puntos de Extensión	NA



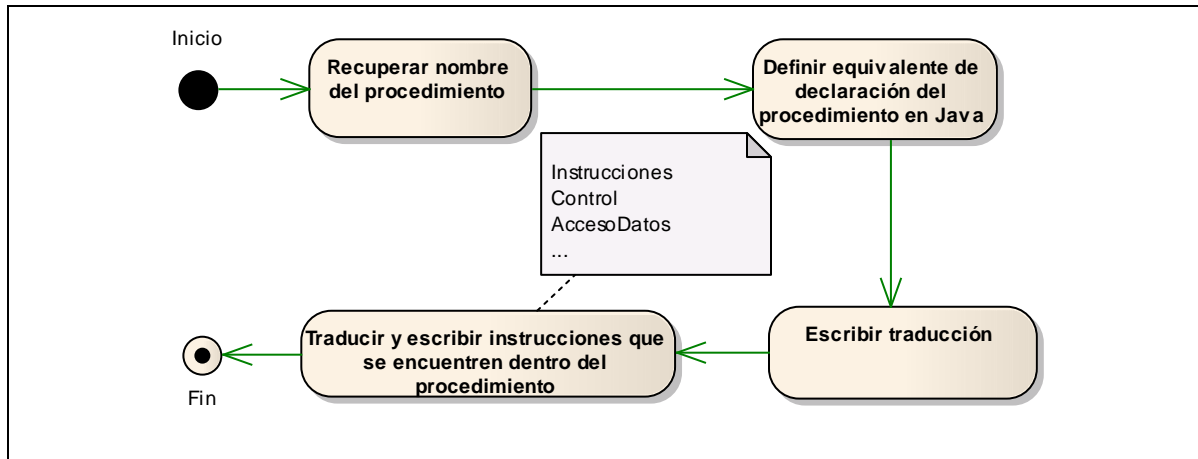
Nombre del Caso de Uso	SW-CU3.3:CnvEstructuras2Clase
Descripción	El objetivo del caso de uso es transformar los datos estructurados (estructuras) identificadas en el código COBOL a clases en Java que contengan atributos y métodos de acceso a ellos.
Casos de Uso Relacionados	<ul style="list-style-type: none"> SW-CU3.1:AnalizarCodigoCobol
Requisitos Especiales	<ul style="list-style-type: none"> NA
Pre-condiciones	<ul style="list-style-type: none"> La información extraída del análisis se encuentra disponible en tablas o registros para llevar a cabo la traducción.
Post-condiciones	Las estructuras identificadas en la aplicación de COBOL son traducidas a clases en Java



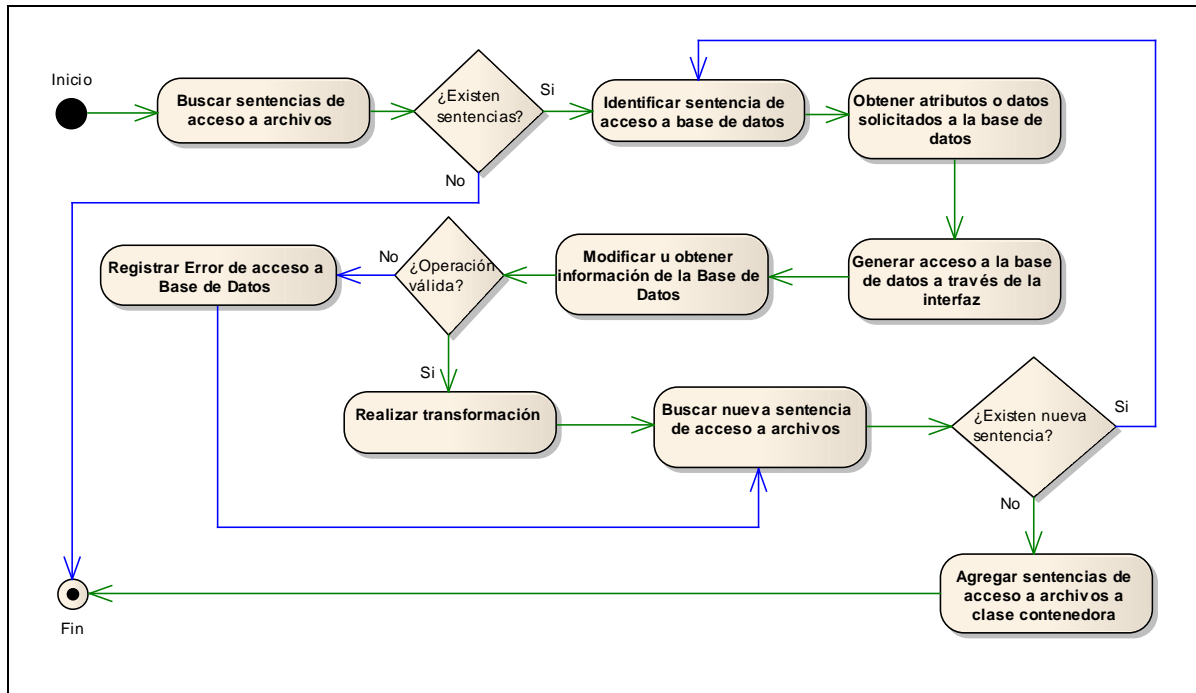
Nombre del Caso de Uso	SW-CU3.4:CnvSentenciasControl
Descripción	El objetivo del caso de uso es traducir las sentencias de control identificadas dentro del código fuente escrito en COBOL al lenguaje de programación Java.
Casos de Uso Relacionados	<ul style="list-style-type: none"> SW-CU3.1:AnalizarCodigoCobol
Requisitos Especiales	<ul style="list-style-type: none"> NA
Pre-condiciones	<ul style="list-style-type: none"> La información extraída del análisis se encuentra disponible en tablas o registros para llevar a cabo la traducción.
Post-condiciones	Las estructuras de control identificadas en la aplicación de COBOL son traducidas e integradas a la clase general contenedora de la nueva aplicación en Java.
Puntos de Extensión	NA
Escenarios del Caso de Uso	



