Universidad de las Ciencias Informáticas
Facultad 3

Título: Creación de un sistema virtual de Transmisión de Datos para el apoyo a la docencia en la asignatura de Teleinformática I

Trabajo de Diploma para optar por el título de

Ingeniero en Ciencias Informáticas

Autores:

Carlos Matos Carbonell

Arnolis Rodríguez del Valle

Tutor: Ing. Abel Arias Hernández

Co-tutor: Alberto Limia Navarro

Ciudad de la Habana, Junio del 2008

DECLARACIÓN DE AUTORÍA

Declaramos ser autores de la presente tesis y reconocemos a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

Para que así conste firmo la presente a los ____ días del mes de _____ del año _____.

Carlos Matos Carbonell

_____
Firma del Autor

Arnolis Rodríguez del Valle

_____
Firma del Autor

Ing. Abel Arias Hernández

_____

Firma del Tutor

AGRADECIMIENTOS

A nuestro tutor Abel Arias Hernández y a nuestro Co tutor Alberto Limia Navarro por ayudarnos incondicionalmente en todo momento y darnos todo el apoyo necesario cuando nos hizo falta. A nuestros padres que son y serán siempre nuestra principal inspiración para realizar cualquier labor, guiándonos por sus pensamientos y enseñanzas, principales armas para lograr que hoy en día este trabajo haya cumplido con los objetivos trazados.

DEDICATORIA

A nuestros por padres, por confiar en nosotros en todo momento, por ser partícipes de educarnos e impulsarnos a llegar tan lejos, por apoyarnos incondicionalmente en nuestras decisiones y siempre que lo necesitamos, brindándonos los ánimos y fuerzas necesarios para seguir adelante y no claudicar nunca.

RESUMEN

Entre los temas impartidos en Teleinformática I están los procesos que intervienen en la transmisión de datos, estos no son asimilados correctamente por los estudiantes, provocando que presenten problemas a la hora de realizar los ejercicios orientados, influyendo negativamente en las evaluaciones y la promoción. Con la intención de mejorar los resultados de la asignatura se propone la idea de construir una herramienta que sirva de apoyo a los estudiantes y profesores para comprobar los resultados de los ejercicios. Para la realización de la aplicación se hace un estudio de algunas herramientas existentes que realizan procesos similares, metodologías de desarrollo, lenguajes de programación y sus IDEs. Se muestran los requisitos funcionales con que debe cumplir el software, dados por los profesores del Departamento de Sistemas Digitales. Se realizan las descripciones de los casos de uso. En el diseño se crearon los diagramas de clases pertinentes por caso de uso, además de sus respectivos diagramas de iteración, se seleccionó como patrón arquitectónico el de Filtros y Tuberías, siendo un estilo ideal para realizar transformaciones de datos, dividiéndolos en pasos sucesivos. Como parte de la implementación se explica el estándar de codificación creado; también se muestra el diagrama de componentes. Se les realizaron pruebas de unidad al código mediante el framework JUnit, el cual viene integrado junto al NetBeans 6.0, quedando demostrado el correcto funcionamiento del Simulador PSTD.

TABLA DE CONTENIDOS

INDICE DE FIGURAS

INDICE DE TABLAS

INTRODUCCIÓN

Antecedentes

La especie humana es de carácter social, es decir, necesita de la comunicación, pues de otra manera cada ser humano sería un ente aislado. Así, desde los inicios de la especie, la comunicación fue evolucionando hasta llegar a la más sofisticada tecnología, para lograr acercar espacios y tener mayor velocidad en el proceso.

En los años 3500 AC solo había comunicación a partir de signos abstractos dibujados en papel hecho de hojas de árboles; hacia 1184 AC ya se podían transmitir mensajes a distancia con señales de fuego, el antiguo imperio Romano y Griego poseían muy buenos sistemas de este tipo, hacia los años 500 AC dos ingenieros de Alejandría (Kleoxenos y Demokleitos) usaban un sistema de recepción y transmisión de información solo en la noche, el sistema constaba de dos caminos separados por una colina, dependiendo de cuantas antorchas y como fueran acomodadas en la colina el mensaje podía ser leído (para el mensaje “One hundred Cretans have deserted" fueron utilizadas 173 antorchas y la transmisión duró alrededor de 1 hora y media). Pero quizás uno de los primeros intentos de telecomunicaciones o transmisión de información a largas distancias fue la maratón que consistía en que una persona llevaba un mensaje de un sitio a otro corriendo a través de kilómetros de distancia. Luego nacieron otras formas de comunicación donde las personas se situaban en sitios altos y transmitían la información a otros a través de gestos hechos por el movimiento de sus brazos.

En áreas selváticas donde se dificultaba obtener línea de vista para transmisión de información, desde sitios altos, fueron desarrollados los telégrafos de tambor, la idea era transmitir la información a través de sonidos que emanaban de un tambor hecho con madera de los árboles para los nativos de África, Nueva Guinea y América, mientras que en China usaban el conocido Tamtam que era un gran plato metálico creado para transmitir información audible con algunos toque de un martillo sobre él.

Posteriormente el ser humano fue desarrollándose, inventado e investigando dando origen a nuevas técnicas y tecnologías para una mejor comunicación, reflejándose fuertemente desde las primeras máquinas programables manualmente (máquina diferencial de Babbage) o con procedimientos electrónicos (ENIAC, con tubos al vacío, en 1947), actualmente con las potentes computadoras digitales que se han introducido en prácticamente todas las áreas de la sociedad (industria, comercio, educación, comunicación, transporte y otros sectores).

Así, progresivamente el hombre fue realizando nuevos y novedosos descubrimientos que marcaban pautas para un mejor desarrollo en las comunicaciones, ejemplo de esto es como a lo largo de los últimos años se ha podido experimentar una serie de cambios tecnológicos a favor de las tareas

diarias. Los cambios van desde las aplicaciones más sencillas y cotidianas como sería la telefonía móvil, boletos del metro, tarjetas de crédito, hasta grandes redes de información, desarrollo aeroespacial, comunicación inalámbrica. Todo esto ha proporcionado una mejor comunicación del hombre, no importa la distancia física que exista para realizar diversas actividades como son: compras online, participación en foros, envío de correos electrónicos, entre otras a través de diferentes medios como la red inalámbrica o por cable, celulares que utilizan los satélites y otros dispositivos que cada año mejoran su velocidad de transmisión y procesamiento de información.

En la actualidad la información juega un papel importante en la sociedad, la gran cantidad de información almacenada por la humanidad, junto a la incapacidad de almacenarla en un único lugar físico hace necesaria la transmisión de la información de un lugar a otro constantemente. En la Universidad de las Ciencias Informáticas (Uci) se imparte la asignatura de Teleinformática I en tercer año de la carrera, la cual trata la transmisión de datos, explicando los procesos que intervienen de manera teórica.

Situación problemática:

Esta asignatura tiene un carácter teórico, se imparten muchos contenidos que los estudiantes no lo asimilan correctamente, influyendo esto de forma negativa en los conocimientos adquiridos, la calidad de las notas y la promoción al finalizar el semestre. Reciben una serie de algoritmos y métodos que intervienen en la transmisión de datos que presentan complejidad a la hora de desarrollarlo, y no cuentan con una herramienta que sea capaz de ayudarles a comprobar y verificar los ejercicios que realizan en las clases prácticas y sus estudios independientes.

Problema:

¿Cómo representar los procesos que intervienen en la transmisión de datos mediante un software educativo?

Hipótesis:

Si se desarrolla una aplicación capaz de mostrar los procesos de transmisión de datos de forma visual y metodológica, el departamento contará con un software educativo que servirá de apoyo a la docencia.

Objeto de Estudio:

Teoría de la información y los softwares educativos.

Campo de Acción:

Software simulador de procesos de transmisión de datos.

Objetivo:

Construir un software capaz de realizar y mostrar los procesos de transmisión de datos impartidos en la asignatura de Teleinformática I.

Tareas de Investigación:

1. Investigar herramientas que simulan la transmisión de datos.
2. Investigar acerca de las diferentes metodologías de desarrollo.
3. Investigar acerca del lenguaje de programación.
4. Investigar el uso de IDE de desarrollo.
5. Crear el diseño del software.
6. Implementar algoritmos de cifrado, codificación y compresión.

El trabajo está formado por cuatro capítulos donde se abarcan los siguientes temas:

En el capítulo 1: Se tratan los temas que fue necesario investigar para mostrar la importancia de la transmisión de la información a modo general y en específico en la Uci, además de explicar los diferentes conceptos importantes en el tema. Se hace un estudio de las principales herramientas para la implementación de la aplicación, así como los posibles lenguajes de programación a utilizar, IDE para el lenguaje seleccionado y algunas de las metodologías de desarrollo de software.

En el capítulo 2: Se presentan las características del sistema propuesto, describiendo como este debe funcionar, o sea sus potencialidades y características esenciales. Se exponen los requisitos del sistema, tanto funcionales como no funcionales, los casos de uso, los actores y sus respectivas descripciones. También se muestran los prototipos de interfaz de usuario.

En el capítulo 3: Se obtienen los artefactos del flujo de trabajo de diseño, etapa fundamental en el desarrollo de un software. Se aborda el patrón arquitectónico utilizado, tuberías y filtros, muy útil en la construcción de una aplicación que contiene las características de la que se desea elaborar. Se explican los patrones de diseño utilizados y las ventajas que brindan.

En el capítulo 4: Los artefactos generados en el capítulo anterior sirven de entrada a la implementación. Es aquí donde se generan los diagramas de componentes y paquetes de implementación. Se propone el estándar de codificación utilizado. También se tratan las pruebas realizadas para comprobar las funcionalidades del sistema, quedando construido completamente el software al terminar el capítulo.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

Introducción

El desarrollo de las telecomunicaciones comenzó desde el año 1844 cuando Samuel Morse puso en marcha el primer sistema telegráfico confiable donde los símbolos utilizados eran puntos y rayas .Tres décadas después Alejandro Gram Bell comenzó la era de la transmisión analógica, en esta época los datos originales solo eran codificados y transmitidos, una vez llegados al receptor se decodificaban y se obtenían los datos originales nuevamente, así durante casi un siglo se fue desarrollando la tecnología de las computadoras acelerando las técnicas de transmisión de señales digitales.

A lo largo del desarrollo de la computación el hombre vio la necesidad de comunicarse a través de los ordenadores creando así diferentes tecnologías para la transmisión de la información como son los cables de comunicación (par trenzado, coaxial, fibra óptica), los módems, las interfaces de red, entre otros dispositivos que ayudan en el proceso de comunicación. Todo este desarrollo ha permitido que hoy en día las comunicaciones sean aún más rápidas y efectivas permitiendo la transmisión de grandes cantidades de información, entre ellas: imágenes, sonido, películas, programas, informes, documentos; eliminando técnicas viejas utilizadas en las telecomunicaciones. En la actualidad esta ha tenido un gran desarrollo debido a todos los logros en la rama de la computación. El avance logrado en las técnicas del envío y recepción de la información trajo consigo el aumento de los procesos que intervienen en un sistema de transmisión de datos, ya no bastaba con codificar la información y mandarla, ahora es necesario tener en cuenta criterios de seguridad, distancias y estándares, así como la necesidad de poder enviar varias señales por un mismo canal de transmisión.

Durante el desarrollo del presente capítulo se abordarán diferentes temas como los sistemas de transmisión de datos y los procesos que intervienen en este, se realizará un análisis de herramientas que existen en el mundo para simular a estos, además se tendrán en cuenta metodologías para el desarrollo de software, IDE de desarrollo, lenguajes de programación, así como el concepto de diseño, su importancia, y los patrones de diseño, seleccionando los más convenientes para desarrollar el sistema que se requiere.

1.1 Sistema de Transmisión de Datos

La transmisión de la información es la operación de enviar y recibir datos, mediante el empleo de determinados procesos para eliminar problemas de información innecesaria(ruido), erradicar riesgos de intercepción o modificación de un mensaje ya sea por un agente externo o el medio ambiente, recarga del medio al transmitir gran cantidad de información, corregir algunos errores que posea la

información entre otros eventos que podrían provocar que el proceso no se ejecute de forma efectiva y eficiente.

En la Figura 1 se puede apreciar un sistema de transmisión de datos, el cual está compuesto por un transmisor y un receptor por los que el mensaje enviado sufre transformaciones como resultado a diferentes procesos como la codificación, el cifrado, la multiplexación y la modulación, realizando luego el procedimiento inverso a estos, ayudando de una forma u otra a evitar problemas en la comunicación entre la fuente y el destino.

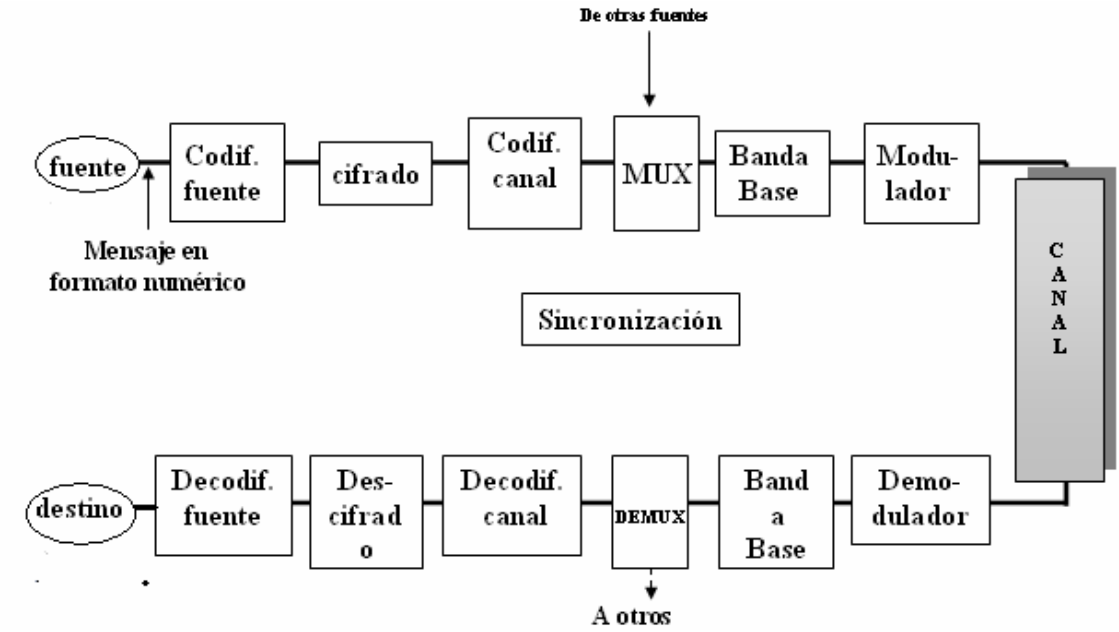


Figura 1: Sistema de Transmisión de Datos

Hoy en día un mensaje desde que se envía hasta que se recibe sufre cambios producto a la utilización de determinados procesos, que permiten que los sistemas actuales sean más seguros, eficientes y rápidos a la hora de establecer una comunicación entre un emisor y un receptor. De ellos solo se realizará un estudio de los que se imparten en clases de Teleinformática I.

Un mensaje cuando es enviado se codifica, cifra, comprime, multiplexa, y finalmente se modula para luego realizar los procesos inversos como decodificar, descifrar, descomprimir, demultiplexar y desmodular para que llegue a su destino sin que halla sufrido ningún cambio o alteración. A continuación se dará una explicación más detallada de estos procedimientos.

1.1.1 Codificación

El objetivo de la codificación es obtener una representación eficiente de los símbolos del alfabeto fuente, en la cual, para que sea eficiente es necesario tener un conocimiento de las probabilidades de cada uno de los símbolos del alfabeto. El dispositivo que realiza esta tarea es el codificador, este debe cumplir el requisito de que cada palabra de código debe decodificarse de forma única, tal que la secuencia original sea reconstruida perfectamente a partir de la secuencia codificada.

La codificación consiste en establecer una correspondencia entre cada uno de los símbolos de un alfabeto fuente y una secuencia de símbolos de un alfabeto destino. Al alfabeto destino se le denomina alfabeto código y a cada una de las secuencias de símbolos de este alfabeto que se corresponda con un símbolo del alfabeto fuente se denomina palabra de código. El alfabeto fuente contiene los símbolos originales que se quieren codificar. El alfabeto código contiene las palabras de código equivalentes en que se codificarán los símbolos originales. [1]

Entre los tipos de codificaciones están las eficientes y las redundantes, dentro de las primeras se encuentran la de Shanno Fano y la de Huffman, teniendo esta última el inconveniente de que si la fuente posee un conjunto enorme de símbolos el proceso se haría lento y consumiría mucha memoria su implementación. Dentro de las redundantes se tiene el código de Hamming y el Algebraico.

Luego de codificar la fuente se pasa al cifrado de esta para lograr la seguridad, el cual permite la confidencialidad y la integridad de la información; no es más que transformar un texto plano en uno totalmente distinto llamado criptograma, el cual es realizado mediante algoritmos matemáticos.

1.1.2 Cifrado

El cifrado es un método que permite aumentar la seguridad de un mensaje o de un archivo mediante la codificación del contenido, este consiste en transformar un texto plano o entendible a un texto sin ningún significado.[2]

Los algoritmos de encriptación pueden ser:

Sustitución: Esta técnica establece correspondencia con los símbolos del alfabeto del mensaje original con otros, que pueden ser del mismo o de uno distinto. Dentro de esta se encuentran el cifrado de César, el de Vigenére que es una generalización del anterior, el de Beaufort, y el de Vernam, los cuales son nombrados algoritmos clásicos de encriptación.[3]

Transposición: Consiste en intercambiar los símbolos del mensaje original colocándolos de una manera diferente. Dentro de esta técnica se encuentran las transposiciones por grupos, por series y por columnas, que pueden ser por clave o sin clave, estos también son considerados algoritmos clásicos.[3]

Cifrado simétrico: Cuando se emplea la misma clave en las operaciones de cifrado y descifrado, se dice que el criptosistema es simétrico o de clave secreta. Para ello se emplean algoritmos como IDEA, RC5, DES, TRIPLE DES, etcétera.[2]

Cifrado asimétrico: Cuando cada usuario crea un par de claves, una privada y otra pública, inversas dentro de un cuerpo finito. Lo que se cifra en emisión con una clave, se descifra en recepción con la clave inversa. La seguridad del sistema reside en la dificultad computacional de descubrir la clave privada a partir de la pública. Para ello, usan funciones matemáticas de un solo sentido o con trampa, ejemplos el RSA, Diffie-Hellman, entre otros.[2]

En la universidad se tienen en cuenta los algoritmos clásicos antes mencionados, pues son fáciles de aprender de manera práctica, además permiten explotar algunas de sus propiedades para entender mejor los algoritmos modernos de manera teórica, producto a su complejidad. Estos son los simétricos, y se emplearon con éxito hasta principio del siglo XX. Algunos se remontan incluso, como el algoritmo de César, a la Roma Imperial.

Luego del cifrado se pasa a la compresión que se encarga de reducir el tamaño de la información y transmitirla en menor tiempo, siendo menos la información que se envía.

1.1.3 Compresión

La compresión de datos es el proceso que trata de eliminar la redundancia dentro de un archivo (considerando cada byte como un mensaje elemental, y codificándolo con más o menos bits según su frecuencia de aparición).[3]

Según David Salomón la compresión de datos es el proceso de convertir un flujo de datos de entrada (datos originales) en un flujo de datos de salida (datos comprimidos) el cual es de menor tamaño que el de entrada.[4]

O sea la compresión de datos permite que la información se transmita a una velocidad superior a la velocidad de conexión real. Normalmente, los datos y, en particular, el texto y los gráficos, contienen secuencias repetidas de información idéntica. La compresión de datos funciona al sustituir muchos caracteres de información repetida por unos pocos caracteres y transmitir sólo una copia de las secuencias de datos repetidas, el mismo se clasifica en compresión con pérdida y sin pérdida.

Compresión sin pérdida: Es donde toda la información se guarda al comprimir sin perder el mínimo detalle, se usa cuando no se quiere perder dato de información. Dentro de estos se encuentran los compresores estadísticos y los basados en diccionarios.

Compresores estadísticos: Los que se basan en aprovechar una medida de información basada en probabilidades. Entre ellos se pueden citar a los compresores de tipo Huffman y Shanno Fano, los aritméticos, y los predictivos.

Compresores basados en diccionarios: Estos se basan en el uso de diccionarios estáticos o sea de conocer la estructura de los datos que se desean codificar. Entre ellos se puede citar al compresor RLE.

Compresión con pérdida: Esta compresión hace que se pierda cierta cantidad de información la cual no es importante desde cierto punto de vista porque no es permisible por los sentidos del ser humano. Por ejemplo al comprimir una imagen o una música o un video, esto es debido a que ni la vista ,ni el oído humano pueden percibir esta pequeña falta de información, en este tipo de compresión se tiene a la codificación por fuente, a la transformación Discreta por Coseno(DCT) y a la cuantización vectorial.

Para hacer un uso eficiente de las líneas de telecomunicaciones de alta velocidad se emplean técnicas de multiplexación, las cuales permiten que varias fuentes de transmisión compartan una capacidad de transmisión superior.

1.1.4 Multiplexación

Las facilidades y servicios de transmisión son caros. Es habitual que dos estaciones que se vayan a comunicar no utilicen toda la capacidad del enlace de datos. Por cuestiones de rendimiento, es conveniente compartir esa capacidad. El término genérico que alude a ese proceso de compartir es la multiplexación.[5]

En la práctica es necesario enviar simultáneamente una gran cantidad de mensajes diferentes por un medio de transmisión dado. El proceso de operación multicanal permite, mediante las técnicas llamadas de “multiplex” o “multiplexamiento”, combinar en el transmisor los mensajes de varias fuentes de información, transmitirlos como un solo bloque y luego separarlos en el receptor.[6]

Existen muchas formas de multiplexación según el sistema de comunicación, los más utilizados son por división de tiempo o TDM (Time division multiplexing),de frecuencia o FDM (Frequency-division multiplexing) y en código o CDM (Code division multiplexing), donde la multiplexación por división en frecuencias se puede usar con señales analógicas, de modo que se transmiten varias señales a través del mismo medio gracias a la asignación de una banda de frecuencia diferente para cada señal y la multiplexación por división en el tiempo síncrona se puede utilizar con señales digitales o con señales analógicas que transportan datos digitales. En esta forma de multiplexación, los datos procedentes de varias fuentes se transmiten en tramas repetitivas. [6]

Terminado el procedimiento de multiplexación el sistema pasa a la modulación que nace de la necesidad de transportar una información a través de un canal de comunicación a la mayor distancia y menor costo posible.

1.1.5 Modulación

Proceso de colocar la información contenida en una señal, generalmente de baja frecuencia, sobre una señal de alta frecuencia, donde la de baja frecuencia es la señal moduladora y la de alta frecuencia es denominada portadora.

En la transmisión de señales portadoras se utiliza una gran variedad de métodos de modulación, pero es posible identificar dos tipos básicos de modulación de acuerdo con la clase de portadora: la "Modulación de Señales Continuas", en la cual la portadora es una señal sinusoidal, y la "Modulación de Impulsos", aquí la portadora es un tren de impulsos.

La modulación de señales continuas es un proceso continuo y por lo tanto es la apropiada para señales que varían en forma continua en el tiempo.

La Modulación de Impulsos es un proceso discreto, en el sentido de que los impulsos están presentes en ciertos intervalos de tiempo, lo que hace que la Modulación de Impulsos sea la forma apropiada para la transmisión de mensajes o información de naturaleza discreta. [6]

Finalizando, los procesos que intervienen en la comunicación son de sumo interés para que la misma cumpla sus verdaderos objetivos: la información tiene que llegar rápidamente, no sufrir cambios y no ser interceptada por agentes externos a la comunicación.

1.2 Herramientas para la simulación de la transmisión de datos

En el Departamento de Sistemas Digitales de la Uci, en reiteradas ocasiones los resultados académicos en la asignatura de Teleinformática I, específicamente en el tema de la teoría de la información no han sido muy buenos. Los estudiantes no poseen una guía efectiva para entender con mayor facilidad estos conocimientos, por lo que se ha pensado en el empleo de herramientas que muestren cómo ocurren los procesos que intervienen en la comunicación.

Las aplicaciones han sido un medio muy eficaz para el aprendizaje hoy en día, permitiendo la realización de distintas operaciones de manera práctica, eficiente y rápida. Por las razones antes mencionadas se hará un pequeño análisis de algunas aplicaciones que existen para la simulación de estos procesos entre las que se pueden citar:

1. LVSIM-DCOM

2. QS Versión 1.0
3. Matlab

1.2.1 LVSIM-DCOM

Software de simulación basado en Windows, el cual reproduce un laboratorio de clase tridimensional en una pantalla de computadora. Utilizando el ratón los estudiantes pueden instalar un equipo virtual en telecomunicaciones digitales y obtener los mismos resultados con el equipo de Lab-Volt.

Los sofisticados modelos matemáticos incluidos simulan exactamente las características de los componentes reales en telecomunicaciones digitales. Esta herramienta permite que los estudiantes puedan realizar las tareas de una forma más fácil y efectiva.

Esta herramienta en si familiariza a los estudiantes con las actividades prácticas relacionadas con:

1. Modulación de amplitud de impulso
2. Modulación de anchura de impulso
3. Modulación de posición de impulso
4. Conversión analógico digital
5. Conversión digital analógico

Características

1. Instalar, desplazar, rotar y remover módulos de telecomunicaciones digitales.
2. Conectar los componentes de telecomunicaciones digitales.
3. Modificar o remover las conexiones de los componentes de telecomunicaciones digitales.
4. Cambiar el color de los cable
5. Ampliar o reducir para ajustar la vista.
6. Aplicar potencia virtual al equipo.
7. Observar las formas de onda en un osciloscopio virtual.
8. Observar información espectral en un analizador de espectro virtual.
9. Realizar mediciones de frecuencia usando un frecuencímetro virtual.
10. Anotar las mediciones en una tabla de datos.
11. Dibujar las gráficas usando los datos anotados.
12. Imprimir pantallas de visualización.
13. Guardar y reabrir la configuración del equipo, los datos y las formas de onda.
14. Codificador MIC (PCM)
15. Decodificador MIC (PCM)
16. Codificador MICD (DPCM)

17. Decodificador MICD (DPCM)
18. Módem MDF (FSK)
19. Modulador MDFB (BPSK)
20. Demodulador MDFB (BPSK)
21. Codificador Delta/DPVC (Delta/CVSD)
22. C Decodificador Delta/DPVC (Delta/CVSD)[7]

1.2.2 QS Versión 1.0

Software simulador de las condiciones de tráfico en las redes de telecomunicaciones, la implementación del mismo sigue un modelo del tráfico que considera la distribución de arribo, de servicio, la disciplina de cola, el número de canales de entrada y de servicio, y combina los métodos analíticos y de simulación. Este analiza el tráfico, fallos y comportamiento durante la transmisión de la información.

Este software posee una sencillez tal, que logra un bajo costo computacional y puede ser utilizado para la simulación de nodos de una red considerando:

1. La distribución del tráfico de arribo.
2. La distribución del tráfico de servicio.
3. La disciplina de la cola.
4. El número de canales de entrada.
5. El número de canales de servicio.
6. La contabilidad de los estados intermedios.
7. Una combinación de los métodos analíticos y de simulación.

La herramienta se ha probado en la simulación de redes con servicios de http y ftp sobre Ethernet, con redes de IP sobre ATM y de voz IP y sobre modelos con características diferentes: M/M/1, M/D/1 y en sistemas con prioridad sin desalojo para varias clases de tráfico. En algunos casos las soluciones analíticas son conocidas y sirven como base para evaluar la calidad de los resultados del simulador. Todas las simulaciones se garantizan en una computadora personal con arquitectura superior a la de prueba con procesador Pentium III a 1000 MHz y al menos 128 Mb de memoria RAM y Sistema Operativo XP.[8]

1.2.3 Matlab

Matlab es una herramienta que permite hacer resolver funciones matemáticas así como simular redes neuronales, hacer operaciones estadísticas y además permite la simulación del proceso de transmisión

de datos entre otras actividades. Para la simulación de la emisión y recepción de la información este software brinda las siguientes opciones:

Simulink.

Funciones .m utilizadas desde la línea de comandos.

Funciones .m utilizadas mediante Interfaces Gráficas de Usuario (GUI).

Las últimas versiones de Matlab incorporan la posibilidad de utilizar interfaces gráficas de usuario para trabajar con sus archivos .m. Esto permite tener un entorno amigable para pedir datos y mostrar resultados al usuario, además puesto que es posible implementar funciones .m propias, el sistema que se desea simular estará totalmente personalizado desde el punto de vista de funcionalidad y de interfaz de usuario. En cuanto al proceso de modulación permite simular a las:

1. Modulaciones ASK
2. Modulaciones PSK
3. Modulaciones QAM
4. Modulaciones FSK

Este programa brinda la posibilidad de variar los parámetros internos de la simulación como la frecuencia. Los resultados pueden ser observados mediante gráficas o en una ventada de valores.[9]

De las herramientas antes tratadas se ha podido comprobar que son útiles en algunos de los casos ,como el Matlab que muestra gráficamente el proceso de modulación, el QS Versión 1.0 que simula la condiciones del tráfico en una red y el LVSIM-DCOM que contienen algunas opciones para la codificación y la modulación, pero estas herramientas no se centran exactamente en los algoritmos que se tienen en cuenta en la universidad como los algoritmos clásicos de cifrado, las codificaciones de Huffman y de Shanno mostrando como proceder con los mismos, para poderlos entender, es por eso que se ha decidido hacer una aplicación que trate los procesos de transmisión, impartidos de manera práctica en la asignatura de teleinformática, para que los estudiantes puedan entender con mayor facilidad a los mismos y tener una manera de comprobar los resultados de los ejercicios orientados en las clases prácticas.

Para crear la aplicación se ha tenido en cuenta el estudio de diferentes metodologías, herramientas y lenguajes de programación para su desarrollo, esto permitirá que al elegir la apropiada en cada uno de los casos, se realice el diseño e implementación de la herramienta. Antes de comenzar se dará a conocer algunos conceptos necesarios como:

1. ¿Qué es la programación orientada a objetos?
2. ¿Qué es una metodología de desarrollo de software?

3. ¿Qué es un IDE de programación?

1.3 Conceptos para el desarrollo de software

1.3.1 ¿Qué es la programación orientada a objetos?

La Programación Orientada a Objetos es la forma de expresar un programa en términos de objetos que colaboran entre si para realizar determinadas tareas, siendo estos representaciones de elementos de la vida real y que a su vez contienen toda la información que permiten su definición e identificación frente a otros objetos, además de contener funciones llamadas métodos que sirven de comunicación entre ellos.

Tiene características fundamentales que la identifican y son la base de las potencialidades que brinda a la hora de usarla para implementar cualquier sistema informático; ellas son: Herencia, Polimorfismo, y Encapsulamiento.

Es totalmente distinta a la programación estructurada, el programador trata de acercarse más al mundo real donde todo se vería como un objeto. Todo se refleja en términos de objetos, propiedades, métodos y otros elementos que se verán rápidamente para aclarar conceptos y dar una pequeña base sobre este paradigma. A continuación se citarán algunos conceptos de programación orientada a objetos según algunos autores:

La programación orientada a objetos es mucho más que una frase comercial (aunque haya llegado a ser así por culpa de algunas personas), una nueva sintaxis o una nueva interfaz de programación de aplicación. La programación orientada a objetos es un conjunto completo de conceptos e ideas. Es una manera de pensar en el problema al que va dirigido un programa de ordenador y de afrontarlo de modo más intuitivo e incluso más productivo. [10]

Los principios de la programación al objeto es “modelar” y representar, a través de elementos de programación, objetos que existen en la realidad (tangible o intangibles).[11]

1.3.2 ¿Qué es una metodología de desarrollo de software?

Una metodología de desarrollo de software es un proceso, el cual define qué, cuándo, y como alcanzar un determinado objetivo. Este debería ser capaz de servir como guía para todos los participantes (clientes, usuarios, desarrolladores y directores ejecutivos), evolucionar durante muchos años permitiendo limitar su alcance en un momento del tiempo dado a las realidades de la tecnología, herramientas, personas y patrones de organización.[12]

Se puede decir que, una metodología de desarrollo de software es un proceso que guía a los desarrolladores a los que les brinda métodos y herramientas, proporcionándoles una ayuda muy importante e indispensable para que el producto final posea las funcionalidades requeridas por el cliente y que cumpla con las necesidades del mismo y del usuario final, es una secuencia de actividades organizadas y bien pensadas que transforman los requisitos del cliente en el producto final.

1.3.3 ¿Qué es un IDE de programación?

IDE: Herramienta de soporte al proceso de desarrollo de software que integra las funciones básicas de edición de código, compilación y ejecución de programas. [13]

Los IDEs (Integrated Development Environment), tal y como su nombre indica, son entornos de desarrollo integrados.[14]

Resumiendo un IDE de programación es simplemente un software, pero que a diferencia de otros es una herramienta para ayudar a los programadores a escribir programas informáticos, interpretando uno o varios lenguajes de programación, compilándolos y mostrándole al programador si hubo o no un error en el código. Estos programas contienen un compilador que interpreta uno o varios lenguajes de programación.

1.4 Metodologías de desarrollo de software

En el mundo existen varias metodologías que son usadas para realizar el análisis de un software, entre ellas se encuentran las metodologías tradicionales y las metodologías ágiles. Las metodologías tradicionales se centran en el control de los procesos estableciendo actividades involucradas como: los artefactos que se deban producir, las herramientas que se deben de usar, sin embargo las metodologías ágiles se centran en otras dimensiones como el factor humano o el producto de software dándose mayor valor al individuo, a la colaboración con el cliente y al desarrollo incremental del software con iteraciones cortas. [15]

Por estas razones la metodología ágil está cobrando gran importancia en el mundo actual de desarrollo del software producto de la necesidad de satisfacer al cliente y obtener como resultado lo que en verdad este desea, teniendo resultados positivos en una gran cantidad de proyectos con requisitos que están en constantes cambios, que es lo más común que sucede cuando se desarrolla un software en el que se tiene un cliente que complacer y además de que en el mundo actual se necesita de un desarrollo rápido. A continuación se citarán algunas de las metodologías existentes.

1. SCRUM

2. Crystal Methodologies
3. Dynamic Systems Development Method (DSDM)
4. Adaptive Software Development (ASD)
5. Feature Driven Development (FDD)
6. Lean Development (LD)
7. RUP
8. XP
9. MSF

De una forma u otra, todas han contribuido en el desarrollo de la industria del software, sirviéndoles de guía a los desarrolladores para un mejor desempeño y cumplimiento de los requisitos de un determinado producto. Pero como todo, ellas tienen su forma particular de ser útil, pues no plantean lo mismo y no se pueden utilizar en cualquier proyecto, debido a que no existe una metodología estándar para cualquier tipo de desarrollo en la industria del software, pues todo depende del cliente, o de la empresa, de las características de la misma, entre otros factores como la cantidad de personal, el tiempo en que se debe entregar el producto y otros agentes que influyen en el desarrollo de un software.

Se centrará el estudio en RUP y XP donde la primera es tradicional y la segunda ágil.

1.4.1 Rational Unified Process (RUP)

La metodología RUP, llamada así por sus siglas en inglés Rational Unified Process, divide en 4 fases el desarrollo del software:

1. *Inicio:* Determinar la visión del proyecto.
2. *Elaboración:* Determinar la arquitectura óptima.
3. *Construcción*: Llegar a obtener la capacidad operacional inicial.
4. *Transmisión*: Llegar a obtener el release del proyecto.

Cada una de estas etapas es desarrollada mediante el ciclo de iteraciones, la cual consiste en reproducir el ciclo de vida en cascada a menor escala. Los objetivos de una iteración se establecen en función de la evaluación de las iteraciones precedentes.

Vale mencionar que el ciclo de vida que se desarrolla por cada iteración, es llevada bajo dos disciplinas:

Disciplina de Desarrollo

1. *Ingeniería de Negocios*: Entendiendo las necesidades del negocio.

2. *Requerimientos:* Trasladando las necesidades del negocio a un sistema automatizado.
3. *Análisis y Diseño:* Trasladando los requerimientos dentro de la arquitectura de software.
4. *Implementación:* Creando software que se ajuste a la arquitectura y que tenga el comportamiento deseado.
5. *Pruebas:* Asegurándose que el comportamiento requerido es el correcto y que todo los solicitado está presente.

Disciplina de Soporte

1. *Configuración y administración del cambio:* Guardando todas las versiones del proyecto.
2. *Administrando el proyecto:* Administrando horarios y recursos.
3. *Ambiente*: Administrando el ambiente de desarrollo.
4. *Distribución*: Hacer todo lo necesario para la salida del proyecto.

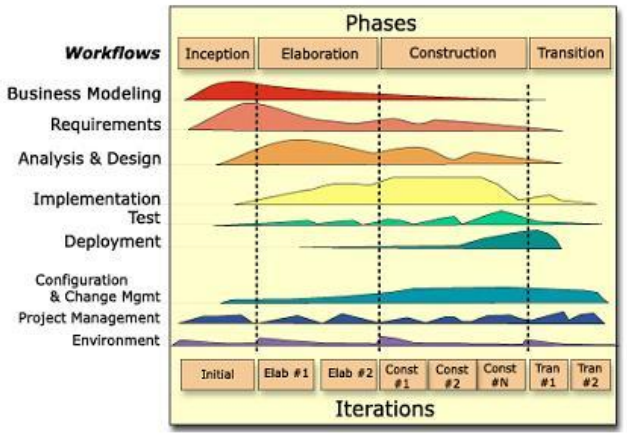


Figura 2: Metodología RUP.

Los elementos del RUP son:

1. *Actividades*: Procesos que se llegan a determinar en cada iteración.
2. *Trabajadores:* Las personas o gentes involucrados en cada proceso.
3. *Artefactos*: Un artefacto puede ser un documento, un modelo, o un elemento de modelo.

Una particularidad de esta metodología es que, en cada ciclo de iteración, se hace exigente el uso de artefactos, siendo por este motivo, una de las metodologías más importantes para alcanzar un grado de certificación en el desarrollo del software.[16]

1.4.2 Extreme Programing (XP)

Es una de las metodologías de desarrollo de software más exitosas en la actualidad, utilizadas para proyectos de corto plazo, entrega y equipo de desarrolladores pequeños. La misma consiste en una

programación rápida o extrema, cuya particularidad es tener como parte del equipo, al usuario final, pues es uno de los requisitos para llegar al éxito del proyecto.

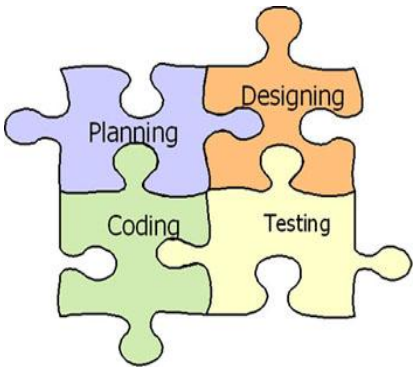


Figura 3: Metodología XP.

Características de XP, la metodología se basa en:

1. *Pruebas Unitarias:* se basa en las pruebas realizadas a los principales procesos, de tal manera que se adelanta en algo hacia el futuro, se pueda hacer pruebas de las fallas que pudieran ocurrir. Es como si se adelantara a obtener los posibles errores.
2. *Refabricación:* se basa en la reutilización de código, para lo cual se crean patrones o modelos estándares, siendo más flexible al cambio.
3. *Programación en pares:* una particularidad de esta metodología es que propone la programación en pares, la cual consiste en que dos desarrolladores participen en un proyecto en una misma estación de trabajo. Cada miembro lleva a cabo la acción que el otro no está haciendo en ese momento. Es como el chofer y el copiloto: mientras uno conduce, el otro consulta el mapa.

¿Qué es lo que propone XP?

Empieza en pequeño y añade funcionalidad con retroalimentación continua. El manejo del cambio se convierte en parte sustantiva del proceso. El costo del cambio no depende de la fase o etapa. No introduce funcionalidades antes que sean necesarias. El cliente o el usuario se convierten en miembro del equipo.

Lo fundamental en este tipo de metodología es:

1. La comunicación, entre los usuarios y los desarrolladores.
2. La simplicidad, al desarrollar y codificar los módulos del sistema.
3. La retroalimentación, concreta y frecuente del equipo de desarrollo, el cliente y los usuarios finales.[16]

Como se ha podido apreciar las metodologías de desarrollo de software antes tratadas tienen su forma de guiar a los desarrolladores de software, teniendo sus métodos particulares. Además el uso de estas depende del producto que se quiera desarrollar, del usuario final, de las características de la empresa, entre otras circunstancias que se presenten. Para el desarrollo del trabajo se tendrá en cuenta utilizar como metodología a RUP a pesar de que las metodologías ágiles como XP son aplicables a proyectos de corto plazo y de equipos de desarrollo pequeños, posee el inconveniente de que estos equipos deben estar compuesto por personal de experiencia en proyectos de software basándose en heurísticas provenientes de prácticas de antiguos proyectos, sin embargo RUP es basada en estándares seguidos por el entorno de desarrollo. Otro factor es en cuanto al cliente, en la que, en XP es parte del equipo de desarrollo o sea se considera un rol, mientras que en RUP el cliente interactúa mediante reuniones. Otro factor importante es en cuanto a los artefactos que se generan siendo RUP más abarcador en el tema, generando más artefactos que XP y centrándose más en la arquitectura, fundamental para cualquier desarrollo de un software. Ya determinada la metodología RUP para el desarrollo de la aplicación es necesario el apoyo en una herramienta para la modelación de los diferentes artefactos que la misma propone, teniendo en cuenta aquellos que pertenecen al lenguaje UML, como son los diferentes diagramas, es por eso que se dedicará el siguiente epígrafe al estudio de herramientas que permitan realizar los diferentes diagramas establecidos por UML.

1.5 Herramientas de Modelado

El modelado es el proceso que permite representar gráficamente el sistema de software, permitiendo resaltar los detalles más importantes. En el mundo de la ingeniería existen herramientas que ayudan a los desarrolladores a modelar los distintos diagramas especificados por UML, ejemplo de esto son el Rational Rose, el Umbrello, el Visual Paradigm entre otras. Para la realización de los diagramas pertinentes en el desarrollo de la aplicación se determinó emplear el Visual Paradigm, debido a sus características, entre las que se pueden mencionar:

1. Soporte para el ciclo de vida completo de desarrollo de software.
2. Soporte UML versión 2.1.
3. Permite realizar ingeniería inversa a lenguajes como Java, C++, C#, Esquemas XML.
4. Generación de código a través de los diagramas de diseño.
5. Permite la creación de diagramas de flujos de datos.
6. Disponibilidad en múltiples plataformas.

7. Diseño centrado en casos de usos y enfocado al negocio que genera un software de mayor calidad.
8. Modelado colaborativo con CVS y Subversion.
9. Generación de objetos Java desde la base de datos.
10. Generación de bases de datos, transformando diagramas de Entidad/Relación en tablas de base de datos.
11. Ingeniería inversa de bases de datos, desde Sistemas Gestores de Bases de Datos (DBMS) existentes a diagramas de Entidad/Relación.
12. Generador de informes para generación de documentación.
13. Importación y exportación de ficheros XML.
14. Integración con Visio.
15. Editor de figuras.
16. SDE para Eclipse.
17. SDE para NetBeans.
18. SDE para Sun ONE.
19. SDE para Oracle JDeveloper.
20. SDE para JBuilder.
21. SDE para WebLogic Workshop. [17]

A continuación se dedicará un espacio a explicar aspectos fundamentales en el diseño, pues es uno de los flujos más importantes de RUP, siendo el anterior al de implementación y el principal esfuerzo del presente trabajo esta incluido en el diseño.