Nombre del Caso de Uso	SW-CU3.5:CnvProcedimientos
Descripción	El objetivo del caso es aplicar una serie de pasos para llevar a cabo la traducción de procedimientos en COBOL al lenguaje de programación Java.
Casos de Uso Relacionados	<ul style="list-style-type: none">SW-CU3.1:AnalizarCodigoCobol
Requisitos Especiales	<ul style="list-style-type: none">NA
Pre-condiciones	<ul style="list-style-type: none">La información extraída del análisis se encuentra disponible en tablas o registros para llevar a cabo la traducción.
Post-condiciones	Los procedimientos identificados en la aplicación de COBOL son traducidos e integrados a la clase general contenedora de la nueva aplicación en Java.
Puntos de Extensión	NA
Escenarios del Caso de Uso	
<div><div><div>→ Flujo normal</div><div>→ Flujo alternativo</div><div>→ Flujo de fracaso</div></div></div>	

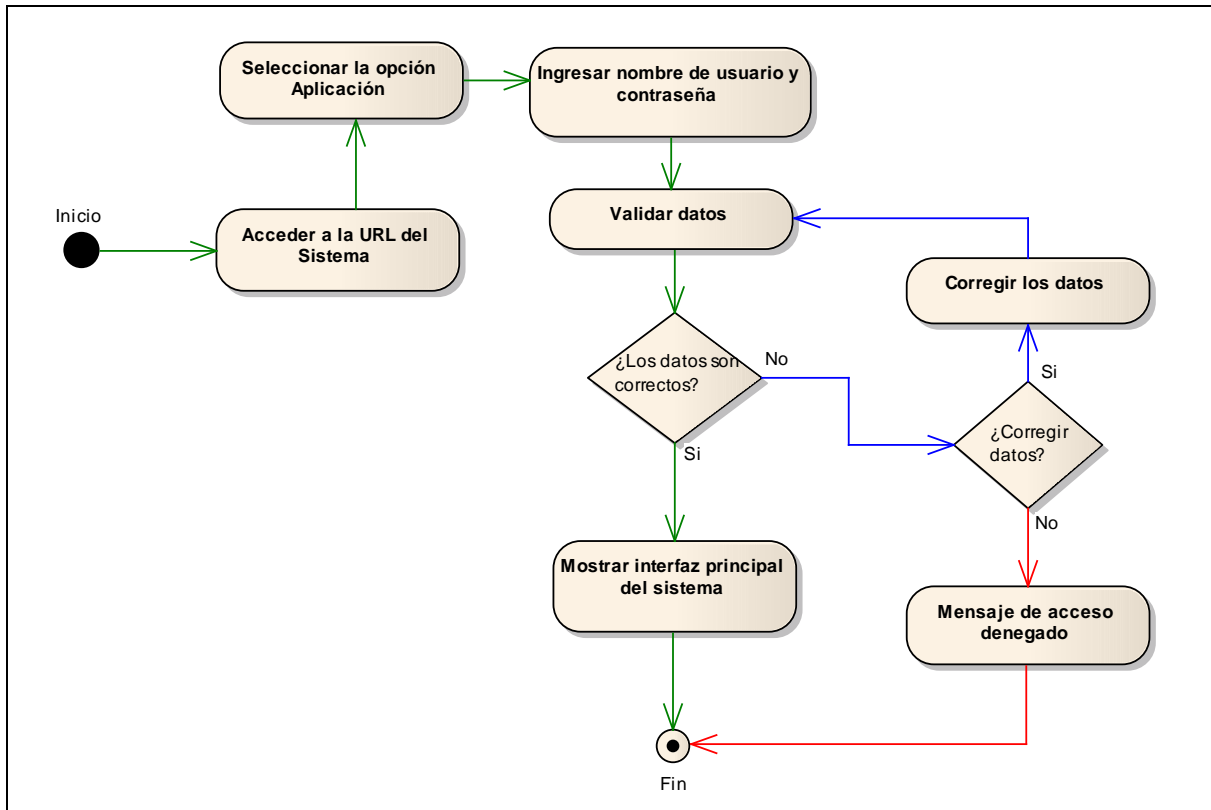


Nombre del Caso de Uso	SW-CU3.6:CnvSntAccesoBD
Descripción	El objetivo del caso de uso es llevar a cabo la transformación de las sentencias correspondientes al acceso al sistema de archivos en COBOL por instrucciones en Java de acceso a una Base de Datos.
Casos de Uso Relacionados	<ul style="list-style-type: none"> SW-CU3.1:AnalizarCodigoCobol
Requisitos Especiales	<ul style="list-style-type: none"> El caso de uso UC_1_TransformarCobol2SQL ha sido ejecutado exitosamente.
Pre-condiciones	<ul style="list-style-type: none"> La información extraída del análisis se encuentra disponible en tablas o registros para llevar a cabo la traducción. Las clases correspondientes a la interfaz con la nueva base de datos en SQL se encuentran generadas y son accesibles por el sistema.
Post-condiciones	Las sentencias correspondientes de acceso al sistema de archivos de COBOL son traducidas e integradas a la clase general contenedora de la nueva aplicación en Java.
Puntos de Extensión	NA
Escenarios del Caso de Uso	
<div style="display: flex; justify-content: space-around; align-items: center;"> <div style="display: flex; align-items: center;"> ➔ Flujo normal </div> <div style="display: flex; align-items: center;"> ➔ Flujo alternativo </div> <div style="display: flex; align-items: center;"> ➔ Flujo de fracaso </div> </div>	