1.6 El Diseño

Para desarrollar cualquier aplicación o sistema es necesario el diseño, pues es una de las etapas fundamentales que tiene el desarrollo del software. Un buen diseño es crucial para que la aplicación funcione correctamente y se termine el sistema en tiempo y forma, además que es el paso fundamental para la implementación. El mismo es importante para la creación de sistemas robustos y de fácil mantenimiento utilizando el paradigma orientado a objetos. Para desarrollar una aplicación, es necesario contar con descripciones detalladas y de alto nivel de la solución lógica y saber como satisface los requerimientos y restricciones, es este flujo de trabajo el encargado de realizar dichas operaciones.

1.6.1 ¿Qué es el diseño?

Según Pressman, el diseño es una representación significativa de ingeniería de algo que se va a construir. Se puede hacer el seguimiento basándose en los requisitos del cliente, y al mismo tiempo la calidad se puede evaluar y cotejar con el conjunto de criterios predefinidos para obtener un diseño "bueno". En el contexto de la ingeniería del software, el diseño se centra en cuatro áreas importantes de interés: datos, arquitectura, interfaces y componentes.

El diseño del software es un proceso iterativo mediante el cual los requisitos se traducen en un plano para construir el software y es tanto un proceso como un modelo, además es una secuencia de pasos que hacen posible que el diseñador describa todos los aspectos del software que se van a construir. [18]

Diseño orientado a objetos

Es durante el diseño orientado a objetos, donde se definen los objetos lógicos del software que finalmente serán implementados en un lenguaje de programación orientado a objetos. [19]

1.6.2 Importancia del Diseño

La importancia del diseño del software se puede describir con una sola palabra "calidad". Es el lugar en donde se fomentará la calidad en la ingeniería del software; proporciona las representaciones del software que se pueden evaluar en cuanto a calidad; es la única forma de convertir exactamente los requisitos de un cliente en un producto o sistema de software finalizado y sirve como fundamento para todos los pasos siguientes del soporte del software y de la ingeniería del software.[18]

En este se construyen distintos artefactos como el modelo de diseño, se determinan las clases necesarias para la construcción del software así como los distintos diagramas, se diseña el modelo de datos, se construye el diagrama de colaboración y secuencia en la que se reflejan como colaboran los objetos entre otros artefactos y actividades que son de sumo interés para los desarrolladores, en el mismo se aplican patrones que es una manera de evitar futuros problemas que ya se han presentado en disímiles situaciones, por esta razón se le dedicará un estudio ha determinados patrones que se consideran importantes para el desarrollo.

1.6.3 Patrones de diseño

A lo largo de los años los desarrolladores se han encontrado con problemas que se repiten reiteradamente en el desarrollo de un software, es por estas razones que los mismos han creado los patrones para usarlos en determinadas situaciones que se presenten. Los patrones han servido de ayuda y hoy en día se aplican mucho en distintos proyectos. En el diseño existen patrones los cuales

se nombran patrones de diseño que son: “Soluciones ya probadas y eficaces de los problemas de diseño que pueden expresarse como un conjunto de principios”. [19]
Una de las principales razones de las ciencias de la computación en cuanto a los patrones de diseño, es la necesidad de obtener soluciones elegantes, simples y reutilizables. Algunas definiciones de patrones de diseños se citan a continuación:

1. “Los patrones de diseño son soluciones a problemas de diseño que se ven una y otra vez”. (The Smalltalk Companion)[20]
2. “Los patrones de diseño constituyen un conjunto de reglas que describen como cumplir ciertas tareas en el desarrollo del software”. (Pree 1994) [20]
3. “Los patrones identifican y especifican abstracciones que están por encima de clases sencillas e instancias, o componentes” (Gamma et al., 1993)[20]

Existen una gran cantidad de patrones y cada uno hace un pequeño aporte para solucionar determinados problemas, pero cuidado no se pueden utilizar siempre los patrones donde no hacen falta, o sea no son efectivos en cualquier proyecto ni situación que se presente, simplemente se deben aplicar cuando realmente hacen falta, porque sino puede traer serios problemas más adelante. Para su mejor estudio se tiene en cuenta entre los de diseño a los patrones GRASP y los GOF.

1.6.3.1 Patrones generales de software para asignar responsabilidades (GRASP)

Los patrones Grasp describen principios fundamentales de diseño de objetos para la asignación de responsabilidades las que están relacionadas con las obligaciones de un objeto en cuanto a su comportamiento. Existen varios patrones Grasp, a continuación se citan algunos:

Experto:
¿Quién asumirá la responsabilidad en el caso general?
Asignar una responsabilidad al experto en información: la clase que posee la información necesaria para cumplir con la responsabilidad.

Creador:
¿Quién crea?
Asignar a la clase B la responsabilidad de crear una instancia de la clase A, si se cumple una de las siguientes condiciones:

1. B contiene A
2. B agrega A
3. B tiene los datos de inicialización de A

4. B registra A
5. B utiliza A muy de cerca

Controlador:
¿Quién administra un evento del sistema?

Asignar la responsabilidad de administrar un mensaje de eventos del sistema a una clase que represente una de las siguientes opciones:

1. El negocio o la organización global (un controlador de fachada).
2. El "sistema" global (un controlador de fachada).
3. Un ser animado del dominio que realice el trabajo (un controlador de papeles).
4. Una clase artificial (Fabricación Pura) que represente el caso de uso (un controlador de casos de uso).[19]

1.6.3.2 Patrones GOF

Los patrones GOF se separan en patrones de creación, estructurales y de comportamiento. Los de creación son los que se encargan de crear los objetos, mientras que los estructurales se dedican al planteamiento de las relaciones entre las clases combinándolas para la formación de estructuras, finalmente los de comportamiento se dedican a la interacción y cooperación entre las clases, estudiando las relaciones entre los mensajes que ocurren entre los objetos. [20]

Entre los patrones de creación se pueden citar a:

1. *Fábrica Abstracta:* Permite trabajar con objetos de distintas familias de manera que las familias no se mezclen entre sí y haciendo transparente el tipo de familia concreta que se esté usando.
2. *Constructor:* Abstrae el proceso de creación de un objeto complejo, centralizando dicho proceso en un único punto.
3. *Método de fabricación:* Centraliza en una clase constructora la creación de objetos de un subtipo de un tipo determinado, ocultando al usuario la casuística para elegir el subtipo que se crea.
4. *Prototipo:* Crea nuevos objetos clonándolos de una instancia ya existente.
5. *Singleton:* Garantiza que una clase sólo tenga una instancia y proporciona un punto de acceso global a esta instancia.

Entre los patrones estructurales se pueden citar a:

1. *Adaptador:* Adapta una interfaz para que pueda ser utilizada por una clase que de otro modo no podría utilizarla
2. *Puente:* Desacopla una abstracción de su implementación

3. *Objeto compuesto:* Permite tratar objetos compuestos como si de uno simple se tratase.
4. *Envoltorio:* Añade funcionalidad a una clase dinámicamente.
5. *Fachada:* Provee de una interfaz unificada simple para acceder a una interfaz o grupo de interfaces de un subsistema.
6. Peso ligero: Reduce la redundancia cuando gran cantidad de objetos poseen idéntica información.
7. Proxy: Mantiene un representante de un objeto.

Entre los patrones de comportamiento se pueden citar:

1. *Iterador:* Permite realizar recorridos sobre objetos compuestos independientemente de la implementación de estos.
2. *Mediador:* Define un objeto que coordine la comunicación entre objetos de distintas clases, pero que funcionan como un conjunto.
3. *Observador:* Define una dependencia de uno a muchos entre objetos, de forma que cuando un objeto cambie de estado se notifique y actualicen automáticamente todos los objetos que dependen de él.
4. *Estado:* Permite que un objeto modifique su comportamiento cada vez que cambie su estado interno.

1.7 Lenguajes de Programación

Los lenguajes de programación y los IDE de desarrollo tienen una suma importancia para cualquier proyecto de software. ¿Se imaginan qué sería de los desarrolladores sin la existencia de un lenguaje de programación y de un IDE que lo ayude en la tarea de corregir código? Bueno, sería fatal, pues con estos se han creado innumerables programas y herramientas que han ayudado al hombre a controlar de una manera más sencilla a los ordenadores, así como de realizar múltiples tareas y actividades.

Hoy en día existen muchos lenguajes a los que se puede acudir para la realización de cualquier software, pero como unos son más fáciles de utilizar que otros debido a la eliminación de código inseguro, el cumplimiento con el paradigma de la programación orientada a objetos, sentencia más amigable y sencilla, la facilidad que proporcionan para su aprendizaje, la propiedad de que sean multiplataforma, entre otras disímiles características que permiten a los programadores elegirlos, por esto seguidamente se analizarán algunos de ellos.

1.7.1 Java

Java es un lenguaje de programación creado por SUN Microsystems muy parecido al estilo de programación del lenguaje “C” y basado en lo que se llama programación orientada a objeto.[11]

Entre las principales características de Java se pueden citar:

1. Sintaxis similar a la de C++. Aunque se simplifican algunas características del lenguaje como:
2. La sobrecarga de operadores, la herencia múltiple, el paso por referencia de parámetros, la gestión de punteros, la liberación de memoria y las instrucciones de pre compilación.

3. Soporte homogéneo a la Programación Orientada a Objetos. A diferencia de C++, que puede considerarse un lenguaje multiparadigma, Java está diseñado específicamente para utilizar el paradigma de orientación a objetos.
4. Independencia de la plataforma. En Java se pretende que con una sola compilación se obtenga código ejecutable en diferentes Sistemas Operativos e incluso sobre diferente hardware.[21]

La compañía Sun describe el lenguaje Java como “simple, orientado a objetos, distribuido, interpretado, robusto, seguro, de arquitectura neutra, portable, de altas prestaciones, multitarea y dinámico”. [14]

Orientado a objetos: Desde un principio fue diseñado orientado a objetos que agrupan en estructuras encapsuladas tanto sus datos como los métodos que manipulan esos datos.

Distribuido: Proporciona una colección de clases para su uso en aplicaciones de red, que permiten abrir sockets y establecer y aceptar conexiones con servidores o clientes remotos, facilitando así la creación de aplicaciones distribuidas.

Interpretado y compilado a la vez: Java es compilado, en la medida en que su código fuente se transforma en una especie de código máquina, los bytecodes, semejantes a las instrucciones de ensamblador. Por otra parte, es interpretado, debido a que los bytecodes se pueden ejecutar directamente sobre cualquier máquina a la cual se hayan portado el intérprete y el sistema de ejecución en tiempo real.

Robusto: Fue diseñado para crear software altamente fiable. Para ello proporciona numerosas comprobaciones en compilación y en tiempo de ejecución. Sus características de memoria liberan a los programadores de una familia entera de errores como la aritmética de punteros, porque ya en este lenguaje se ha prescindido por completo de los punteros, y la recolección de basura elimina la necesidad de liberación explícita de memoria. Java está diseñado para soportar aplicaciones que serán ejecutadas en los más variados entornos de red, desde Unix a Windows NT, pasando por Mac y estaciones de trabajo, sobre arquitecturas distintas y con sistemas operativos diversos. La indiferencia

a la arquitectura representa sólo una parte de su portabilidad. Además, especifica los tamaños de sus tipos de datos básicos y el comportamiento de sus operadores aritméticos, de manera que los programas son iguales en todas las plataformas.

Estas dos últimas características se conocen como la Máquina Virtual Java (JVM).

Multihilo: En la actualidad muchas aplicaciones realizan varias operaciones al mismo tiempo, este lenguaje posee la característica de que los programadores puedan explotar esto, pues java permite la programación multihilo o multiproceso en el cual se crean múltiples procesos que se encargan de realizar cálculos y operaciones distintas en el mismo instante.

1.7.2 C Sharp

Este lenguaje posee una estructuración y una sintaxis muy parecida a la de C++ o Java, esto con el interés de Microsoft de atraer a los programadores de Java y C++, tomando las mejores características de estos lenguajes. También es un lenguaje orientado a objetos simple, elegante y con seguridad en el tratamiento de tipos, que permite a los programadores de aplicaciones empresariales crear una gran variedad de aplicaciones.

Proporciona la capacidad de generar componentes de sistema duraderos debido a que posee total compatibilidad entre COM y plataforma para integración de código existente, también presenta gran robustez, gracias a la recolección de elementos no utilizados a esto se le conoce como liberación de memoria. Además seguridad implementada por medio de mecanismos de confianza intrínsecos del código, presenta plena compatibilidad con conceptos de meta datos extensibles. En este lenguaje es posible interaccionar con otros lenguajes, entre plataformas distintas, y con datos heredados, en virtud de varias características como la plena interoperabilidad por medio de los servicios de COM+ 1.0 y .NET Framework con un acceso limitado basado en bibliotecas, compatibilidad con XML para interacción con componentes basados en tecnología Web y capacidad de control de versiones para facilitar la administración y la implementación.

Orientación a objetos: Como todo lenguaje de programación de propósito general actual, C# es un lenguaje orientado a objetos. Una diferencia de este enfoque orientado a objetos respecto al de otros lenguajes como C++ es que el de C# es más puro en tanto que no admiten ni funciones ni variables globales sino que todo el código y datos han de definirse dentro de definiciones de tipos de datos, lo que reduce problemas por conflictos de nombres y facilita la legibilidad del código.

Soporta todas las características propias del paradigma de programación orientada a objetos: encapsulación, herencia y polimorfismo.

Gestión automática de memoria: Proporciona un recolector basura el cual se encarga de destruir los objetos innecesarios o que están sin uso.

Seguridad de tipos: Incluye mecanismos que permiten asegurar que los accesos a tipos de datos siempre se realicen correctamente, lo que permite evita que se produzcan errores difíciles de detectar por acceso a memoria no perteneciente a ningún objeto y es especialmente necesario en un entorno gestionado por un recolector de basura. Para ello se toman medidas del tipo:

Instrucciones seguras: Para evitar errores muy comunes, en C# se han impuesto una serie de restricciones en el uso de las instrucciones de control más comunes. Por ejemplo, la guarda de toda condición ha de ser una expresión condicional y no aritmética, con lo que se evitan errores por confusión del operador de igualdad (==) con el de asignación (=); y todo caso de un switch ha de terminar en un break o goto que indique cuál es la siguiente acción a realizar, lo que evita la ejecución accidental de casos y facilita su reordenación.[10]

Para la elección de uno de los lenguajes que se tomaron como referencia, se tuvieron en cuenta factores que influyen a la hora de elegir a uno de los dos. Entre estos factores se pueden mencionar el significado de software libre, esto se debe a que para Java existen potentes IDE de desarrollo que son libres y C# posee algunos que todavía están empezando a desarrollarse como el MonoDevelop. Otro factor es que las aplicaciones desarrolladas en Java pueden ejecutarse en diferentes sistemas operativos siempre que estos tengan la máquina virtual instalada, sin embargo las que están desarrolladas en C# no poseen esta facilidad. Por estas razones se ha elegido al Java como el lenguaje idóneo para la implementación del software. Cualquiera sea el lenguaje que se utilice, es necesario contar con medios para que los desarrolladores puedan editar el código, compilar, y ejecutar programas con mayor facilidad, es ahí donde radica la importancia de un IDE de desarrollo. Java es un lenguaje que posee una larga oferta de IDEs, entre los que se encuentran: Netbeans, Eclipse, JBuilder y otros.

1.8 IDEs para Java

La elección de un IDE no es simple y sobre todo para un lenguaje como Java donde existen muchas variedades como se había explicado anteriormente. Muchos son los parámetros a la hora de elegirlos, debido a que algunos solo funcionan en unas plataformas y otras no, su facilidad para desarrollar, la interfaz gráfica, que sea libre o no, entre otras.

Los IDE que se tuvieron en cuenta para su estudio fueron el Netbeans y el Eclipse, estos poseen una interfaz amigable, son fáciles de manipular, y sobre todo son multiplataforma y libres.

1.8.1 NetBeans

Netbeans es un IDE libre con ambiente integrado para desarrolladores de código abierto. Contiene todas las herramientas necesarias para el desarrollo de aplicaciones profesionales de escritorio, web, y de móviles en los lenguajes de programación java C, C++ y en Ruby. Este IDE puede ser ejecutado en distintas plataformas incluyendo Windows, Linux, Mac OS X y Solaris. Es muy fácil de instalar y permite una agradable interacción con el desarrollador.

Características principales

Herramienta para construcción de interfaces de visuales (GUI)

Posee una barra de herramientas que permite con facilidad la creación de interfaces, donde el desarrollador puede arrastrar diferentes componentes, alinearlos y programarlos.

Modelado UML

Contiene una barra de herramientas para el modelado UML de la aplicación que se esté desarrollando.

Desarrollo en C y C++

Está concebido para el desarrollo en Java, pero permite la potencialidad de que el desarrollador pueda programar en otros lenguajes como C y C++, pues contiene un editor, y herramientas de corrección del código para estos lenguajes.[22]

Desarrollo de aplicaciones para celulares

Permite el desarrollo de aplicaciones para celulares, contiene entre sus características opciones para la creación de un proyecto en el cual el desarrollador puede realizar la construcción de un software destinado a cualquier móvil.

A pesar de las características esenciales descritas anteriormente, Netbeans permite la integración de múltiples herramientas y protocolos, brindando motivos para la migración. También la capacidad de crear eficientemente aplicaciones Java para una amplia variedad de plataformas es también resultado del soporte de lenguajes de especificaciones no propietarias para el modelado de objetos. Ello permite la edición en ambos sentidos, el código fuente cambia conjuntamente con los cambios en el modelo y libra a los desarrolladores de la necesidad de consultar constantemente los comentarios en el código. El soporte al modelado mejora la productividad del desarrollador, además de que conlleva a que los desarrolladores se concentren en la lógica de negocio durante todo el ciclo de construcción de la aplicación manteniendo un reflejo de como hoy el software debe cumplir con los requisitos del mercado.

1.8.2 Eclipse

Eclipse es una plataforma universal para integrar herramientas de desarrollo, con una arquitectura abierta y basada en plug-ins. Además, Eclipse da soporte a todo tipo de proyectos que abarcan desde el ciclo de vida del desarrollo de aplicaciones, incluyendo soporte para modelado.[23]

Características principales

1. Editor visual con sintaxis coloreada
2. Compilación incremental de código
3. Modifica e inspecciona valores de variables
4. Avisa de los errores cometidos mediante una ventana secundaria
5. Depura código que resida en una máquina remota

Sistemas operativos que lo soportan

1. Linux
2. Windows
3. Solaris 8 (SPARC/GTK 2)
4. Mac OSX –Mac/Carbon

Ambos son muy potentes y poseen innumerables características que han sido aprovechadas, pero entre ellos se decidió realizar la aplicación con el Netbeans 6.0, debido al rendimiento y el uso de recursos, pues el eclipse consume demasiado recursos y no posee una paleta de componentes para construir aplicaciones visuales como el primero, sino que debe de acudir a la instalación de plugins.

Conclusiones

En este capítulo se han tratado diversos temas de sumo interés, se analizaron herramientas que simulan la transmisión de la información, pero no resuelven a grandes rasgos el problema que se presenta en la asignatura de Teleinformática I, por esas razones se ha llegado a la siguiente conclusión:

1. Es necesario dominar los conceptos y algoritmos que se imparten en la asignatura de Teleinformática I referente a los Procesos de Transmisión de Datos.
2. Los softwares estudiados no cubren la necesidad de los estudiantes y la asignatura, siendo necesario la construcción de uno que satisfaga las mismas.
3. Para el desarrollo de este se seleccionó como Metodología de Desarrollo a RUP, como lenguaje de programación a Java, como IDE al NetBeans 6.0 y como herramienta de modelado al Visual Paradigm.

4. Dentro del proceso de desarrollo tiene gran importancia el diseño, siendo una buena práctica utilizar aquellas soluciones que ya han sido probadas en casos anteriores.

CAPÍTULO 2: CARACTERÍSTICAS DEL SISTEMA

Introducción

En el presente capítulo se realizarán todos los pasos necesarios para dar inicio al desarrollo de la aplicación, siguiendo como guía a RUP, la cual quedó seleccionada como la más apropiada en el capítulo anterior. Además se mostrarán y explicarán los principales artefactos que se obtienen en los flujos de trabajo de Negocio y Requisitos, facilitando así una mejor compresión inicial de la funcionalidad del software. Entre los artefactos a generar se tienen al modelo del dominio, los requisitos funcionales y no funcionales, el diagrama de casos de uso del sistema y las descripciones de cada caso de uso.

2.1 Caso de estudio

El sistema que se propone realizar en apoyo a la asignatura de Teleinformática I debe de contar con módulos de codificación, cifrado y compresión, permitiéndoles a los estudiantes la entrada de un mensaje al cual la aplicación le ejecutará los procesos mencionados, mostrando los resultados de entrada y salida de cada acción. El proceso puede ser de manera general o específica, o sea, el estudiante tiene la posibilidad de probar cada procedimiento de manera particular, eligiendo un módulo en específico, compresión, codificación o cifrado y por el algoritmo deseado. Otra de las funcionalidades requeridas es que el software permita al estudiante realizar el proceso de transmisión a modo general obviando la modulación y la multiplexación, dándole la facilidad de ver paso a paso los resultados obtenidos durante la emisión de un mensaje.

2.2 Procesos de la teoría de la información que se automatizarán

Los procesos que intervienen en la transmisión son: La codificación de la fuente y el canal de comunicación, el cifrado, la compresión, la modulación y la multiplexación, de los que se obviarán a la hora de automatizar los algoritmos de modulación y multiplexación, pues no son objetivos principales para la asignatura de Teleinformática I en esta parte de las comunicaciones. A continuación se mencionan los procesos con sus respectivos algoritmos a incluir en el sistema como principales funcionalidades.

Tabla 1: Algoritmos a implementar.

Proceso de transmisión	Algoritmos
Codificación	Huffman, Shanno Fanno, Hammig.
Cifrado	Vernam, César con clave y sin clave.
Compresión	RLE, MNP, LZ77, Aritmético.

2.3 Compresión del contexto del sistema mediante un modelo de dominio

Un modelo de dominio es el encargado de capturar los tipos más importantes de objetos en el contexto del sistema, que representan las "cosas" que existen o los eventos que suceden en el entorno en el que trabaja el sistema. Las clases del dominio representan conceptos u objetos en el mundo real, cosas que se manipulan en el negocio o sucesos que ocurrirán o han ocurrido. [12]

Se ha decidido usar un modelo de domino y no acudir al modelo del negocio pues para el sistema que se trata de desarrollar los actores del negocio y trabajadores no son fáciles de determinar, o sea que cuando se dificulta a la hora de realizar un software el proceso de obtención de los mismos es recomendable aplicar un modelo del dominio, además de que todo fluye en el problema conceptualmente, es de ahí la elección de no emplear un modelo del negocio y si uno del dominio.

Para la construcción del modelo del dominio se tuvo en cuenta a cada proceso que interviene en la transmisión de la información. Mediante la aplicación de los conceptos que intervienen en la transmisión de la información se obtuvo el siguiente modelo de dominio Figura 4.

.

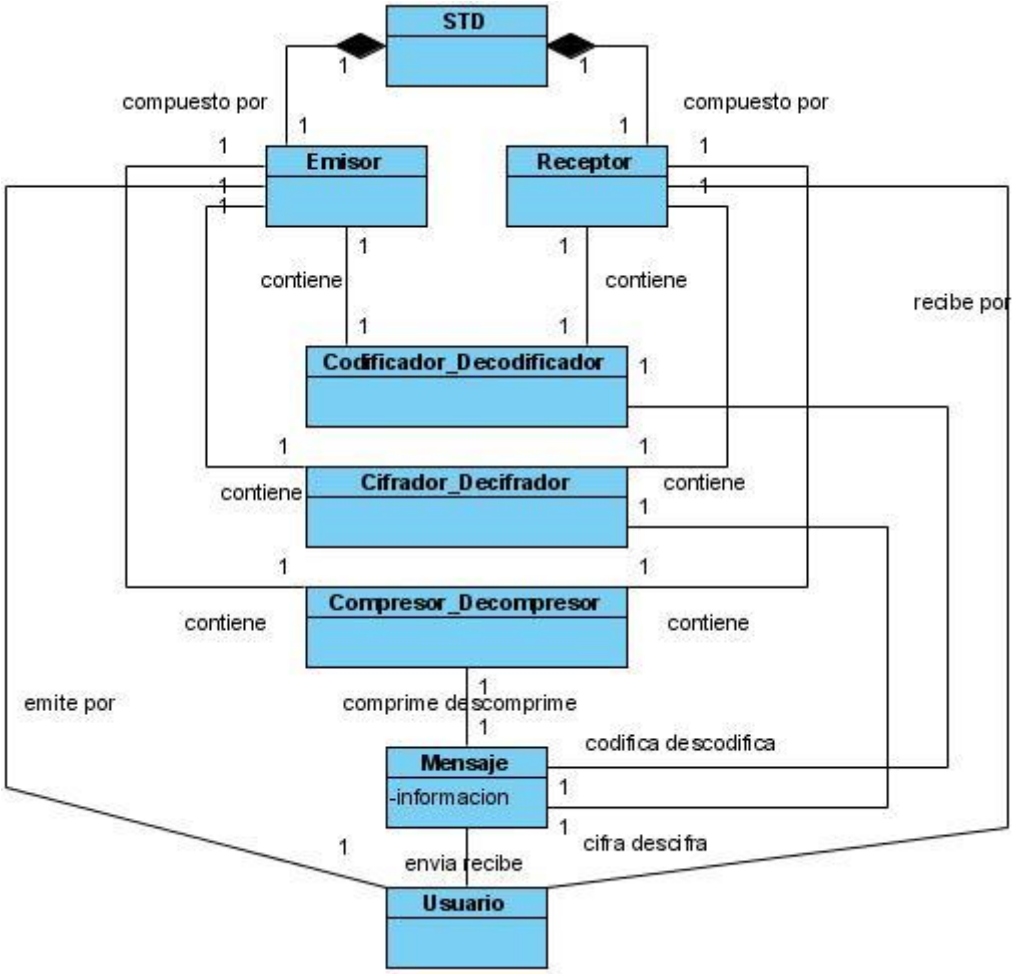


Figura 4: Modelo de Dominio

2.3.1 Descripción de las clases del dominio

1. *STD (Sistema de Transmisión de Datos):* Contiene a la clase Emisor y Receptor, un sistema de transmisión está compuesto por los mismos, además es la encargada de controlar todo el proceso de comunicación.
2. *Emisor:* Es la encargada de enviar cualquier mensaje, contiene a la clase Codificador_Descodificador, Cifrador_Descifrador, Compresor_Descompresor, que son las que simulan los procesos que ocurren a la hora de transmitir información.
3. *Receptor:* Contiene las mismas clases que Emisor, pero realiza el proceso inverso al Emisor, recibe el mensaje que envía este.

4. *Codificador_Descodificador:* Su misión es la de codificar y decodificar la información transmitida.
5. *Cifrador_Descifrador:* Encargada de cifrar y descifrar la información transmitida.
6. *Compresor_Descompresor:* Su objetivo es de comprimir y descomprimir la información transmitida.
7. *Mensaje:* Representa a la información que sufrirá los distintos procesos que intervienen en la comunicación.
8. *Usuario:* En este caso representa al usuario que envía y recibe el mensaje o sea el que inicia el proceso de transmisión.

2.4 Especificación de los requisitos del sistema

Para la realización de cualquier aplicación una de las tareas primordiales es la captura de requisitos. Los requerimientos no son más que una condición o capacidad que tiene que tener un sistema para satisfacer un contrato o documento formal, dividiéndose en dos grandes grupos, funcionales y no funcionales, donde los primeros no son más que capacidades o funcionalidades que el sistema debe cumplir, y los segundos son propiedades o cualidades que el producto debe de tener, representando las características del producto. Los no funcionales se clasifican por categorías, que pueden ser:

1. De Software.
2. De Hardware.
3. De Seguridad.
4. De Usabilidad.
5. De Rendimiento.
6. De Soporte.
7. De Portabilidad.
8. De Apariencia o Interfaz Externa.

El propósito fundamental del flujo de trabajo de los requisitos es guiar el desarrollo hacia el sistema. Un reto fundamental para conseguir una buena captura de requisitos es que el cliente, que en la mayoría de las veces o en el peor de los casos no es un especialista en informática, consiga entender el resultado de la obtención de los requisitos, para eso se debe utilizar el lenguaje del mismo para describir esos resultados.

2.4.1 Requisitos funcionales del sistema

Para la captura de los mismos se hizo una entrevista a algunos profesores del departamento de Sistemas Digitales que atienden la asignatura de Teleinformática I y se obtuvieron los siguientes requisitos funcionales:

R1. Procesar mensaje

1. R1.1 Seleccionar algoritmo de codificación. Permite al usuario seleccionar un algoritmo de codificación para enviar el mensaje.
2. R1.2 Seleccionar algoritmo de cifrado. Permite al usuario seleccionar un algoritmo de cifrado para enviar el mensaje.
3. R1.3 Seleccionar algoritmo de compresión. Permite al usuario seleccionar un algoritmo de compresión para enviar el mensaje.
4. R1.4 Mostrar información.
    1. R1.4.1 Mostrar resultado del proceso de codificación seleccionado. Se le muestra al usuario el resultado del proceso de codificación.
    2. R1.4.2 Mostrar resultado del proceso de cifrado seleccionado. Se le muestra al usuario el resultado del proceso de cifrado.
    3. R1.4.3 Mostrar resultado del proceso de compresión seleccionado. Se le muestra al usuario el resultado del proceso de compresión.

R2. Mostrar pasos realizados por cada proceso.

1. R2.1 Mostrar pasos realizados en el proceso codificación. Permite al usuario mostrar paso a paso el resultado de la codificación.
2. R2.2 Mostrar pasos realizados en el proceso cifrado. Permite al usuario mostrar paso a paso el resultado del cifrado.
3. R2.3 Mostrar pasos realizados en el proceso compresión. Permite al usuario mostrar paso a paso el resultado de la compresión.

R3. Codificar.

R4. Descodificar.

R5. Cifrar.

R6. Descifrar.

R7. Comprimir.

R8. Descomprimir.

2.4.2 Requisitos no funcionales

Requerimiento de Apariencia o Interfaz Externa: Se necesita que la aplicación este provista de una interfaz amigable, profesional y clara.

Requerimientos de Usabilidad: El sistema está concebido para ser usado por los profesores del departamento de sistemas digitales que imparten la asignatura de Teleinformática I y por los alumnos que la cursan.

Requerimientos de Rendimiento: La aplicación debe ser eficiente, rápida y precisa, por lo cual el tiempo de respuesta debe oscilar entre los 3 a 12 milisegundos.

Requerimientos de Software: Para el uso de la aplicación se deberá contar con la máquina virtual de java (JVM) versión 6.0, además la aplicación está concebida para todos los sistemas operativos que soporten la instalación de la (JVM).

Requerimientos de Hardware: Para que la aplicación pueda ejecutarse a una velocidad aceptable y no provoque afectaciones en el rendimiento del ordenador, debe garantizarse que la computadora contenga al menos 256 MB de memoria RAM y un microprocesador Pentium 4 o superior.

2.5 Modelo de Casos de Uso del Sistema

El modelo de casos de uso permite que los desarrolladores de software y los clientes lleguen a un acuerdo sobre los requisitos, es decir sobre las condiciones y posibilidades que debe cumplir el sistema y es la entrada fundamental para el análisis, el diseño y las pruebas. Para ello su forma representativa en UML es a través de los casos de usos y sus relaciones con los actores.

Una técnica excelente que permite mejorar la comprensión de los requerimientos es la creación de casos de usos, estos son descripciones narrativas de los procesos del dominio. Los mismos requieren tener al menos un conocimiento parcial de los requerimientos del sistema expresados en el documento donde se especifican, por lo que según Craig Larman en el libro UML y Patrones brinda la siguiente definición: “Un caso de uso son fragmentos de funcionalidad que el sistema ofrece para aportar un resultado de valor para sus actores. De manera más precisa, un caso de uso especifica una secuencia de acciones que el sistema puede llevar a cabo interactuando con sus actores, incluyendo alternativas dentro de la secuencia”. Además para su mejor compresión el autor antes mencionado da a conocer el concepto de actor que inserta en el de caso de uso: “Un actor es una entidad externa del sistema que de alguna manera participa en la historia del caso de uso. Por lo regular estimula el sistema con eventos de entrada o recibe algo de él”.

2.5.1 Definición de los Actores del Sistema

Tabla 2: Actores del Sistema

Actor	Descripción
Usuario	Es el encargado de introducir un mensaje y hacer la petición de envío o de ejecución de un proceso de transmisión (codificación, cifrado, compresión). El mismo puede ser un estudiante o un profesor.

2.5.2 Listado de los Casos de Uso del Sistema

Tabla 3: CU Procesar Mensaje

Caso de Uso	**Procesar Mensaje**
Actor	**Usuario**
Descripción	Realiza todos los procesos de transmisión a un mensaje previamente introducido por el usuario, mostrando los resultados de cada uno de ellos.
Referencia	R1, R1.1 , R1.2 , R1.3 , R1.4 , R1.4.1 , R1.4.2 , R1.4.3

Tabla 4: CU Mostrar Pasos

Caso de Uso	**Mostrar Pasos**
Actor	**Usuario**
Descripción	Encargado de mostrar paso a paso los procedimientos realizados a un mensaje previamente introducido por el usuario.
Referencia	R2 , R2.1 , R2.2 , R2.3

Tabla 5: CU Codificar

Caso de Uso	**Codificar**
Actor	**Usuario**
Descripción	Encargado de realizar la codificación del mensaje entrado por el usuario, mostrando los resultados del proceso.
Referencia	R3

Tabla 6: CU Descodificar

Caso de Uso	**Descodificar**
Actor	Usuario
Descripción	Encargado de realizar la descodificación a un mensaje previamente codificado, mostrando los resultados del proceso.
Referencia	R4

Tabla 7: CU Cifrar

Caso de Uso	**Cifrar**
Actor	Usuario
Descripción	Encargado de realizar el cifrado del mensaje entrado por el usuario, mostrando los resultados del proceso.
Referencia	R5

Tabla 8: CU Descifrar

Caso de Uso	**Descifrar**
Actor	Usuario
Descripción	Encargado de realizar el descifrado a un mensaje previamente cifrado, mostrando los resultados del proceso.

Referencia	R6

Tabla 9: CU Comprimir

Caso de Uso	**Comprimir**
Actor	Usuario
Descripción	Encargado de realizar la compresión del mensaje entrado por el usuario, mostrando los resultados del proceso.
Referencia	R7

Tabla 10: CU Descomprimir

Caso de Uso	**Descomprimir**
Actor	Usuario
Descripción	Encargado de realizar la descompresión a un mensaje previamente comprimido, mostrando los resultados del proceso.
Referencia	R8

2.5.3 Diagrama de Casos de Uso del Sistema

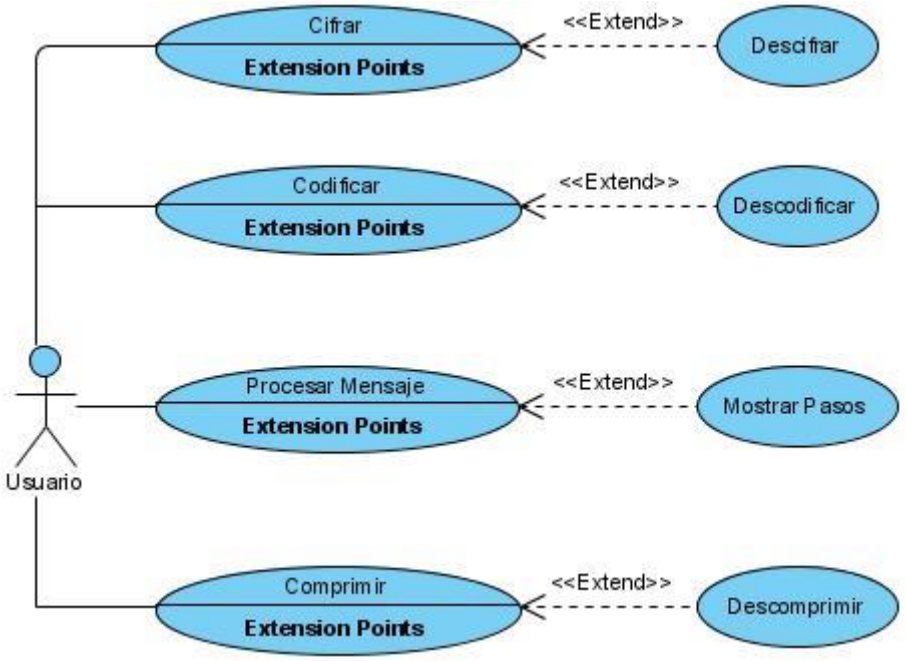


Figura 5: Diagrama de Casos de Uso

2.5.4 Descripción de los casos de uso del sistema

Tabla 11: Descripción CU Procesar Mensajes

Caso de uso	**Procesar Mensaje**
Actores	**Usuario**
Resumen	El caso de uso se inicia cuando el usuario introduce un mensaje y pulsa la opción enviar mensaje, entonces el sistema realiza todos los procesos de transmisión, para luego mostrar los resultados de cada uno de ellos.
Referencia	R1, R1.1 , R1.2 , R1.3 , R1.4 , R1.4.1 , R1.4.2 , R1.4.3
Casos de usos asociados	Mostrar Pasos
Precondiciones	
Flujo normal de eventos	

Acción del Actor	Respuesta del sistema
1. El usuario introduce el mensaje. 2. El usuario selecciona los algoritmos por proceso de transmisión a ejecutar. 3. El usuario pulsa la opción Procesar mensaje.	4. El sistema realiza los procedimientos que intervienen en la transmisión del mensaje. 5. El sistema muestra los resultados de cada proceso que interviene en la transmisión del mensaje.
Flujos alternos	
Acción del Actor	**Respuesta del sistema**
1. El actor no introduce el mensaje a procesar.	2. El sistema muestra una ventana de error advirtiendo de que debe introducir un mensaje.
Pos condiciones	
Prioridad	alta
Puntos de extensión	4. El usuario pulsa opción mostrar pasos. Ver CU Mostrar Pasos

Tabla 12: Descripción CU Codificar

Caso de uso	**Codificar**
Actores	**Usuario**
Resumen	El caso de uso se inicia cuando el usuario selecciona la opción de codificación, entonces el sistema se encarga de realizar la codificación del mensaje previamente entrado por el usuario, mostrando los resultados obtenidos por el mismo.
Referencia	R3
Casos de usos asociados	Descodificar
Precondiciones	
Flujo normal de eventos	
Acción del Actor	**Respuesta del sistema**
1. El usuario selecciona la opción Codificación. 3. El usuario selecciona algoritmo de codificación.	2. El sistema muestra ventana, para la codificación.

4. El usuario introduce un mensaje y pulsa codificar.	5. El sistema realiza la codificación del mensaje, mostrando los resultados y los pasos realizados para codificar.
Flujos alternos	
Acción del Actor	**Respuesta del sistema**
1. El actor no introduce el mensaje a codificar	2. El sistema muestra una ventana de error advirtiendo de que debe introducir un mensaje.
Pos condiciones	
Prioridad	alta
Puntos de extensión	6. El usuario pulsa la opción descodificar. Ver CU Descodificar.

Tabla 13: Descripción CU Descodificar

Caso de uso	**Descodificar**
Actores	**Usuario**
Resumen	El caso de uso se inicia cuando el usuario pulsa la opción descodificar, entonces el sistema realiza el proceso inverso a la codificación devolviendo el mensaje descodificado.
Referencia	R4
Casos de usos asociados	
Precondiciones	El usuario debe de haber realizado el proceso de codificación.
Flujo normal de eventos	
Acción del Actor	**Respuesta del sistema**
1. El usuario pulsa descodificar	2. El sistema realiza la operación de descodificación y muestra los resultados.
Flujos alternos	
Acción del Actor	**Respuesta del sistema**

Pos condiciones	
Prioridad	alta
Puntos de extensión	

Tabla 14: Descripción CU Cifrar

Caso de uso	**Cifrar**
Actores	**Usuario**
Resumen	El caso de uso se inicia cuando el usuario selecciona la opción de cifrado introduce un mensaje y pulsa cifrar, entonces el sistema realiza el proceso de cifrado al mensaje introducido, mostrando los resultados del mismo.
Referencia	R5
Casos de usos asociados	Descifrar
Precondiciones	
Flujo normal de eventos	
Acción del Actor	**Respuesta del sistema**
1. El usuario selecciona la opción de cifrado. 3. El usuario selecciona un algoritmo de cifrado. 4. El usuario introduce un mensaje y pulsa cifrar.	2. El sistema muestra ventana para el cifrado. 5. El sistema realiza el cifrado del mensaje, mostrando los resultados y los pasos realizados.
Flujos alternos	
Acción del Actor	**Respuesta del sistema**
1. El usuario no introduce el mensaje a cifrar.	2. El sistema muestra una ventana de error advirtiendo de que debe introducir un mensaje.
Pos condiciones	
Prioridad	alta
Puntos de extensión	6. El usuario pulsa la opción descifrar. Ver CU Descifrar.

Tabla 15: Descripción CU Descifrar

Caso de uso	**Descifrar**
Actores	**Usuario**
Resumen	El caso de uso se inicia cuando el usuario pulsa descifrar, entonces el sistema realiza el proceso inverso al cifrado devolviendo el mensaje descifrado.
Referencia	R6
Casos de usos asociados	
Precondiciones	El usuario debe de haber realizado el cifrado de un mensaje.
Flujo normal de eventos	
Acción del Actor	**Respuesta del sistema**
1. El usuario pulsa la opción descifrar.	2. El sistema realiza el proceso de descifrado devolviendo el mensaje descifrado.
Flujos alternos	
Acción del Actor	**Respuesta del sistema**
Pos condiciones	
Prioridad	alta
Puntos de extensión	

Tabla 16: Descripción CU Comprimir

Caso de uso	**Comprimir**
Actores	**Usuario**
Resumen	El caso de uso se inicia cuando el usuario selecciona la opción Compresión, entonces el sistema realiza la compresión a un mensaje previamente entrado por el usuario mostrando los resultados de la misma.
Referencia	R7

Casos de usos asociados	Descomprimir
Precondiciones	
Flujo normal de eventos	
Acción del Actor	**Respuesta del sistema**
1. El usuario selecciona la opción de Compresión. 3. El usuario selecciona algoritmo de compresión. 4. El usuario introduce un mensaje y pulsa comprimir.	2. El sistema muestra ventana, para la compresión. 5. El sistema realiza la compresión del mensaje, mostrando los resultados, y los pasos realizados.
Flujos alternos	
Acción del Actor	**Respuesta del sistema**
1. El usuario no introduce el mensaje a comprimir.	2. El sistema muestra una ventana de error advirtiendo la necesidad de introducir el mensaje para realizar el proceso.
Pos condiciones	
Prioridad	alta
Puntos de extensión	6. El usuario pulsa la opción descomprimir. Ver CU Descomprimir.