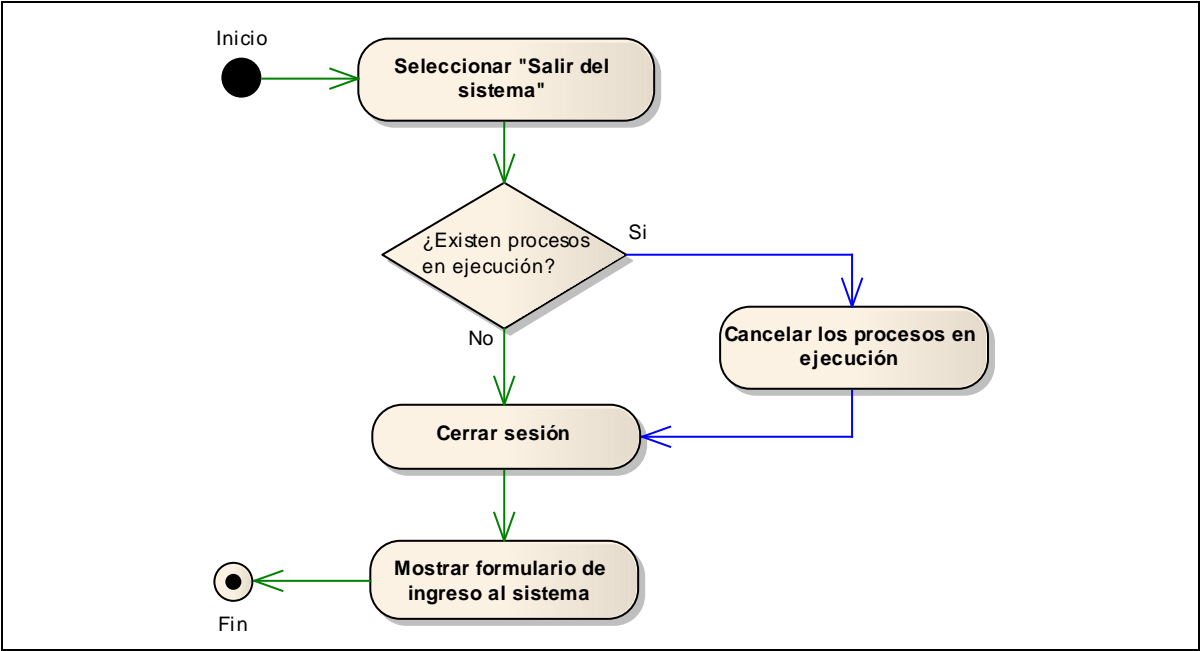


Casos de Uso del Cliente Web

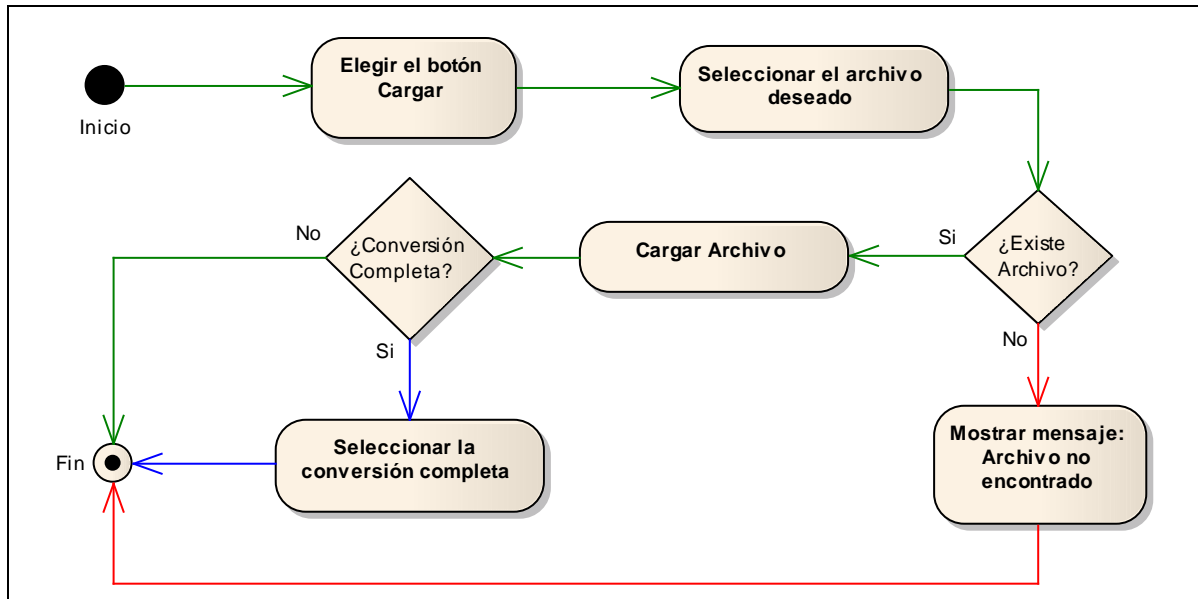
Nombre del Caso de Uso	CL-CU1:IngresarSistema
Descripción	El objetivo del caso de uso es acceder al sistema por medio de una cuenta de usuario (nombre y contraseña) y permitir al usuario realizar las operaciones de traducción de código COBOL a código Java.
Casos de Uso Relacionados	<ul style="list-style-type: none"> • NA
Requisitos Especiales	<ul style="list-style-type: none"> • NA
Pre-condiciones	<ul style="list-style-type: none"> • El usuario abrió un "browser" para acceder a la aplicación.
Post-condiciones	El usuario se encuentra en sesión para llevar a cabo las traducciones.
Puntos de Extensión	NA
Escenarios del Caso de Uso	