Tabla 17: Descripción CU Descomprimir

Caso de uso	**Descomprimir**
Actores	**Usuario**
Resumen	Realiza el proceso inverso a la compresión devolviendo el mensaje descomprimido.
Referencia	R8
Casos de usos asociados	
Precondiciones	El usuario debe de haber realizado el proceso de compresión.
Flujo normal de eventos	
Acción del Actor	**Respuesta del sistema**
1. El usuario pulsa la opción descomprimir.	2. El sistema realiza el proceso de descompresión devolviendo el mensaje original.

Flujos alternos	
Acción del Actor	**Respuesta del sistema**
Pos condiciones	
Prioridad	alta
Especificaciones complementarias:	

Tabla 18: Descripción CU Mostrar Pasos

Caso de uso	**Mostrar Pasos**
Actores	**Usuario**
Resumen	El caso de uso se inicia cuando el usuario pulsa la opción mostrar pasos, entonces el sistema muestra como se realizaría paso a paso cada proceso de transmisión en el proceso en general.
Referencia	R2 , R2.1 , R2.2 , R2.3
Casos de usos asociados	
Precondiciones	El usuario debe de haber realizado el envío de un mensaje.
Flujo normal de eventos	
Acción del Actor	**Respuesta del sistema**
1. El actor pulsa la opción mostrar pasos	2. El sistema muestra por proceso de transmisión los pasos realizados para obtener cada resultado.
Flujos alternos	
Acción del Actor	**Respuesta del sistema**
Pos condiciones	
Prioridad	media
Especificaciones complementarias:	

2.5.5 Prototipo de Interfaz de Usuario

Los prototipos de interfaz de usuario ayudan a comprender y especificar las iteraciones entre actores humanos y el sistema durante la captura de requisitos. No solo ayudan a desarrollar una interfaz gráfica mejor, sino también a comprender mejor los casos de uso. A la hora de especificar la interfaz de usuario también podrían utilizarse otros artefactos como los modelos de interfaz gráfica y los esquemas de pantalla. Ver prototipos de interfaz:

1. Caso de Uso Procesar Mensaje. Ver Anexo 1
2. Caso de Uso Mostrar Pasos. Ver Anexo 2
3. Caso de Uso Codificar. Ver Anexo 3
4. Caso de Uso Descodificar. Ver Anexo 4
5. Caso de Uso Cifrar. Ver Anexo 5
6. Caso de Uso Descifrar. Ver Anexo 6
7. Caso de Uso Comprimir. Ver Anexo 7
8. Caso de Uso Descomprimir. Ver Anexo 8

Conclusiones

El presente capítulo ha servido como iniciador para el desarrollo del sistema propuesto, generando los principales artefactos que propone RUP en la fase de Inicio y parte de la de elaboración, por lo que se puede llegar a la conclusión siguiente:

1. Se obtuvo el modelo del dominio del sistema el cual agrupa los conceptos más importantes que intervienen en el contexto del sistema.
2. Se determinaron los requisitos funcionales y no funcionales del sistema, permitiendo comprender cuáles son las funcionalidades esenciales que debe cumplir la aplicación para su correcto funcionamiento.
3. Se determinaron los casos de usos correspondientes y sus respectivas descripciones aclarando como es que ocurre el flujo de eventos para la realización de cada uno de los mismos.
4. Se brindan los prototipos de interfaz de usuario para cada caso de uso para una mejor comprensión de cada uno de ellos.

CAPÍTULO 3: DISEÑO DEL SISTEMA

Introducción

En el presente capítulo se accionará sobre el flujo de trabajo de diseño. El mismo es de vital importancia en RUP para el desarrollo de cualquier producto de software. Este flujo depende de los artefactos generados en el capítulo anterior, sirven de entrada para luego dar como salida nuevos artefactos, entre los que se encuentran las clases del diseño con sus funcionalidades y atributos, diagramas de clases del diseño, diagramas de interacción, subsistemas de diseño, entre otros. También se dedicará un espacio a la aplicación de patrones tanto de diseño como arquitectónicos para obtener una buena calidad en el producto final, componentes reutilizables entre otras ventajas que aportan.

3.1 El diseño y la arquitectura

El diseño de un software es el centro de atención al final de la fase de elaboración y el comienzo de las iteraciones de construcción según la metodología RUP. El mismo fue tratado de manera especial en el Capítulo 1, se abordaron conceptos importantes, su utilidad y la mención de algunos de los patrones de diseños existentes en el mundo del desarrollo del software. Pero no solo se basa en como se construye su respectivo modelo, la obtención de las clases necesarias y sus relaciones, de como colaboran los objetos, etcétera, sino que va más allá, adentrándose en la arquitectura, pues uno de sus objetivos según Pressman, es derivar una representación arquitectónica de un sistema, sirviendo como marco de trabajo desde donde se llevan a cabo actividades más detalladas, como el empleo de un conjunto de patrones arquitectónicos que permiten que el ingeniero de software reutilice los conceptos a nivel de diseño.

3.1.1 ¿Qué es la arquitectura de software?

La arquitectura de software alude a la estructura global del mismo, y a las formas en que la estructura proporciona la integridad conceptual de un sistema. En su forma más simple es la estructura jerárquica de los componentes del programa y la manera en que interactúan. [18]

3.1.2 Importancia de la arquitectura

En el libro dedicado a la arquitectura de software, Bass y sus colegas identifican tres razones fundamentales:

1. Las representaciones de la arquitectura facilitan la comunicación entre todas las partes interesadas en el desarrollo de un sistema basado en computadoras.
2. Destaca decisiones tempranas de diseño, teniendo profundo impacto en todo el trabajo en la ingeniería.
3. Constituye un modelo relativamente pequeño comprensible de como está estructurado el sistema y como trabajan juntos sus componentes.

3.1.3 Diseño arquitectónico

Roger S. Pressman en su libro Ingeniería de software un enfoque práctico parte 1 da a conocer el siguiente concepto de diseño arquitectónico: “El diseño arquitectónico representa la estructura de los datos y los componentes del programa que se requieren para construir un sistema basado en computadora, constituyendo el estilo arquitectónico que tendrá el sistema, la estructura y la propiedad de los componentes que ese sistema comprende.”

3.1.4 Estilos arquitectónicos y Patrones de arquitectura

Todo software construido por sistemas basados en computadoras cuenta con diversos estilos arquitectónicos, los cuales describen una categoría del sistema que contiene. Entre estos estilos se encuentran el de:

1. Arquitecturas centradas de datos.
2. Arquitectura de flujo de datos.
3. Arquitectura de llamada y retorno.
4. Arquitectura orientada a objetos.
5. Arquitectura estratificada.

De las arquitecturas que se han mencionado serán aplicadas la de orientadas a objetos y la de flujo de datos, debido a que la primera cumple con el paradigma orientado a objetos, y es donde los componentes de un sistema encapsulan los datos y funciones que se encargan de realizar operaciones sobre ellos, además de la comunicación entre los componentes a través de mensajes. La segunda producto a que es aplicada cuando los datos de entrada son procesados a través de una serie de componentes computacionales o manipulativos en los datos de salida en la que se ve implícito el patrón de Filtros y Tuberías aplicado en el desarrollo de la aplicación que se obtendrá.

3.1.5 Patrón arquitectónico de Filtros y Tuberías (Piping and filtering)

Esta arquitectura es muy simple pero a su vez es muy potente y robusta. Consiste en una serie de componentes (filtros), que están conectados a través de tuberías. Cada componente tiene un conjunto de entradas y un conjunto de salidas leyendo corrientes de datos de entradas, generando corrientes de datos de salida que pueden ser posibles entradas al próximo filtro y así sucesivamente hasta producir una salida final.

Filtro: transforma los datos que les llegan a través de las tuberías, el mismo puede tener cualquier número de de entradas y salidas.

Tubería: conecta dos filtros pasándoles los datos.

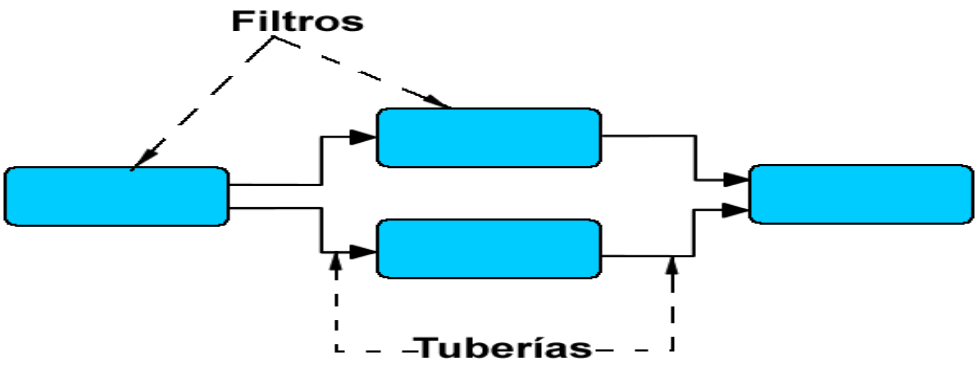


Figura 6: Patrón Tuberías y Filtros

3.1.5.1 ¿Por qué el uso del patrón Filtros y Tuberías?

Se decidió en el empleo del patrón filtros y tuberías como el más idóneo para el tipo de aplicación que se desarrolla, porque como se explicaba en el Capítulo 1, un sistema de transmisión está compuesto por varios procesos, los cuales van transformando un flujo de datos produciendo una salida para el próximo proceso de transmisión, se identificaron a estos procesos como los filtros en el sistema, son los encargados de procesar un flujo de entrada produciendo uno de salida, facilitando el trabajo a la hora de simular el procesamiento de un mensaje. Para una mejor comprensión ver Figura 7.

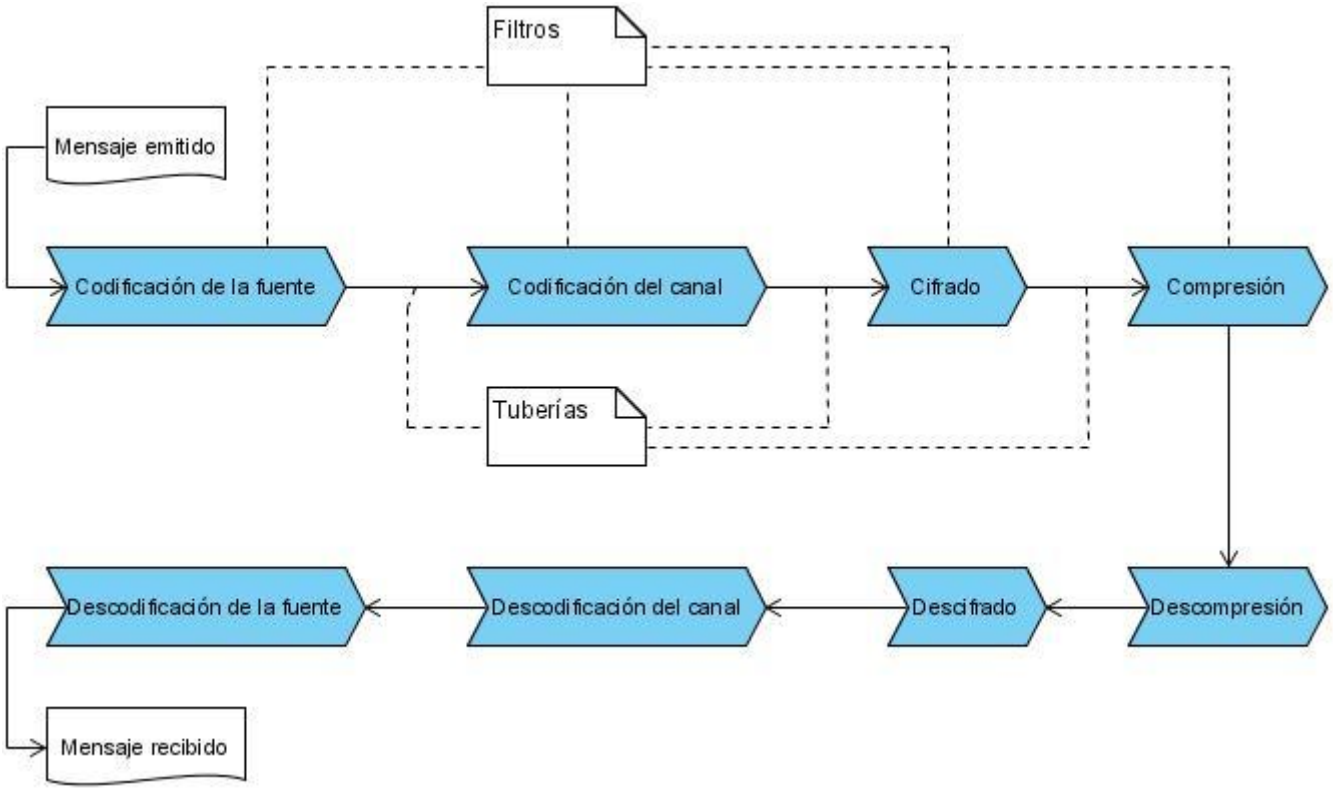


Figura 7: Patrón Tuberías y Filtros aplicado al problema

3.1.5.2 ¿Cómo aplicar el patrón arquitectónico en el diseño de clases?

A la hora de la realización del diseño se creó una clase *CTuberia,* esta será la encargada de pasar los datos de salida de un filtro a otro. Además participan las clases *CSTD*, *CEmisor* y *CReceptor*, la primera es la encargada mediante el uso de la clase *CTuberia* pasar el flujo de salida que genera *CEmisor* como flujo de entrada al *CReceptor*, estas últimas contienen una colección de 4 filtros, procesando información entre ellos mediante el empleo de la clase *CTuberia*. Finalizando la clase *CFiltro* es la unidad más pequeña en este diagrama, siendo la encargada de procesar el flujo de entrada, dando como respuesta un flujo de salida. Los filtros son: el codificador, el compresor y el cifrado. Como se puede apreciar en la Figura 8 se expone el diagrama de clases generado.

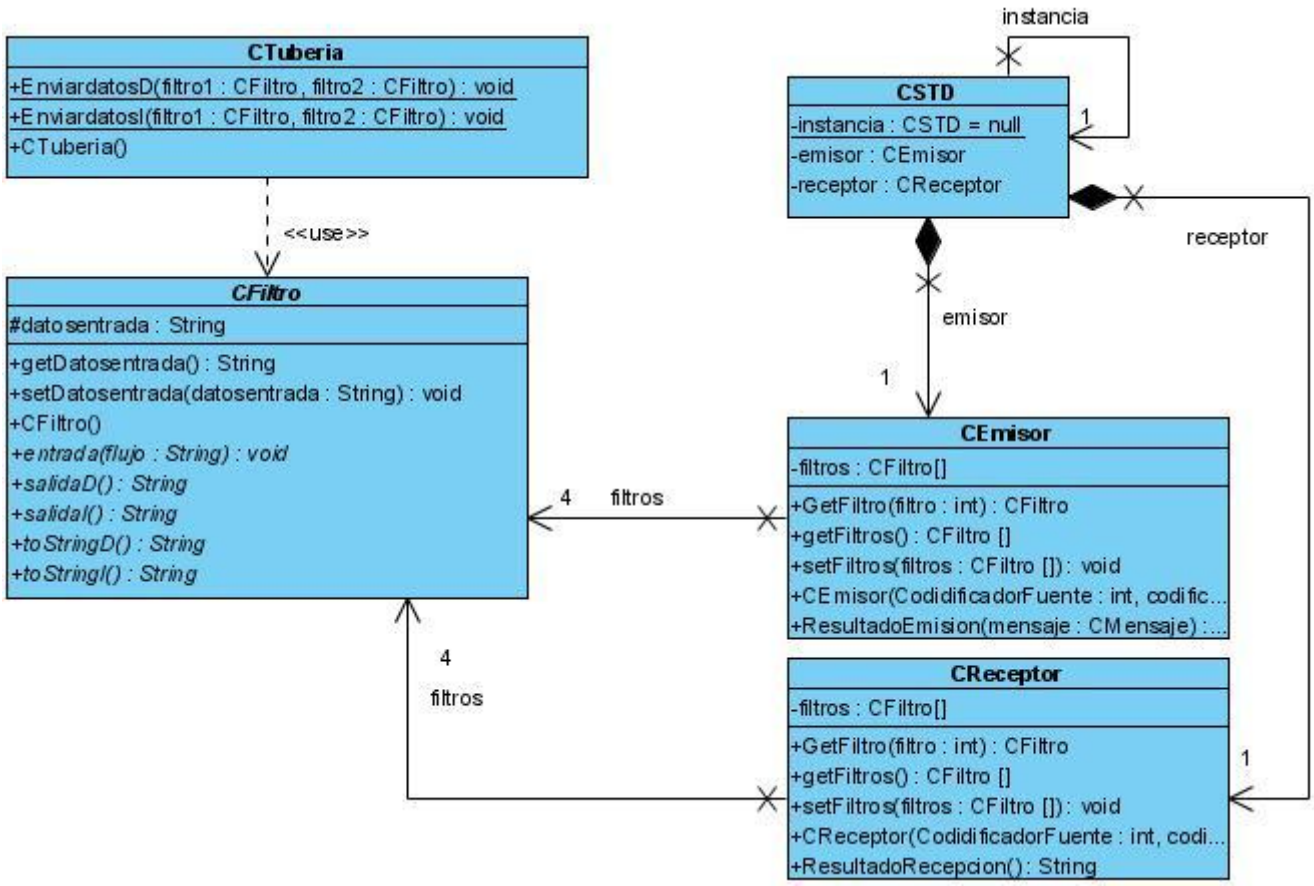


Figura 8: Aplicando Patrón arquitectónico Filtros y Tuberías en el diseño de clases

Como se explicaba anteriormente la clase *CTuberia* tiene un gran significado a la hora de la transmisión del flujo de información de un filtro a otro, por eso se le dedica un espacio mediante un diagrama de iteración para comprender mejor este mecanismo que será uno de los pasos en las realizaciones de algunos de los casos de uso que se tratarán más adelante. En la Figura 9 se observa como ocurre la comunicación entre una tubería y dos filtros cuando se realiza la llamada a *EnviarDatos (filtro1, filtro2)* de la misma.

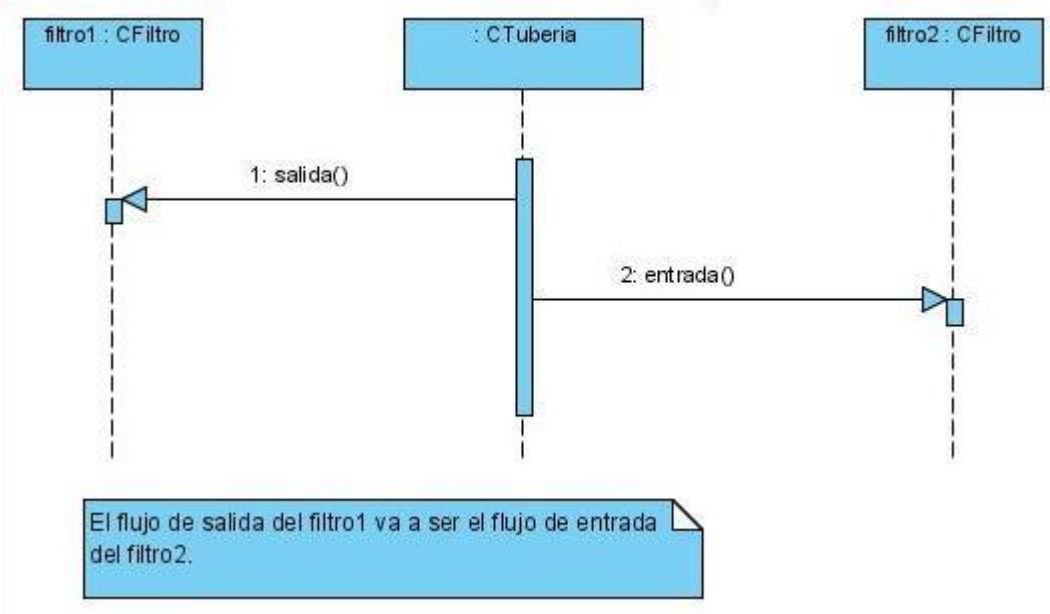


Figura 9: Diagrama de iteración Comunicación Tubería Filtros

3.4 Patrones de diseño empleados

Como se había explicado en el Capítulo 1 los patrones de diseño poseen una gran importancia en el desarrollo de software, eliminando problemas que se han repetido una y otra vez a la hora de construir aplicaciones, es por esto la necesidad de aplicar algunos de ellos. A veces cuando se implementa un software se incluyen patrones sin que el desarrollador se percate de esto, generalmente por falta de conocimiento, pero en el peor de los casos se encuentra con un problema al que no le encuentra una solución, o la que tiene no es la más eficiente. También muchos aplican indiscriminadamente patrones por decir que los han empleado y dar sensación de que lo dominan y los conocen a la perfección sin darse cuenta que no es el problema indicado para el empleo de los mismos o de alguno en particular. Para el desarrollo de la aplicación se pensó detenidamente en el mejor o el más cercano al mejor diseño de clases posibles sin abusar de los patrones existentes, por lo que se llegó a emplear como patrones principales el Singleton para crear una sola instancia en las clases *CSTD* que es controladora, y era necesario que fuera llamada fácilmente de un formulario a otro. El patrón Abstract Factory es el otro y el más interesante pues contiene incluido al Singleton y permite mediante unas clases llamadas fábricas construir familias de objetos. En el caso que se viene tratando fue necesario el empleo del Abstract Factory, es de mucha ayuda a la hora de crear distintos tipos de compresores, codificadores y cifradores. Las clases del diseño involucradas en este patrón son *CFactory_Cifrador*,

CFactory_Codificador y *CFactory_Compresor* como las fábricas, heredando de *CAbstrac_Factory_filtro*. También se encuentran la clase *CFiltro*, de la que heredan *CCifrador_desc*, *CCompresor_desc*, *CCodificador_desc,* cada una con su respectiva familia, o sea esto servirá a la hora de crear los filtros independientemente de que filtro sea: compresor, codificador o cifrador y de que tipo en específico es cada uno de ellos. Para una mejor comprensión ver la Figura 10.

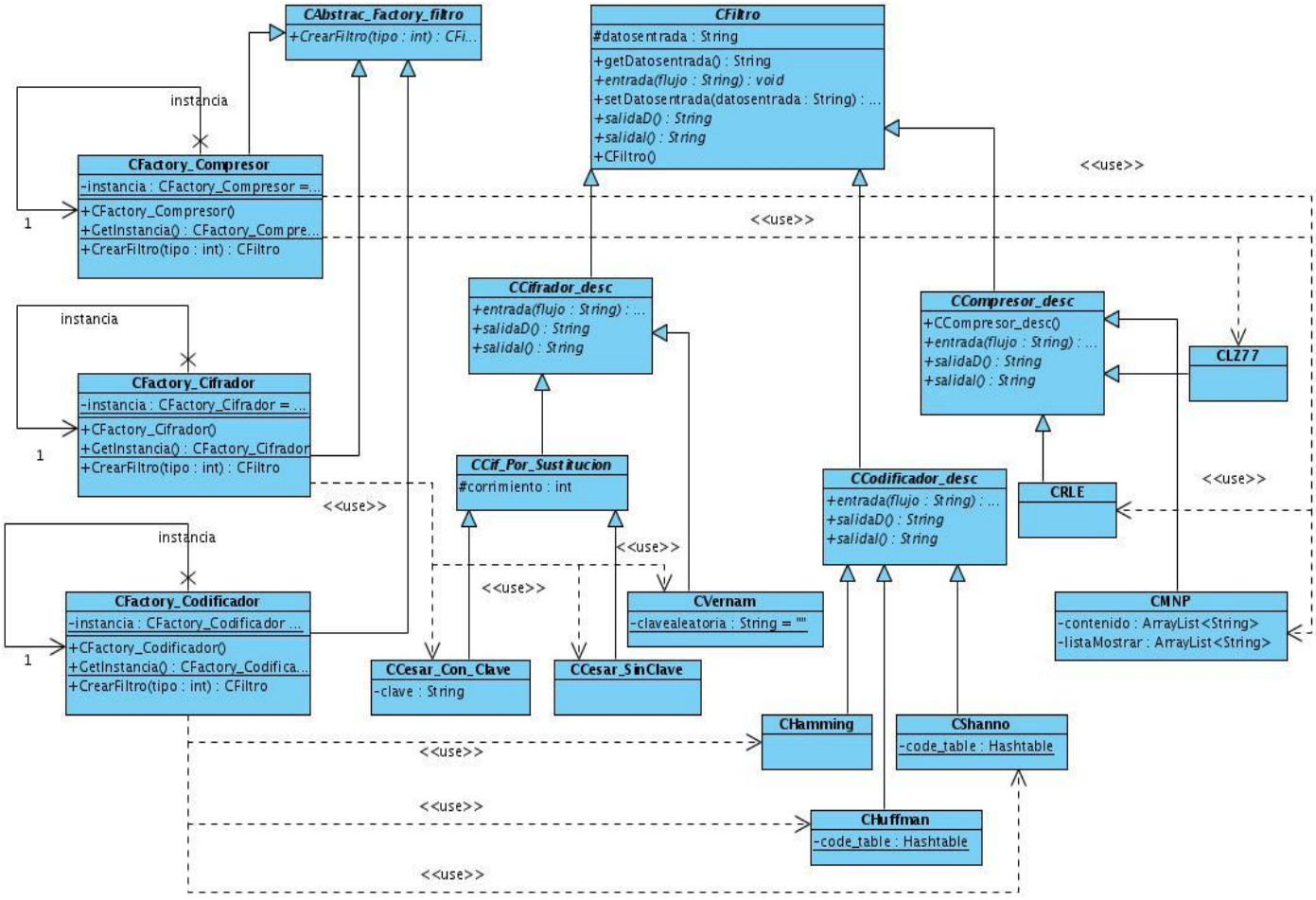


Figura 10: Diagrama de clases del Patrón Abstract Factory

3.5 Modelo del diseño

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

implementación. Según plantea la metodología seguida para el desarrollo de este sistema, está compuesto por varios artefactos, subsistemas, clases y diagramas de clases de diseño; así como las realizaciones de los casos de uso del diseño.

3.5.1 Paquetes de diseño

Son una colección de clases, relaciones, realizaciones de casos de usos, diagramas y otros paquetes que son empleados para dividir en partes más pequeñas al modelo de diseño. Estos son utilizados para agrupar elementos del modelo de diseño que se relacionen y como herramienta organizacional. A diferencia de los subsistemas de diseño no ofrecen una interfaz formal. [24]

Para una mejor organización del diseño de la aplicación se crearon los siguientes paquetes:

1. Procesamiento. Ver Anexo 9
2. Información. Ver Anexo 10
3. Filtros. Ver Anexo 11
4. Visual. Ver Anexo 12
5. Codificador_desc. Ver Anexo 13
6. Cifrador_desc. Ver Anexo 14
7. Compresor_desc. Ver Anexo 15

Más adelante se puede ver en la Figura 11 el diagrama de paquetes de diseño del sistema que se viene desarrollando, plasmando las dependencias existentes entre ellos.

Tabla 19: Paquete de diseño Procesamiento

Paquete	**Procesamiento**
Descripción	Es el encargado de realizar todas las operaciones de control a modo general, o sea contiene las clases más generales que manejan, controlan y procesan información.
Clases	CReceptor, CEmisor, CSTD.

Tabla 20: Paquete de diseño Información

Paquete	**Información**
Descripción	Este subsistema está compuesto por clases que contienen información y son capaces de realizar

	ciertos cálculos para mostrar información adicional.
Clases	CMensaje, CSymbol.

Tabla 21: Paquete de diseño Filtros

Paquete	**Filtros**
Descripción	Contiene clases que aplican el patrón Abstract Factory para la creación de los distintos filtros y la clase CTuberia, encargada de realizar las operaciones de envío de información de un filtro a otro. Además está compuesto por los subsistemas *Codificador_desc, Cifrador_desc y Compresor Desc.*
Clases	CAbstrac_Factory_filtro, Factory_Cifrador, Factory_Codificador y CFactory_Compresor, CTuberia, CFiltro.

Tabla 22: Paquete de diseño Visual

Paquete	**Visual**
Descripción	Como su nombre lo indica en este subsistema están todas las clases que intervienen en el diseño de interfaz de usuario, o sea los formularios.
Clases	Todos los formularios insertados en la aplicación.

Tabla 23: Paquete de diseño Codificador_desc

Paquete	**Codificador_desc**
Descripción	Contiene todas las clases encargadas de realizar la codificación de un mensaje.
Clases	CHuffman, CShanno, CHamming, CHuffmanGraphics, CSimbolPaint, CPantalla,

	CCodificador_desc, CArbolBinario.

Tabla 24: Paquete de diseño Cifrador_desc

Paquete	**Cifrador_desc**
Descripción	Contiene las clases encargadas de realizar el proceso de cifrado de un mensaje.
Clases	CAlfabeto, CCifrador_desc, CVernam,, CCesar_SinClave, CCesar_ConClave, CCif_Por_Sustitucion,

Tabla 25: Paquete de diseño Compresor_desc

Paquete	**Compresor_desc**
Descripción	Contiene las clases encargadas de realizar el proceso de compresión de un mensaje.
Clases	CMNP, CCompresor_desc, CLZ77, CRLE.

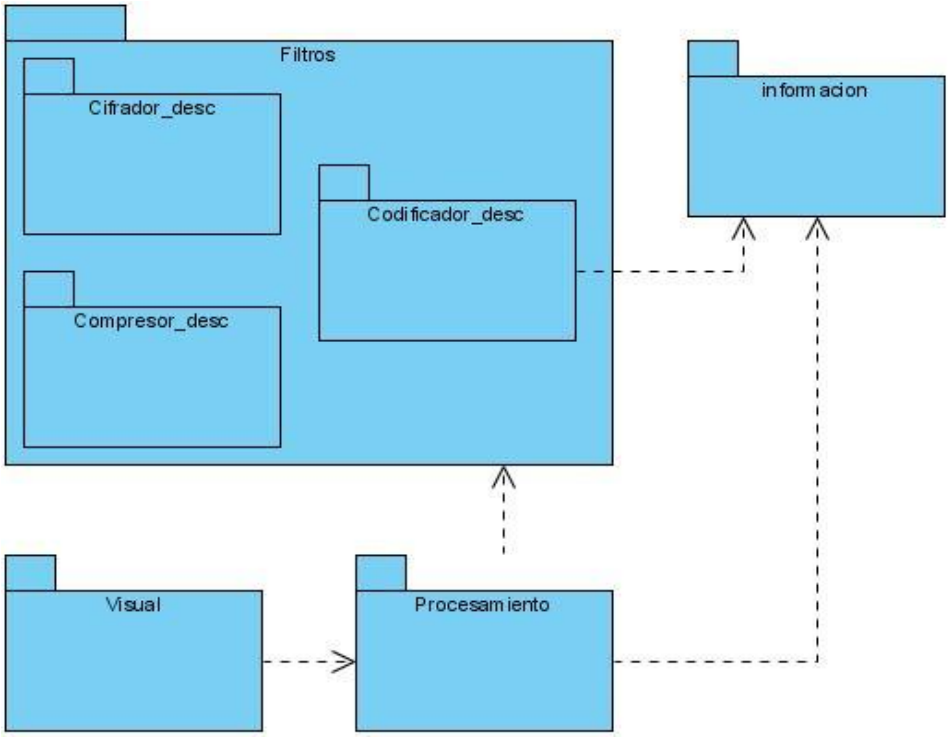


Figura 11: Paquetes del diseño

3.5.2 Descripciones de las clases del diseño más significativas

1. CSTD: Encargada de procesar la salida de receptor y el emisor mostrando los resultados de ambos, además de ser la clase más general en el diseño.
2. CEmisor: Encargada de procesar toda la información durante la emisión del mensaje, devolviendo los resultados por cada proceso que interviene a la hora de emitir información.
3. CReceptor: Encargada de procesar toda la información durante la recepción del mensaje, devolviendo los resultados por cada proceso que interviene a la hora de recibir información.
4. CMensaje: Contenedora de información posee todos los datos del mensaje enviado. Además realiza operaciones fundamentales como separar los símbolos de un mensaje, calculando la probabilidad de ocurrencia de cada uno de ellos.
5. CSymbol: Unidad más pequeña de información, en esta se almacena datos, el símbolo en específico con su respectiva probabilidad de ocurrencia en un mensaje determinado y su codificación a la hora de ser enviado a través del sistema de transmisión.
6. CTuberia: Es la intermediaria entre los procesos de transmisión, recibiendo el flujo de datos de salida de uno y transmitiendo esa salida como entrada al siguiente proceso que en este caso serían los filtros.
7. CAbstrac_Factory_filtro: Encargada de crear los Filtros.
8. CFiltro: Encargada de aplicarle los algoritmos de codificación compresión y cifrado a un mensaje en particular.

3.5.3 Realización de los casos de uso del diseño

Una realización de los casos de uso del diseño es una colaboración en el modelo de diseño que describe como se realiza un caso de uso en específico, y como se ejecuta en términos de clases del diseño y sus objetos, proporcionando una traza directa a una realización de caso de uso del análisis. Sin embargo cuando el modelo de análisis no va a mantenerse a lo largo del ciclo de vida del software, entonces no existen realización de casos de uso del análisis, por lo que la dependencia de traza sería una realización de casos de uso del diseño hacia el caso de uso en el modelo de casos de usos.[12]

En estas realizaciones se obtiene una descripción de flujo de eventos, diagramas de clases que muestran las clases de diseño participantes y diagramas de iteración que muestran la realización de un flujo o escenario concreto de un caso de uso.

3.5.3.1 Diagrama de clases del diseño

Un diagrama de clases de diseño es un diagrama que muestra un conjunto de interfaces, colaboraciones y sus relaciones. Los diagramas de clases de diseño se utilizan para modelar

principalmente la vista de diseño estática de un sistema. Esto incluye modelar el vocabulario del sistema, las colaboraciones o esquemas.

Una clase del diseño y sus objetos, además de los subsistemas de que las contienen a menudo participan en varias realizaciones de casos de uso. También puede darse el caso de algunas operaciones, atributos y asociaciones sobre una clase específica que son relevantes para una realización de casos de uso. Por estas razones es que se utilizan diagramas de clases conectados a una realización de caso de uso, mostrando sus clases participantes subsistemas y relaciones. En el sistema que se está desarrollando fueron determinados 8 casos de uso es por esto que se generarán 8 diagramas de clases:

1. Diagrama de clases del diseño CU Procesar Mensaje. Ver Anexo 16
2. Diagrama de clases del diseño CU Mostrar Pasos. Ver Anexo 17
3. Diagrama de clases del diseño CU Codificar. Ver Anexo 18
4. Diagrama de clases del diseño CU Descodificar. Ver Anexo 19
5. Diagrama de clases del diseño CU Cifrar. Ver Anexo 20
6. Diagrama de clases del diseño CU Descifrar. Ver Anexo 21
7. Diagrama de clases del diseño CU Comprimir. Ver Anexo 22
8. Diagrama de clases del diseño CU Descomprimir. Ver Anexo 23

3.5.3.2 Diagrama de iteración

La secuencia de acciones en un caso de uso comienza con la acción del actor que lo invoca. Si se considera el interior del sistema entonces se tendrá algún objeto del diseño que recibe un mensaje de dicho actor. Este objeto puede enviar mensajes a otros objetos implicados, existiendo una iteración entre ellos para realizar un caso de uso en concreto. Los mensajes entre objetos vienen siendo las operaciones o funciones que uno de ellos solicita a otro. En el sistema que se está desarrollando se hará el empleo de los diagramas de secuencia correspondientes a cada caso de uso por lo que se obtienen 8 diagramas en los que se reflejan como interactúan los objetos que participan en los mismos.

1. Diagrama de iteración CU Procesar Mensaje. Ver Anexo 24
2. Diagrama de iteración CU Mostrar Pasos. Ver Anexo 25
3. Diagrama de iteración CU Codificar. Ver Anexo 26
4. Diagrama de iteración CU Descodificar. Ver Anexo 27
5. Diagrama de iteración CU Cifrar. Ver Anexo 28
6. Diagrama de iteración CU Descifrar. Ver Anexo 29

7. Diagrama de iteración CU Comprimir. Ver Anexo 30
8. Diagrama de iteración CU Descomprimir. Ver Anexo 31

Conclusiones

En el presente capítulo se trataron temas importantes en cuanto al diseño de la aplicación, centrándose en el flujo de trabajo de diseño, generando los artefactos más importantes que se proponen en este flujo, por lo que se puede llegar a la siguiente conclusión:

1. La aplicación del patrón arquitectónico Tubería y Filtros proporciona varias ventajas para la construcción del sistema.
2. Se agrupan las clases del diseño en paquetes para una mejor organización, describiendo a cada uno de ellos respecto a las clases contenidas y sus relaciones.
3. A cada caso de uso se le genera su realización correspondiente obteniendo un diagrama de clases que resuelve a cada uno y su respectivo diagrama de iteración para observar como es que interactúan los objetos para realizar el caso de uso en cuestión.
4. Se hace una descripción verbal y detallada de cada una de las clases esenciales determinadas en el diseño para facilitar la implementación de las mismas.

CAPÍTULO 4: IMPLEMENTACIÓN Y PRUEBA

Introducción

En este capítulo se crearán los artefactos más importantes en el flujo de trabajo de implementación y pruebas, utilizando como entrada los generados en el capítulo anterior, o sea, ahora se implementarán en términos de componentes los resultados del flujo de diseño. Entre los artefactos que se obtendrán están los componentes y su diagrama correspondiente, los subsistemas de implementación con sus dependencias. También se dedicará un espacio al flujo de trabajo de prueba para comprobar que lo que se ha realizado hasta el momento posee la calidad necesaria y así evitar posibles errores, dedicándoselo a las pruebas de unidad, además de que se propone un posible estándar de codificación a emplear a la hora de codificar.

4.1 Estándar de codificación

Los estándares de codificación han sido un elemento crucial para los desarrolladores de software debido a que gracias a ello han logrado una mejor comunicación a la hora de interpretar lo que ha realizado el compañero de trabajo. Además a la hora de que un programador necesite entender lo que hizo otro es más fácil si ambos están codificando con el mismo estándar, esto sirve como cierto lenguaje de comunicación entre ellos. Por estas razones se le dedica un espacio a la creación de un estándar de codificación para su aplicación y entender en un momento dado el componente que se halla programado por cualquiera de los programadores.

4.1.1 Ventajas de un estándar de codificación

1. Asegurar la legibilidad del código entre distintos programadores, facilitando el debugging del mismo.
2. Proveer una guía para el encargado de mantenimiento/actualización del sistema, con código claro y bien documentado.
3. Facilitar la portabilidad entre plataformas y aplicaciones.
4. Facilidad a la hora de entender el código de otro desarrollador que aplica el mismo estándar.

Es por esto y mucho más que la codificación de los módulos del Sistema a desarrollar debe cumplir ciertos requisitos, detallados a continuación:

Operadores binarios

Deben de incluirse espacios en ambos lados de los operadores binarios.
Ejemplo: y = 50 + 15 – z;

Operadores Unarios
Los operadores unarios (++, --, etcétera) deben ponerse junto a su operando.
Ejemplo: v++; m --;

Nombres de variables locales
Los nombres de algunas variables locales, como los iteradores o los contadores, pueden especificarse en minúscula y de forma abreviada, siempre que su contexto sea específicamente local y su lectura sea intuitiva. Ejemplo: **int** cont, **int** i, **int** j.

Identación
Utilizar el formato que contiene el IDE empleado, en este caso el Netbeans.
Ejemplo:

```
 if (<instrucción>){
 }else{
 }
for (int i = 0 ; i < n ; i++ ){
    for (int j = 0 ; j < n+2 ; j++ ){
}
}
public int Suma(){
}
```

Estructuras de control
Las estructuras **if**, **for**, **while**, **switch**, etc. Deben tener un espacio entre la palabra clave y el paréntesis de apertura, para diferenciarlos de las llamadas a funciones.
En caso de anidamiento el contenido debe de estar a 4 espacios.
Ejemplo:

```
 if (<instrucción>){
     if (<instrucción>) {
     }
```

```
}
for (int i = 0 ; i < n ; i++ ){
   for (int j = 0 ; j < n+2 ; j++ ){
   }
}
```

Comentarios

Utilizar el estándar // solo para comentarios de una sola línea en caso de que sea extenso utilizar el estándar /* */

Declaración de clases

La declaración de las clases debe de ser descriptiva o con abreviaciones comenzando obligatoriamente con la letra C mayúscula de **class**, en caso de ser con abreviación poner descriptivamente al lado comentario con lo que representa comenzando obligatoriamente con mayúsculas.

Ejemplo:

```
public class CCodifDecodi{
}
public class CSTD{ //Sistema de Transmisión de Datos
}
```

Declaraciones de atributos

Los atributos de una clase deben de comenzar con letra inicial minúscula y reflejar verdaderamente la propiedad a la que representa.

Ejemplo: nombre, edad, sexo, color

Declaración de funciones

Las funciones pueden o no comenzar con letra inicial mayúscula.

Ejemplo: Insertar () o insertar ().

Interfaces

La declaración de las interfaces debe de comenzar con la vocal I mayúscula de interface.

Ejemplo: IList, IFilter, ISerializable etcétera.

Enumeraciones

Las enumeraciones deben de comenzar con la vocal E mayúscula de enumeración
Ejemplo: EColores, EDiasSemana

Uso de instrucciones de ruptura break, continue, goto , etcétera

Tratar de evitar el uso de instrucciones como **break**, **continue** y **goto** que el empleo de las mismas es considerado una mala práctica de programación.
Ejemplo:

```
        //No                                    //Si
for (int i = 0 ; i < n ; i++ ){        for (int i = 0 ; i < n ; i++ ){
   if( i % 2 == 0)                        if( i % 2 == 0)
      break;                                  i = n;
}                                      }
```

Interfaces Visuales

En cuanto a la interfaces visuales, quienes interactúan con el usuario, nombrarlas comenzando por la palabra Form seguida de un nombre. Los demás componentes visuales se llamarán normalmente por el nombre que le asigna el IDE empleado comenzando por la letra j.

Ejemplo:
Para los componentes JFrame que son los formularios: FormPrincipal, FormCodificar, etcétera. Para los restantes componentes dejar que el IDE les asigne el estándar que lleva incluido.

4.2 Implementación

La entrada a este flujo de trabajo son los artefactos generados en le capítulo anterior, es decir, el Modelo de Diseño. Cada una de las clases, diagramas, subsistemas y paquetes son implementados aquí, se traducen todos en términos de componentes; es decir ficheros de código fuente, scripts, ficheros lógicos, ficheros binarios, ejecutables y similares. Este flujo de trabajo es el centro durante la fase de elaboración, para crear la línea base ejecutable de la arquitectura, y durante la fase de transición, para tratar defectos tardíos. También es bueno aclarar que se obtiene el modelo de

implementación que describe como los elementos del diseño se implementan en términos de componentes.

4.3 Diagrama de componentes

Diagrama de componentes: Conjunto de componentes que se relacionan entre si mediante dependencias, estos pueden ser ejecutables, librerías, ficheros de código, una tabla de base de datos, un documento etcétera.