<div style="display: flex; justify-content: space-around; align-items: center;"> <div style="text-align: center;"> → Flujo normal </div> <div style="text-align: center;"> → Flujo alternativo </div> <div style="text-align: center;"> → Flujo de fracaso </div> </div>	



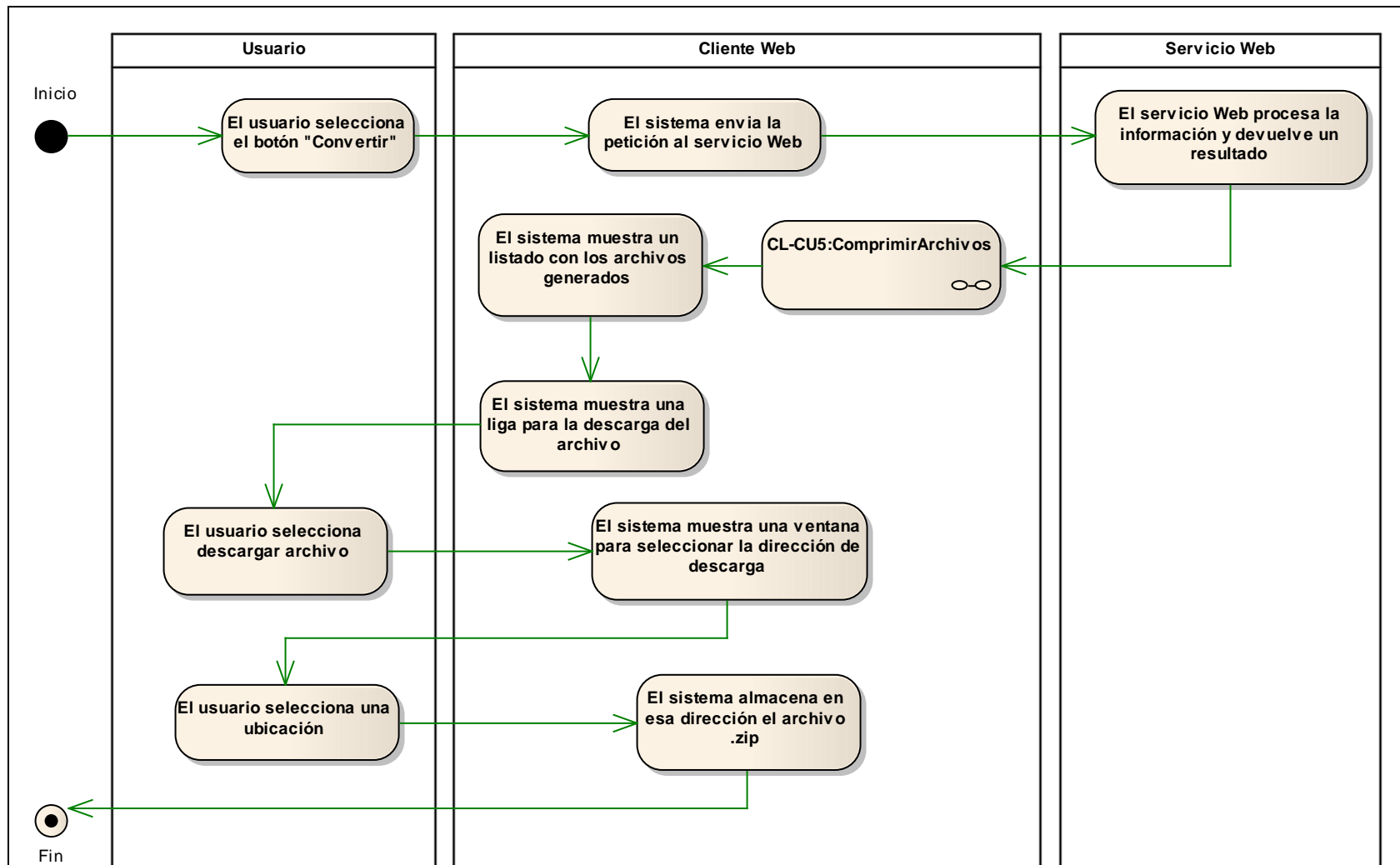
Nombre del Caso de Uso	CL-CU2:SalirSistema
Descripción	El objetivo del caso de uso es cerrar la sesión del usuario dentro del sistema.
Casos de Uso Relacionados	• NA
Requisitos Especiales	• NA
Pre-condiciones	El usuario se encuentra en sesión.
Post-condiciones	La aplicación muestra la pantalla de inicio de sesión.
Puntos de Extensión	NA
Escenarios del Caso de Uso	
<div style="display: flex; justify-content: space-around; align-items: center;"> <div style="display: flex; align-items: center;"> ➔ Flujo normal </div> <div style="display: flex; align-items: center;"> ➔ Flujo alternativo </div> <div style="display: flex; align-items: center;"> ➔ Flujo de fracaso </div> </div>	