En el sistema desarrollado los componentes son ficheros con extensión .java que contienen código Java, se genera el ejecutable de extensión .class. Esto se puede apreciar en la Figura 12 , en la cual aparecen cada uno de ellos agrupados en paquetes.

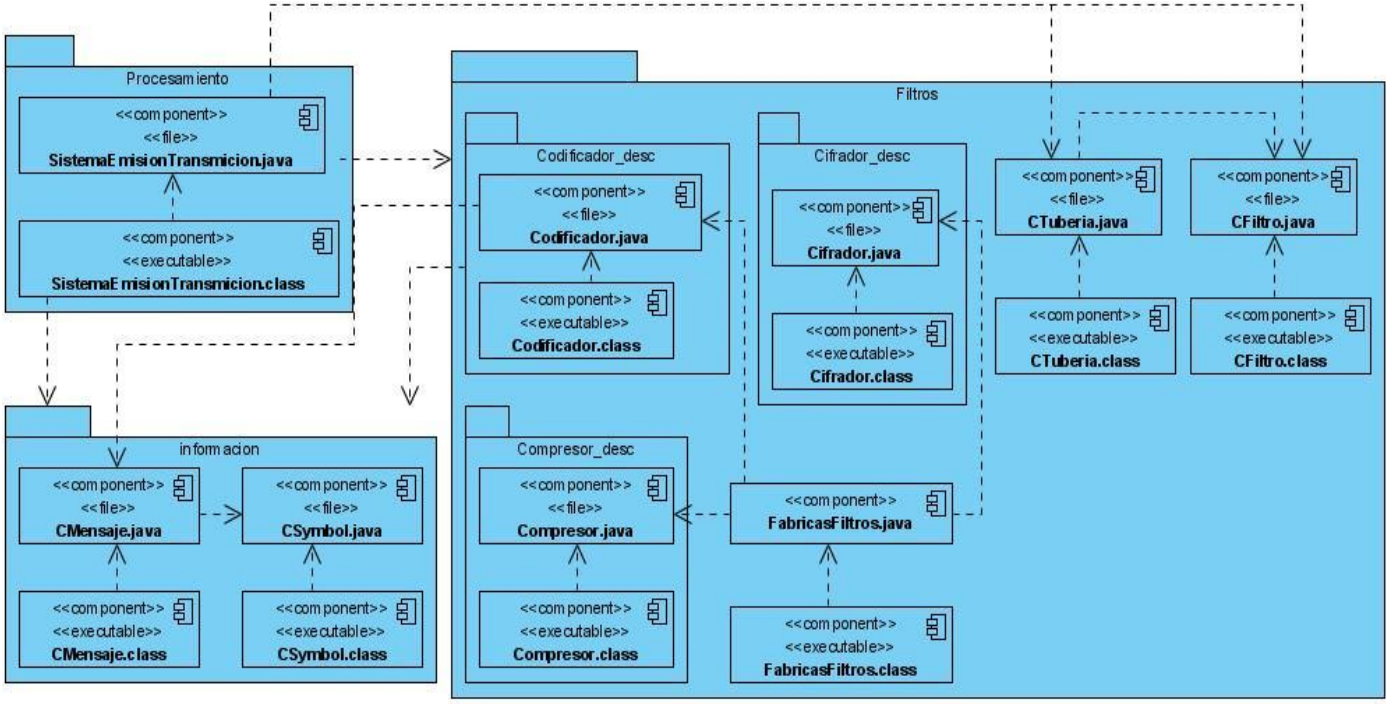


Figura 12: Diagrama de Componentes

4.4 Paquetes de implementación

Los paquetes de implementación proporcionan una forma de organizar los artefactos del modelo de implementación en partes manejables. Un paquete puede estar formado por componentes, interfaces y otros paquetes. Estos están muy relacionados con los paquetes de diseño en el modelo del diseño, por

lo que los paquetes de implementación deberían seguir la traza uno a uno de sus paquetes de diseño correspondientes.[12]

A continuación se mencionan cumpliendo con lo antes mencionado los paquetes de implementación:

1. Información
2. Filtros
3. Visual
4. Codificador_desc
5. Cifrador_desc
6. Compresor_desc

4.4.1 Paquete de implementación Procesamiento

El paquete está compuesto por el componente SistemaEmisionTransmision.java el cual implementa o engloba la implementación de las clases de diseño CSTD, CEmisor y CReceptor, además este paquete está provisto del ejecutable SistemaEmisionTransmision.class. Ver Anexo 32.

4.4.2 Paquete de implementación Información

El paquete está compuesto por CMensaje.java y CSymbol.java implementando a sus respectivas clases CMensaje y CSymbol, además de contener los componentes ejecutables correspondientes. Ver Anexo 33.

4.4.3 Paquete de implementación Filtros

Este es el más complejo, pues está provisto de otros paquetes como son: el Codificador_desc, Cifrador_desc y Compresor_desc, además contiene a los componentes CTuberia.java, CFiltro.java y FabricasFiltro.java con sus respectivos ejecutables, implementando a las clases CTuberia, CFiltro y toda la jerarquía de fábricas productoras de filtros. Ver Anexo 34.

4.4.4 Paquete de implementación Visual

Contiene los componentes visuales, que generan la interfaz de usuario, permitiendo la interacción entre el usuario y la aplicación. Ver Anexo 35.

4.4.5 Paquete de implementación Codificador_desc

Contiene al componente Codificador.java y su respectivo ejecutable Codificador.class, implementando al conjunto de clases que realizan la codificación por cualquiera de los algoritmos impartidos en clases. Ver Anexo 36.

4.4.6 Paquete de implementación Cifrador_desc

Contiene al componente Cifrador.java y su respectivo ejecutable Cifrador.class, implementando al conjunto de clases que realizan el cifrado por cualquiera de los algoritmos de cifrado impartidos en clases. Ver Anexo 37.

4.4.7 Paquete de implementación Compresor_desc

Contiene al componente Compresor.java y su respectivo ejecutable Compresor.class, implementando al conjunto de clases que realizan la compresión por cualquiera de los algoritmos impartidos en clases. Ver Anexo 38.

4.5 Pruebas del Sistema

En este flujo de trabajo es verificado el resultado de la implementación probando cada construcción, incluyendo internas e intermedias, así como las versiones finales del sistema a ser entregadas a terceros.[12]

Las pruebas tienen gran importancia para cualquier software. Pressman en su libro: "Ingeniería de Software. Un enfoque práctico" define algunos objetivos que demuestran claramente lo antes dicho.

1. La prueba es el proceso de ejecución de un programa con la intención de descubrir un error.
2. Un buen caso de prueba es aquel que tiene una alta probabilidad de mostrar un error no descubierto hasta entonces.
3. Una prueba tiene éxito si descubre un error no detectado hasta entonces.

Cada una de las pruebas realizadas demuestra como se cumplen las funcionalidades y requisitos que deben satisfacer el sistema, sirven como medidor para saber la fiabilidad del software. Con este objetivo se realizaron dos tipos de prueba; las de caja blanca y las de unidad. El porque de esta elección fue básicamente el propósito final que tiene el software; servir de ayuda a la asignatura de Teleinformática I, tanto para los profesores como para los estudiantes. Seleccionando como juegos de datos los mismos ejercicios orientados en los documentos de clases.

4.5.1 Pruebas de Caja Blanca

Este tipo de prueba también llamadas Pruebas de Caja de Cristal garantizan que por lo menos se ejecute una vez cada camino independiente de cada módulo o funcionalidad, que se ejerciten todas las variantes lógicas, se ejecutan todos los ciclos con sus limitaciones y ejercitan las estructuras de datos para asegurar su validez.

Estas pruebas se realizaron a medida que se iban implementando las diferentes funcionalidades que debía tener la aplicación. Una vez implementada la arquitectura base del sistema que soporta cada

uno de los módulos se empezaron a desarrollar los diferentes algoritmos. Al finalizar cada uno se le asignaron valores a las variables de entrada y se comprobó si cumplían con la lógica de funcionamiento de cada uno de los procesos tratados y si los valores retornados eran los deseados. De esta manera se detectaron algunos errores, sobre todo en los ciclos y las condicionales, jugando estos un papel crucial, debido al tratamiento con las estructuras de datos y las cadenas que se hace extenso en la mayoría de las clases que tienen grandes responsabilidades, como son las de codificar y decodificar, comprimir y descomprimir. También se encontraron defectos en los métodos recursivos y las funciones que necesitan parámetros por referencia; todos estos fueron corregidos después de realizar las pruebas mencionadas.

Una vez terminado de implementar todos los algoritmos y probados como anteriormente se dijo, se pasó a una nueva fase de pruebas para estar totalmente seguro de su fiabilidad; las pruebas de unidad.

4.5.2 Pruebas de Unidad

Las pruebas de unidad permiten chequear el funcionamiento de cada una de las clases, los módulos, librerías, e incluso hasta métodos y funciones específicas. Actualmente existen varias herramientas y frameworks que posibilitan realizarlas de manera sencilla y rápida, uno de estos es el JUnit, quien viene integrado con el IDE utilizado para la implementación del software, NetBeans 6.0.

Este IDE permite fácilmente seleccionar a cuales clases o métodos se les desean realizar las pruebas y mediante el uso de funciones específicas se le asignan valores a los datos de entrada y lo esperado como retorno, dando positivo si lo devuelto es igual a lo esperado y negativo en caso contrario.

Se decidió realizar las pruebas con los mismos ejercicios que se les orientan a los estudiantes en las Clases Prácticas (CP), las Conferencias, las Guías para los Trabajos de Controles e incluso algunos que salieron en el 1er Trabajo de Control Parcial (TCP) del curso 2007-2008.

Se probaron los algoritmos fundamentales impartidos en clases: Huffman, Shannon-Fano, Hamming, César con clave y sin clave, RLE, MNP, Aritmético y LZ77. En todos los casos se realizaron más de dos ejercicios, siendo positivos los resultados en todos los casos, en la Figura 13 se ven los resultados obtenidos a los casos de pruebas realizados con el framework JUnit, siendo satisfactorios.

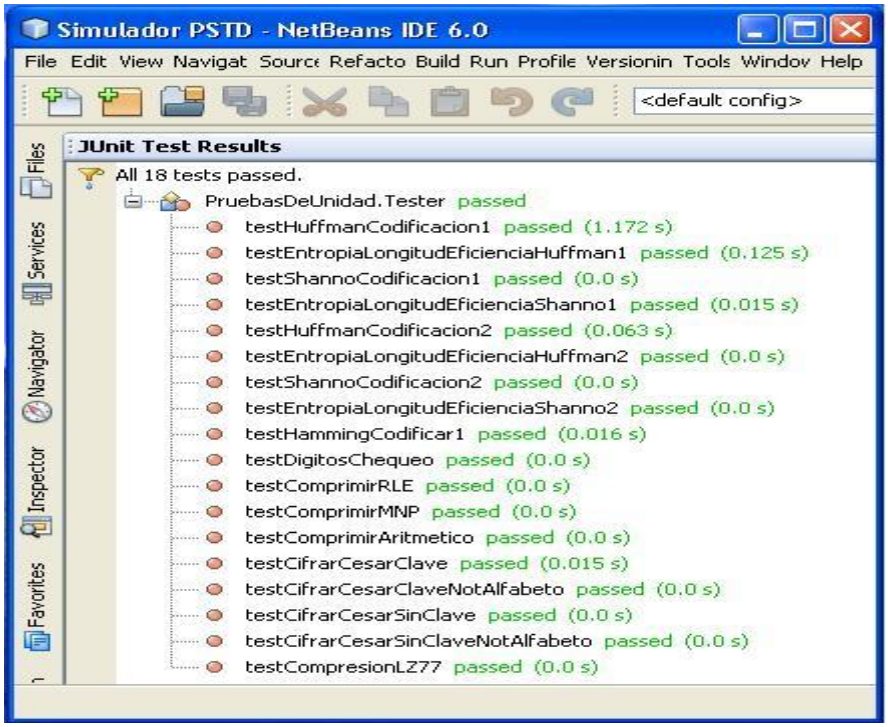


Figura 13: Netbeans 6.0 con JUNIT

Conclusiones

En este capítulo se abordaron los temas correspondientes a la Implementación y Pruebas, de todo el trabajo realizado se obtuvieron varios artefactos que posibilitaron llegar a las siguientes conclusiones:

1. Se propone un estándar de codificación siguiendo la misma estructura que posee el IDE seleccionado, posibilitando así el entendimiento entre ambos programadores y facilitando la actualización y mejoramiento del sistema en años posteriores.
2. Se crearon los paquetes de implementación, coincidiendo cada uno con los de diseño, demostrando la trazabilidad del diseño hasta la implementación.
3. Se realizaron pruebas de unidad mediante el framework JUnit, seleccionando como casos de pruebas ejercicios orientados en la asignatura, comprobando así la funcionalidad del sistema.
4. Al terminar este capítulo quedó construido completamente la versión inicial del software, al cual se le decidió llamar: “Simulador PSTD”, quedando listo para ponerlo en explotación.

CONCLUSIONES

Después de concluir el presente trabajo y haber logrado el cumplimiento de las metas trazadas al inicio del desarrollo del mismo se arribaron a las siguientes conclusiones:

1. El estudio realizado previamente sirvió para la selección de la metodología, el lenguaje de programación y el entorno de desarrollo a utilizar en la elaboración del software.
2. Se obtuvieron los artefactos pertenecientes al Flujo de Trabajo de Diseño, muy importante para la próxima fase de desarrollo, la implementación.
3. Se implementó la primera versión del software, donde se incluyen los procesos y algoritmos esenciales de la asignatura. Cumpliendo de esta forma con las funcionalidades básicas necesarias.
4. Se sometió a pruebas tomando como juegos de datos ejercicios pertenecientes a CP, CTP y TCP. En todos los casos los resultados obtenidos demuestran la calidad del software.
5. La realización de los procesos de codificación, tanto eficiente como redundante, así como los de cifrado y compresión de forma independiente, le van a permitir a los estudiantes comprobar los resultados de los ejercicios orientados en clases.
6. El proceso completo da una visión más general y realista a los estudiantes de verdaderamente como funcionan estos algoritmos y procesos en la transmisión de datos.
7. El sistema fue avalado por el Asesor Técnico Docente del Departamento de Sistemas Digitales Yohandri Gil Ril, quien determinó que el mismo cumplía con todos los requisitos acordados. Ver Anexo 39

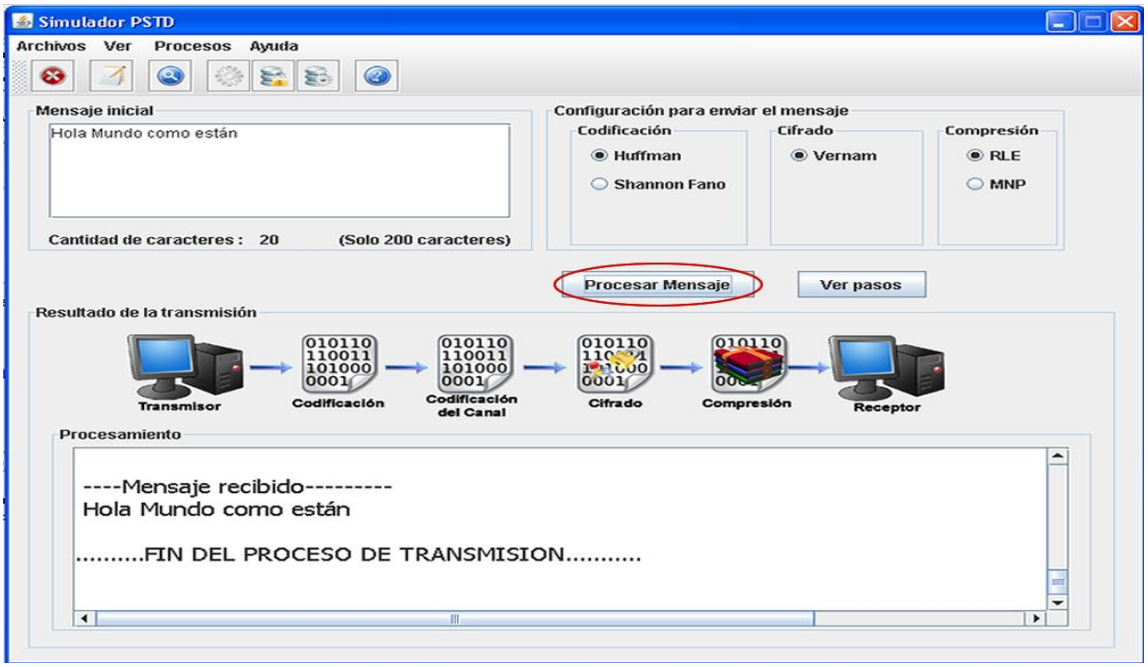
RECOMENDACIONES

1. Utilizarlo en cursos posteriores para evaluar su impacto en los estudiantes y los resultados de la asignatura.
2. Incluirle otros algoritmos dados en clases, como son: el Cíclico, la Transposición Columnar con clave y sin clave.
3. Dar la posibilidad de codificar con otro alfabeto que no sea el binario y seleccionar el orden de codificación deseado.
4. Mostrar el algoritmo de cifrado Vernam como proceso independiente.
5. Brindar la funcionalidad de salvar y cargar ficheros con los datos de los ejercicios resueltos.

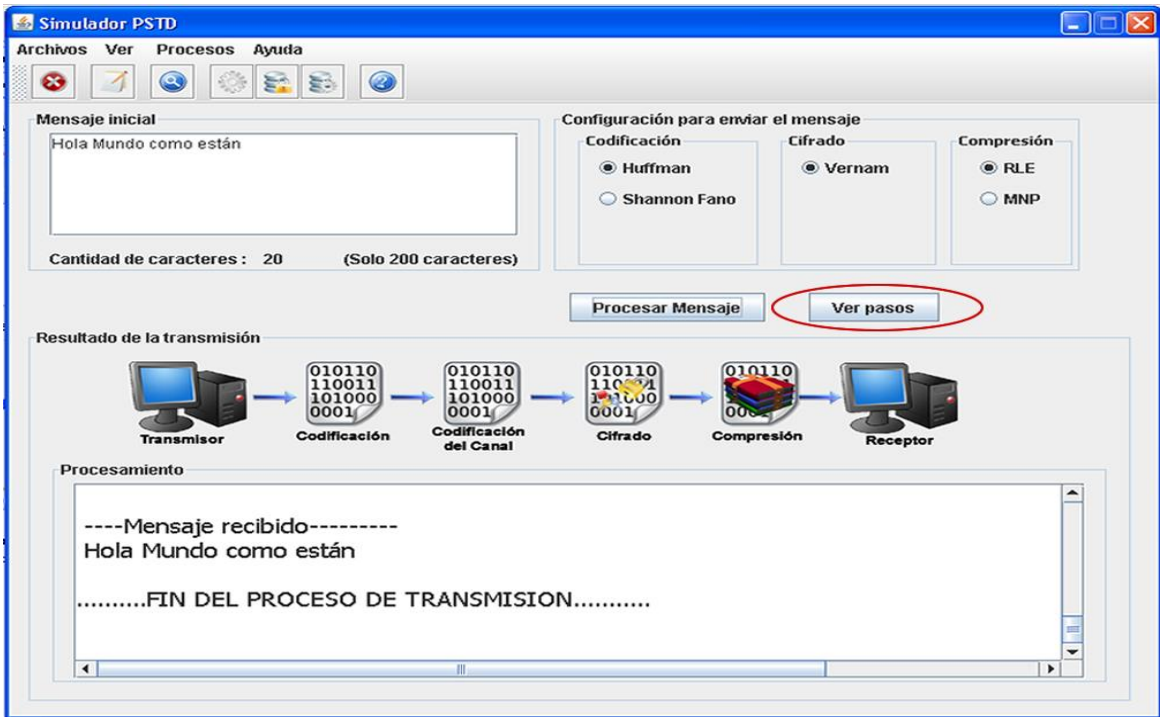
BIBLIOGRAFÍA

1. Valladolid, U.d. *Conceptos generales sobre codificación*. [cited; Available from: http://www.isa.cie.uva.es/proyectos/codec/marco1.html.
2. Aguirre, J.R., *Libro electrónico de seguridad informática y criptografía*. 2005: Madrid.
3. López, M.J.L., *Criptografía y seguridad en computadores*. 2da ed. 1999.
4. Salomon, D., *Data compression*. 3ra ed. 2002.
5. Stallings, W., *Comunicaciones y redes de computadores*. 6ta ed.
6. Márquez, J.E.B., *Transmición de datos*. 3ra ed. 2005.
7. *Software de simulación en telecomunicaciones digitales*. [cited; Available from: http://www.labvolt.com.
8. Daily Echeverría Sadradin, R.R.S., Jorge Cruz Peña. *QS Versión 1.0 Software simulador de redes de transmisión de datos y de telecomunicaciones con facilidad de análisis de tráfico, fallos y comportamiento.* [cited; Available from: http://www.cujae.edu.cu/eventos/cittel/trabajos/Trabajos/Comision%202/CITTEL-101.pdf.
9. Raúl Llinares Llopis, J.I.G., Andrés Camacho García. *Herramienta de simulación para transmición digital*. [cited; Available from: http://w3.iec.csic.es/ursi/articulos_gandia_2005/articulos/ED1/644.pdf.
10. Archer, T., *C# a fondo.* 2001.
11. Muñoz, A., *Java: del Grano a su Mesa.*
12. Ivar Jacobson, G.B., James Rumbaugh, *El proceso unificado de desarrollo de software.*
13. Teknoda, *¿Cómo conformar un entorno de programación java?* 2003.
14. Javier García de Jalón, J.I.R., Iñigo Mingo,Aitor Imaz,Alfonso Brazález,Alberto Larzabal,Jesús Calleja,Jon García, *Aprenda Java como si estuviera en primero.* 2000.
15. José H. Canós, P.L.y.M.C.P., *Metodologías ágiles en el desarrollo de software.* 2003.
16. Sanchez, M.A.M., *Metodologías de desarrollo de software.* 2002.
17. *Visual Paradigm for UML*. 2007 [cited; Available from: http://www.freedownloadmanager.org/es/downloads/Paradigma_Visual_para_UML_(M%C3%8D)_14720_p/.
18. Pressman, R.S., *Ingenería de software un enfoque práctico*. 5ta ed.
19. Larman, C., *UML y patrones*. 1999.
20. Cooper, J.W., *Introduction to Design Patterns in C#*. 2002.
21. José F. Vélez Serrano, Á.S.C., Alfredo Casado Bernárdez,Santiago Doblas Álvarez, *Técnicas avanzadas de diseño de software.* 2005.
22. *Netbeans features*. [cited; Available from: http://www.Netbeans.org.
23. Laura Bermejo Sanz, E.G.M., *Eclipse como IDE.*
24. Rational, *Ayuda del Rational 2007.* 2007.