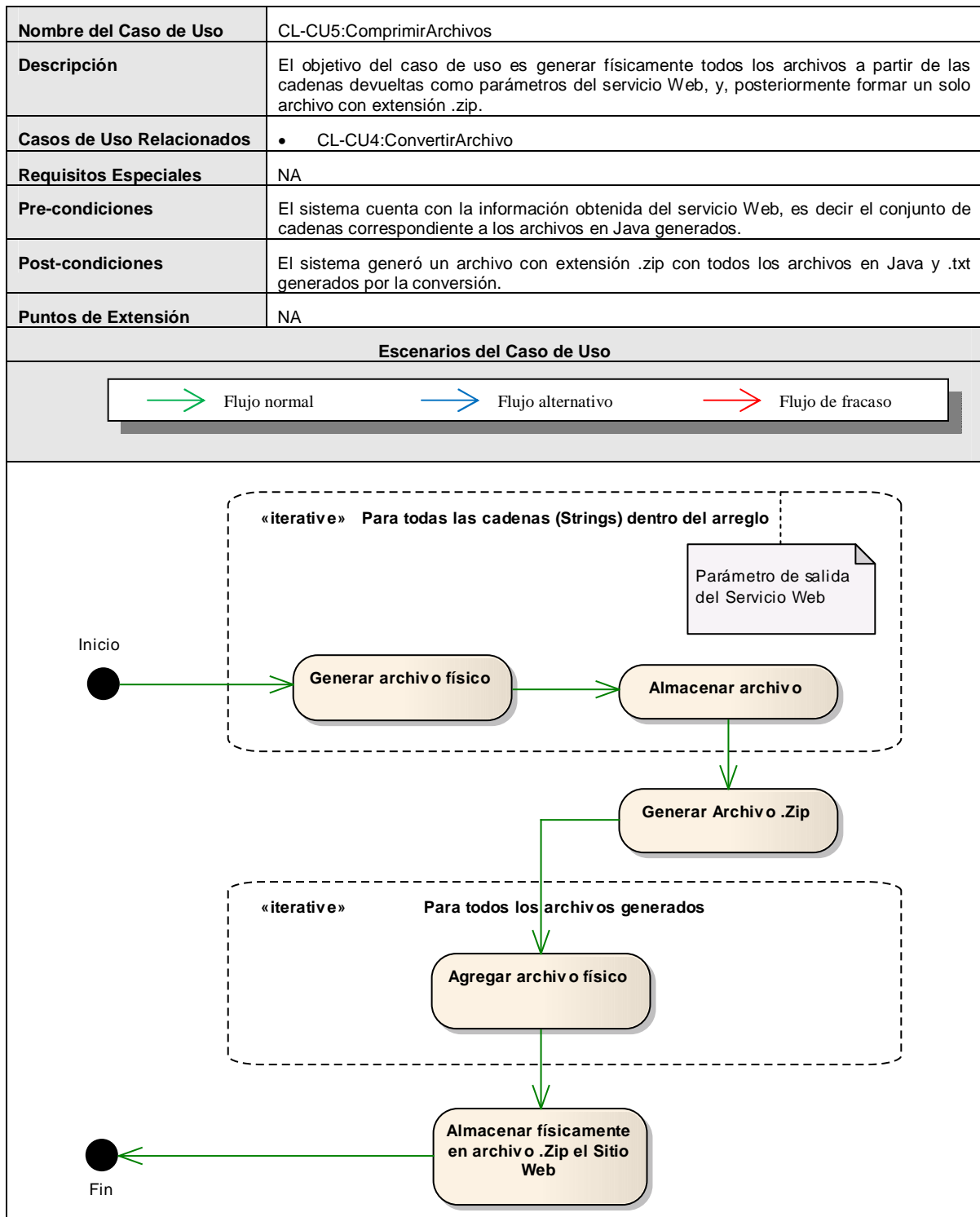


Nombre del Caso de Uso	CL-CU3:CargarArchivo
Descripción	El objetivo del caso de uso es subir un archivo con el código fuente en lenguaje COBL a memoria, con el fin de mantener lista la información para ser enviada al servicio Web y ser convertido a lenguaje Java.
Casos de Uso Relacionados	NA
Requisitos Especiales	NA
Pre-condiciones	El usuario se encuentra en sesión. El contenido del archivo es código en COBOL, el cuál ha sido validado previamente.
Post-condiciones	El archivo con el código fuente se encuentra cargado en memoria.
Puntos de Extensión	NA
Escenarios del Caso de Uso	
<div><div> Flujo normal</div><div> Flujo alternativo</div><div> Flujo de fracaso</div></div>	



Nombre del Caso de Uso	CL-CU4:ConvertirArchivo
Descripción	El objetivo del caso de uso es llevar a cabo la conversión de los archivos en Cobol a lenguaje Java.
Casos de Uso Relacionados	NA
Requisitos Especiales	El servicio Web deberá estar disponible.
Pre-condiciones	<ul style="list-style-type: none"> El usuario se encuentra en sesión. El archivo con el código fuente se encuentra cargado en memoria.
Post-condiciones	El sistema muestra una pantalla con la liga para descargar los archivos generados por la conversión.
Puntos de Extensión	<ul style="list-style-type: none"> CL-CU5:ComprimirArchivos
Escenarios del Caso de Uso	
<div style="display: flex; justify-content: space-around; align-items: center;"> <div style="display: flex; align-items: center;"> → Flujo normal </div> <div style="display: flex; align-items: center;"> → Flujo alternativo </div> <div style="display: flex; align-items: center;"> → Flujo de fracaso </div> </div>	





Anexo C. Descripción de Clases por Paquete

Descripción del Paquete “*Cliente*” de la Figura 31, en la sección 4.1.4.- Correspondiente al componente de interfaz de usuario.

Nombre de la clase	Descripción
AbsCliente	Clase principal abstracta que contiene el llamado al servicio Web.
CCcliente	Clase derivada de “AbsCliente” que contiene los métodos para obtener el código fuente y la manipulación de los parámetros devueltos por el servicio Web.

Descripción del Paquete “*Ensamble*” de la *Figura 32*, en la sección 4.1.4.- Correspondiente al componente traducción del sistema.

Nombre de la clase	Descripción
Util.StringFileReader	La clase se encarga de leer un archivo y devolver el contenido como una cadena (String).
WebService.Convertir	Clase de entrada al servicio Web, a partir de un archivo en Cobol, se genera una lista de cadenas con los archivos en java derivados de la traducción de COBOL a Java y SQL

Descripción del Paquete “*COB2SQL*” de la *Figura 33* en la sección 4.1.4.- Correspondiente a la traducción del sistema de archivos de COBOL.

Nombre de la clase	Descripción
AbsCampoGeneral	Clase abstracta que contiene los meta datos y métodos encargados de manejar los campos en una estructura de archivos en COBOL.
AbsCobol2SQL	Clase abstracta encargada de llevar a cabo la traducción del sistema de archivos.
AbsDato	Clase abstracta que contiene los meta datos y métodos de los datos encontrados en el código fuente y están relacionados con los campos de la definición de archivos.
AbsFD	Clase abstracta que contiene los meta datos y métodos correspondientes a la definición de los archivos en COBOL.
AbsListaNombreDatos	Clase abstracta encargada de manipular una lista con los nombres de los datos a los que son asignadas las variables de los campos en COBOL.
AbsOcurrencia	Clase abstracta que contiene los meta datos y métodos encargados de llevar el control de las variables conforme son encontradas.
AbsValorLista	Clase abstracta que contiene los meta datos y métodos encargados de llevar el control de la lista de valores de cada campo en COBOL.
CCampoGeneral	Clase concreta derivada de “AbsCampoGeneral”. Contiene los meta datos de un campo en COBOL. Implementa las funciones <i>setNombreCampo()</i> , <i>getNombreCampo()</i> , <i>getNumeroNivel()</i> y <i>setNumeroNivel()</i> .
CCobol2SQL	Implementación de la clase abstracta “AbsCobol2SQL”, definiendo

	el comportamiento de los métodos <i>generarScript()</i> y <i>generarInterfaz()</i> .
CDato	Implementación de la clase abstracta "AbsDato", definiendo el comportamiento de los métodos <i>getValor()</i> y <i>setValor()</i> .
CFD	Implementación de la clase abstracta "AbsFD", definiendo el comportamiento de los métodos <i>getDescripcion()</i> y <i>setDescripcion()</i> .
CListaNombreDatos	Implementación de la clase abstracta "AbsListaNombreDatos", definiendo el comportamiento de los métodos <i>Ordenar()</i> y <i>esAscendente()</i> .
COcurrencia	Implementación de la clase abstracta "AbsOcurrencia", definiendo el comportamiento de los métodos <i>getValorInicial()</i> , <i>setValorInicial()</i> , <i>getValorFinal()</i> y <i>setValorFinal()</i> .
ValorLista	Implementación de la clase abstracta "AbsValorLista", definiendo el comportamiento de los métodos <i>getInicio()</i> , <i>setInicio()</i> , <i>getFinal()</i> y <i>setFinal()</i> .

Descripción del Paquete "COBOL2JAVA" de la *Figura 34*, en la sección 4.1.4.- Correspondiente a la conversión de código en COBOL a Java.

Nombre de la Clase	Descripción
C2J	Clase que contiene los atributos y métodos encargados de llevar el control de la conversión.
Convertidor	Clase que contiene los atributos y métodos encargados de realizar cada una de las traducciones de las diferentes sentencias en COBOL. En esta clase sólo son definidos los métodos, pero no implementados. Por ejemplo, el método <i>convertir()</i> , que en cada una de las clases concretas se enfoca a realizar la traducción de los diferentes tipos de instrucciones.
TablaSimbolos	Clase que contiene los atributos y métodos encargados de manipular la tabla de símbolos del analizador. Implementa los métodos <i>agregaVariable()</i> , <i>eliminaVariable()</i> y <i>modificaVariable()</i> .
CnvDatos	Clase abstracta que contiene métodos usados por las clases derivadas, durante la traducción de los diferentes tipos de datos (Nivel 77, registros, acceso a archivos).
CnvDatosNivel77	Clase concreta encargada de llevar a cabo la traducción de los datos

	de nivel 77 en COBOL a datos simples en Java.
CnvDatosVariablesAA	Clase concreta encargada de llevar a cabo la traducción de los datos de acceso a archivos de Cobol a clases en Java y SQL.
CnvDatosRegistros	Clase concreta encargada de llevar a cabo la traducción de los registros de Cobol a Clases en Java.
CnAcceptstm	Clase concreta encargada de llevar a cabo la traducción a Java del verbo "Accept".
CnvOpenStm	Clase concreta encargada de llevar a cabo la traducción a Java del verbo "OPEN".
CnvCloseStm	Clase concreta encargada de llevar a cabo la traducción a Java del verbo "Close".
CnvDeleteStm	Clase concreta encargada de llevar a cabo la traducción a Java del verbo "DELETE".
CnvReadStm	Clase concreta encargada de llevar a cabo la traducción a Java del verbo "READ".
CnvWriteStm	Clase concreta encargada de llevar a cabo la traducción a Java del verbo "WRITE".

Descripción del Paquete “Analizador” de la *Figura 35*, en la sección 4.1.4.- Correspondiente al análisis de cada una de las sentencias en COBOL para su posterior conversión a Java.

Nombre de la clase	Descripción
Analizador	Analizador sintáctico.
AnalizadorCharStream	Clase para el flujo de entrada de caracteres a analizar.
AnalizadorTokenManager	Es el analizador lexicográfico.
Error	Clase base para llevar mantener el control de errores del analizador.
Exceptions	Clase base encargada controlar las excepciones en general de la aplicación.
ParserExceptions	Clase que implementa los errores sintácticos.
Token	Clase que implementa el objeto a través del cual se comunica el analizador léxico y el sintáctico.
TokenMgrError	Clase que implementa los errores lexicográficos
AnalizadorConstants	Interfaz que asocia un código a cada token. Además, almacena de forma de String el nombre definido en la gramática a cada

	token, con el objeto de que el analizador sintáctico pueda emitir mensajes de error claros y explicativos.
--	--

Descripción del Paquete “*InterfazJava2SQLr*” de la *Figura 36*, en la sección 4.1.4.- Correspondiente al uso de base de datos en SQL como sustitución del acceso a archivos en COBOL.

Nombre de la clase	Descripción
AbsCampo	Clase abstracta que contiene los atributos y métodos encargados de manipular los campos en una base de datos SQL.
AbsOperacioneSQL	Clase abstracta que contiene los atributos y métodos encargados de llevar a cabo las operaciones definidas en SQL.
AbsRegistro	Clase abstracta que contiene los atributos y métodos encargados de manipular un registro en una base de datos SQL.
AbsTablaSQL	Clase abstracta que contiene los atributos y métodos encargados de manipular una tabla en una base de datos SQL.
CBDSQL	Clase principal contenedora de la interfaz entre la base de datos y la aplicación en Java. Contiene la estructura del sistema de archivos definido en el código fuente (COBOL).
CCampoSQLCadena	Clase concreta derivada de “AbsCampo” correspondiente al tipo de dato cadena. Implementa los métodos <i>getValor()</i> y <i>setValor()</i> .
CCampoSQLDoble	Clase concreta derivada de “AbsCampo” correspondiente al tipo de dato doble. Implementa los métodos <i>getValor()</i> y <i>setValor()</i> .
CCampoSQLEntero	Clase concreta derivada de “AbsCampo” correspondiente al tipo de dato entero. Implementa los métodos <i>getValor()</i> y <i>setValor()</i> .
CCampoSQLFlotante	Clase concreta derivada de “AbsCampo” correspondiente al tipo de dato flotante. Implementa los métodos <i>getValor()</i> y <i>setValor()</i> .
CCloseSQL	Clase concreta derivada de “AbsOperacioneSQL” encargada implementar la operación de cierre de conexiones a la base de datos en SQL con la implementación del comportamiento del método <i>ejecutar()</i> .
CConexionSQL	Clase encargada de mantener la conexión entre la aplicación Java y la base de datos en SQL.
CConstructorSQL	Clase encargada de definir la estructura de la nueva base de datos de acuerdo a la definición de archivos en COBOL.

CDeleteSQL	Clase concreta derivada de “AbsOperacioneSQL” encargada implementar la operación de eliminación de registros en base de datos SQL con la implementación del comportamiento del método <i>ejecutar()</i> .
CErrorSQL	Clase encargada de llevar el control de errores de las operaciones a la base de datos SQL con la implementación del comportamiento del método <i>ejecutar()</i> .
CInsertSQL	Clase concreta derivada de “AbsOperacioneSQL” encargada implementar la operación de inserción de registros a una base de datos en SQL con la implementación del comportamiento del método <i>ejecutar()</i> .
COpenSQL	Clase concreta derivada de “AbsOperacioneSQL” encargada implementar la operación de apertura de conexiones a la base de datos en SQL con la implementación del método <i>ejecutar()</i> .
CRegistroSQL	Clase concreta derivada de “AbsRegistro” y que implementa la manipulación de registros en una base de datos SQL, definiendo el comportamiento de los métodos <i>setNoCampos()</i> y <i>getNoCampos()</i> .
CSelectSQL	Clase concreta derivada de “AbsOperacioneSQL” encargada implementar la operación de consultas en una base de datos SQL.
CTablaSQL	Clase concreta derivada de “AbsTablaSQL”, implementa la funcionalidad de una tabla en base de datos SQL, definiendo el comportamiento de los métodos <i>ObtenerLlave()</i> y <i>ObtenerCampo()</i> .
CUpdateSQL	Clase concreta derivada de “AbsOperacioneSQL” encargada de implementar la operación de actualización de registros en una base de datos SQL con la implementación del comportamiento del método <i>ejecutar()</i> .

Referencias

- [1]. Aho V. Alfred, Lam Monica, Sethi Ravi, Ullamn D. Jeffrey, Compiladores Principios Técnicas y Herramientas, 1a Edición, México.1998.
- [2]. Andreas S. Philippakis. COBOL Estructurado. Tercera Edición. México 1988.
- [3]. Andrew Parkin and Richard Yorke. COBOL FOR STUDENTS. Fourth Edition. London 1996.
- [4]. Cauldwell Patric, Chawla Rajesh, Chopra Vivek, et al. Profesional. Servicios Web XML. Anaya Multimedia. España 2002.
- [5]. Cooper James W. Java Design Patterns. A Tutorial. Addison-Wesley. USA 2000.
- [6]. Falkner Jayson, Galbraith Ben, Irani Romin. Fundamentos. Desarrollo Web con JSP. Anaya Multimedia. España 2002.
- [7]. Flanagan David. Java en Pocas Palabras. Referencia al Instante. McGraw-Hill Editores. México 1998.

- [8]. Flores Pichardo, Mireya, Tesis de Maestría: "Estudio Empírico para la Transformación Gramatical de Código en Lenguaje COBOL hacia Lenguaje JAVA", CENIDET, 2006.
- [9]. Fowler Martin, con Kendall Scott. UML GOTA a GOTA. Addison Wesley. México 1999.
- [10]. Gálvez, Rojas Sergio, Mora, Mata Miguel. Java a Tope. Traductores y Compiladores con Lex/Yacc, JFlex/Cup y JavaCC. Edición Electrónica. Universidad de Malaga. España. 2005.
- [11]. Grand, Mark. Patterns in Java. A catalog of Reusable Design Patterns Illustrated with UML. Willy. USA 1998.
- [12]. Handel, Yevsei, Degtyar Boris. COBOL With Compiler. WROX. UK 1994.
- [13]. López Rodríguez, Ariel. "Estudio para la Transformación Gramatical de Instrucciones de Acceso a una Base de Datos Escritas en Lenguaje COBOL hacia Instrucciones Escritas en Lenguaje SQL", Tesis de Maestría, CENIDET, 2005.
- [14]. Meyer, Bertrand. Object-Oriented Software Construction. Prentice Hall International Series in Computer Science. USA 1988.
- [15]. Santaolaya Salgado René, Modelo de Representación de Patrones de Código para la Construcción de Componentes Reusables. Tesis de Doctorado. Departamento de Ciencias Computacionales, Centro de Investigación en Computación, IPN, 2003.
- [16]. Shari Lawrence Pfleeger. Ingenieria de Software. Teoría y Práctica. Prentice Hall. Buenos Aires Argentina 2002.
- [17]. Silberschatz, Korth, Sudarshan. Fundamentos de Base de Datos. 5ª Edición. España 2006.
- [18]. Sommerville, Ian. Ingenieria de Software. Addison Wesley. 6a Edición. México 2002.
- [19]. <http://www.bphx.com/en/Solutions/LegacyModernization/Language%20Migration/Pages/COBOL.aspx>
- [20]. http://www.cobolmining.com/download/pdf/SM_Automatic_Modernization_Brochure.pdf
- [21]. <http://www.coboltojava.com/ctoj-ratio-1.html>
- [22]. <http://www.jazillian.com/index.html>.
- [23]. <http://www.mpsinc.com/cob2j.html>
- [24]. <http://www.netcobol.com/products/windows/dconv.htm>
- [25]. <http://www.transtools.com/products/en/caraveli.htm>
- [26]. W3C Working Group Note: Web Services Architecture. [Citado 2010-09-14]. Disponible en línea: <http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/#what>