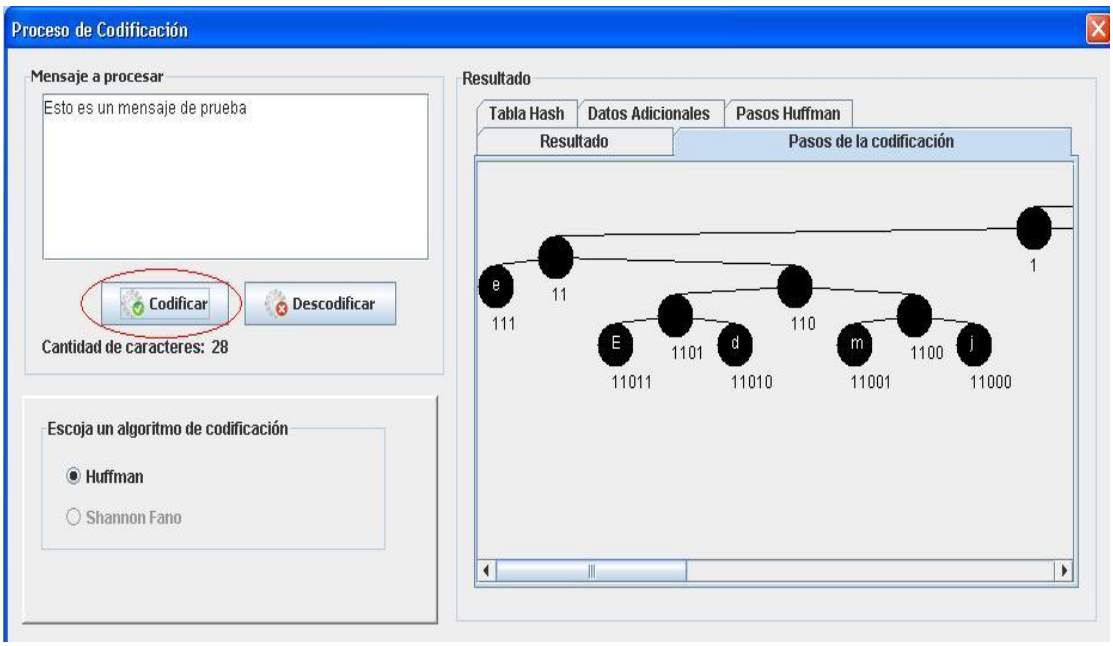
ANEXOS



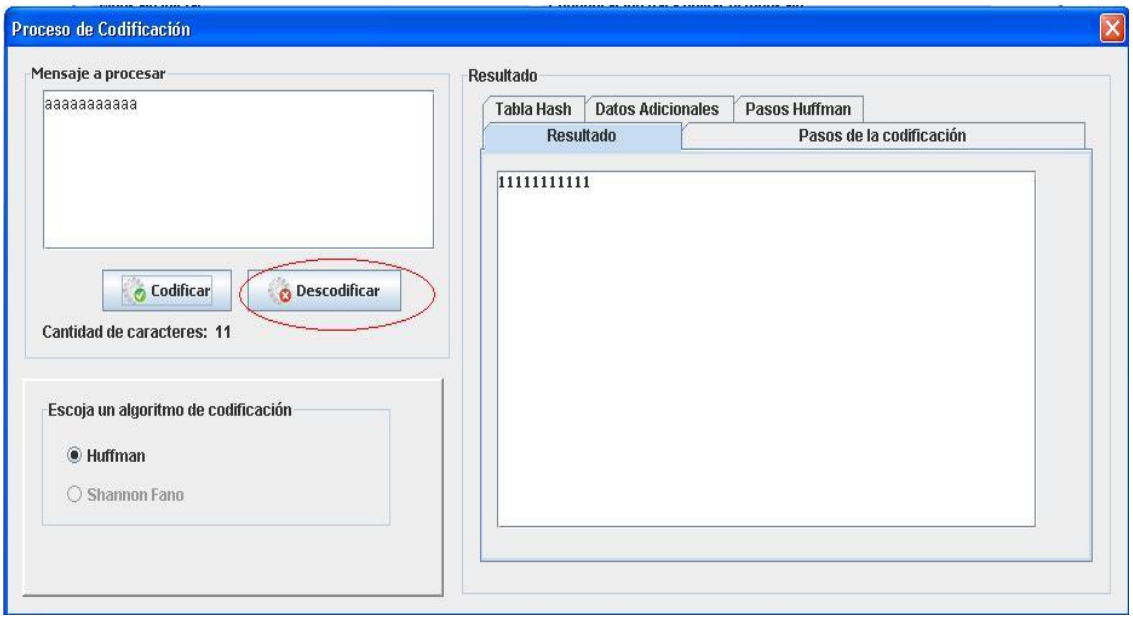
Anexo 1: Interfaz de usuario CU Procesar Mensajes



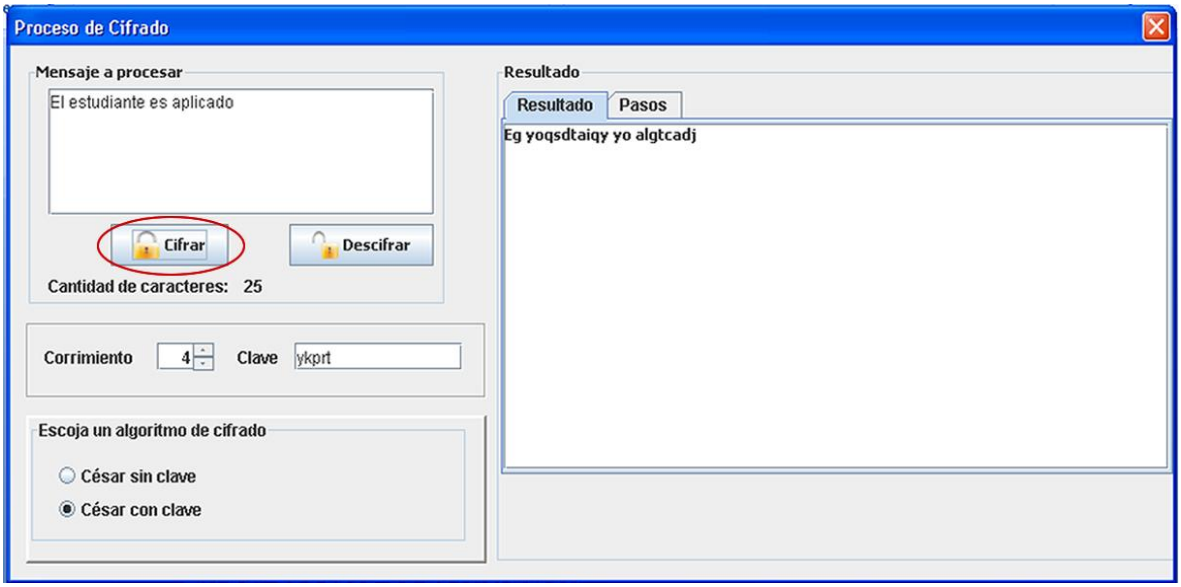
Anexo 2: Interfaz CU Mostrar Pasos



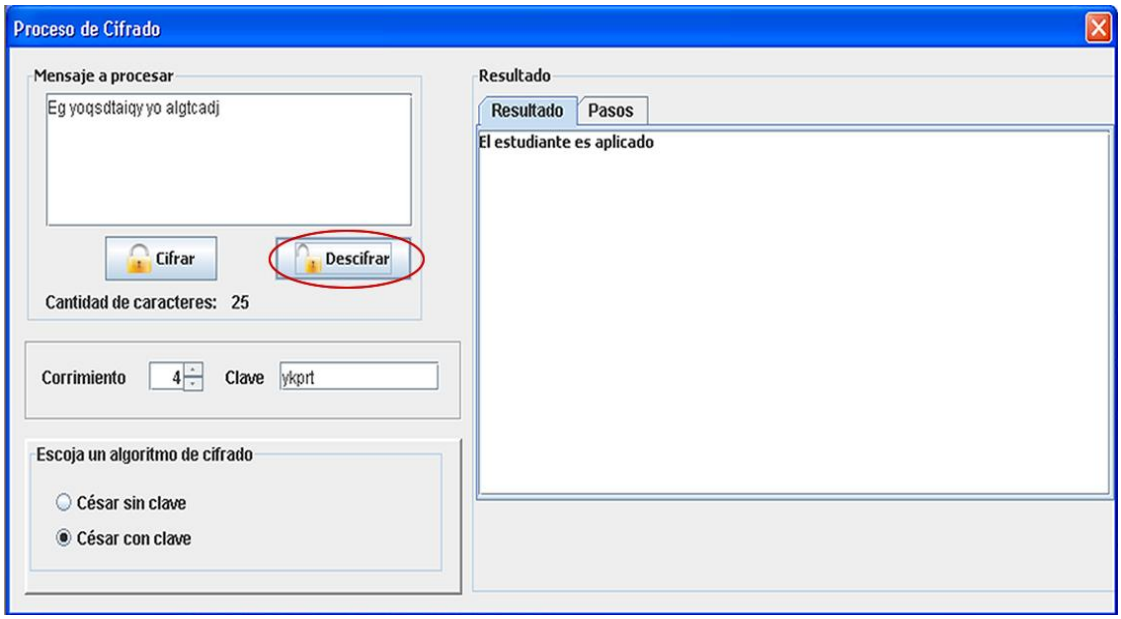
Anexo 3: Interfaz CU Codificar



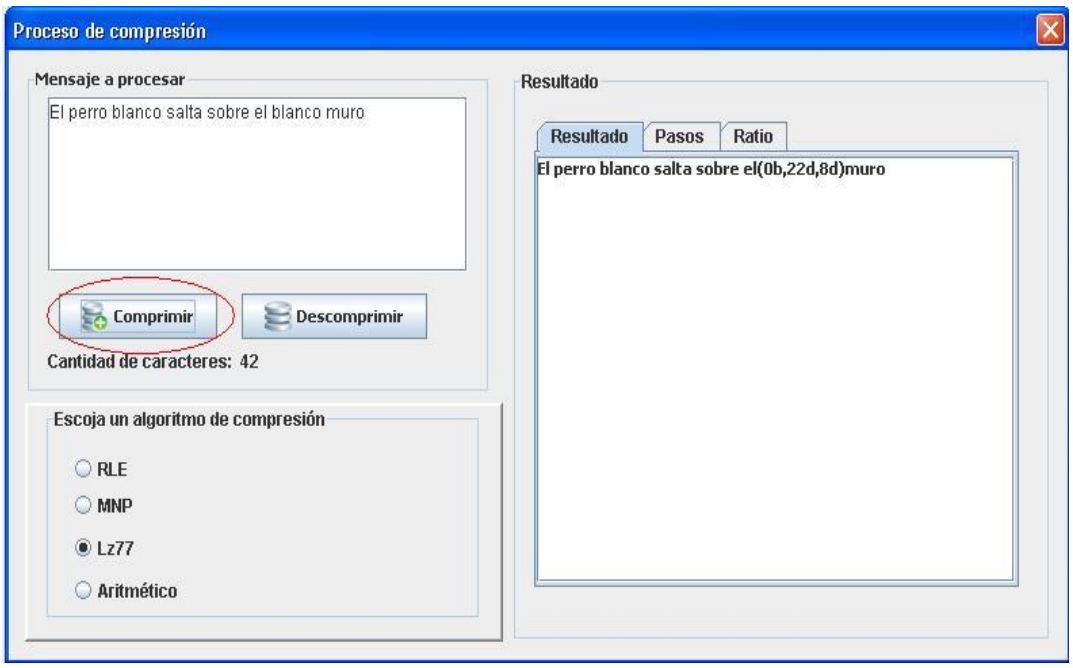
Anexo 4: Interfaz CU Descodificar



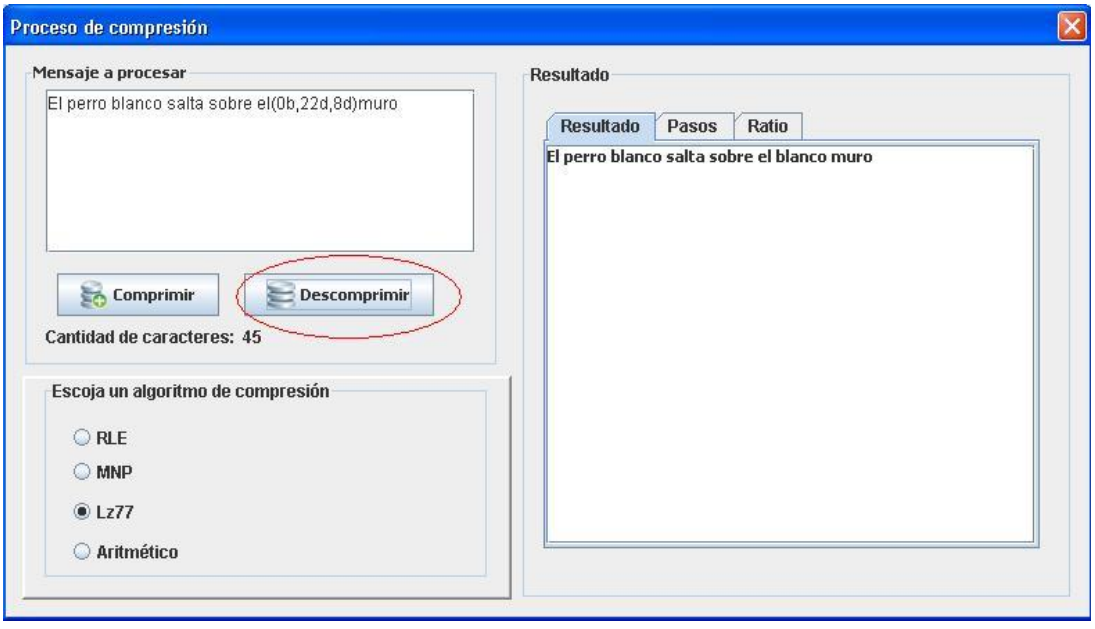
Anexo 5: Interfaz CU Cifrar



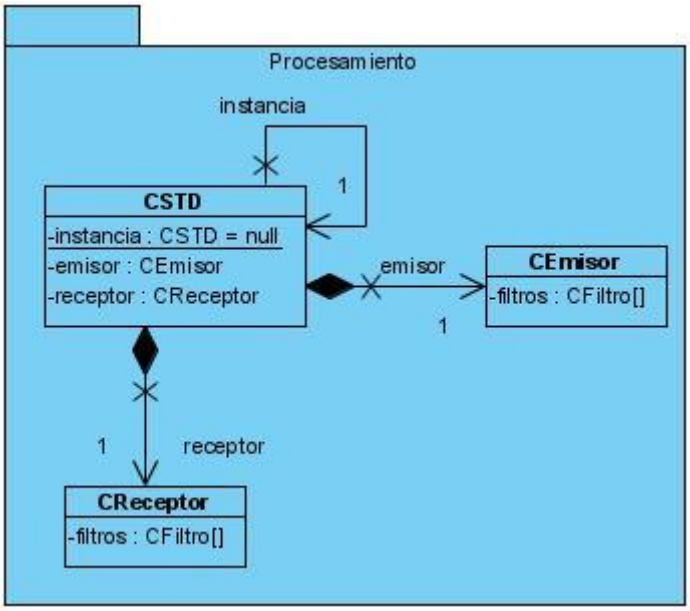
Anexo 6: Interfaz CU Descifrar



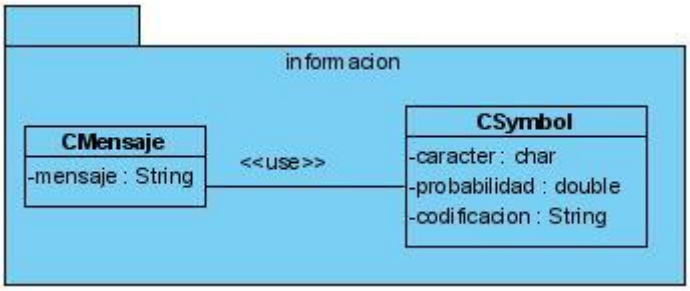
Anexo 7: Interfaz CU Comprimir



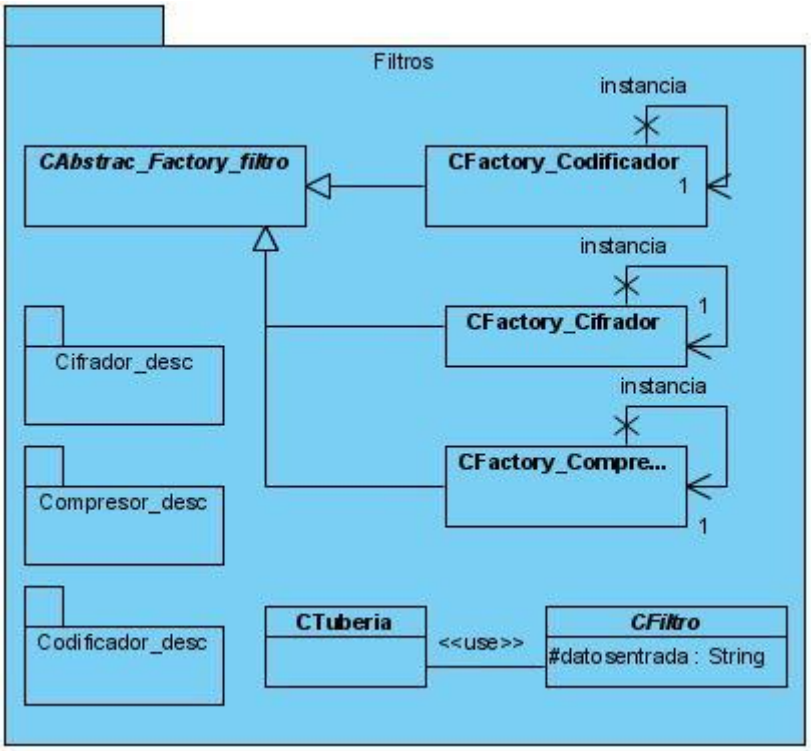
Anexo 8: Interfaz CU Descomprimir



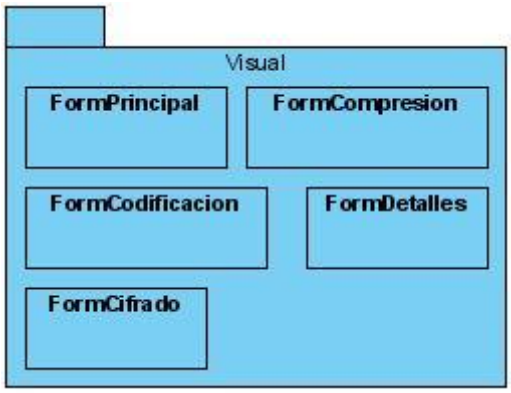
Anexo 9: Paquete de diseño Procesamiento



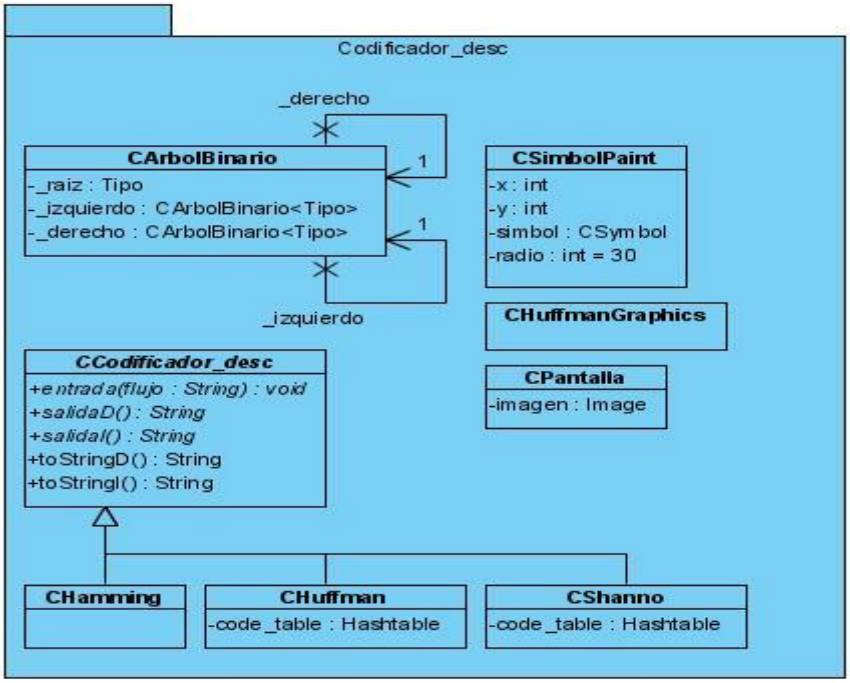
Anexo 10: Paquete de diseño Información



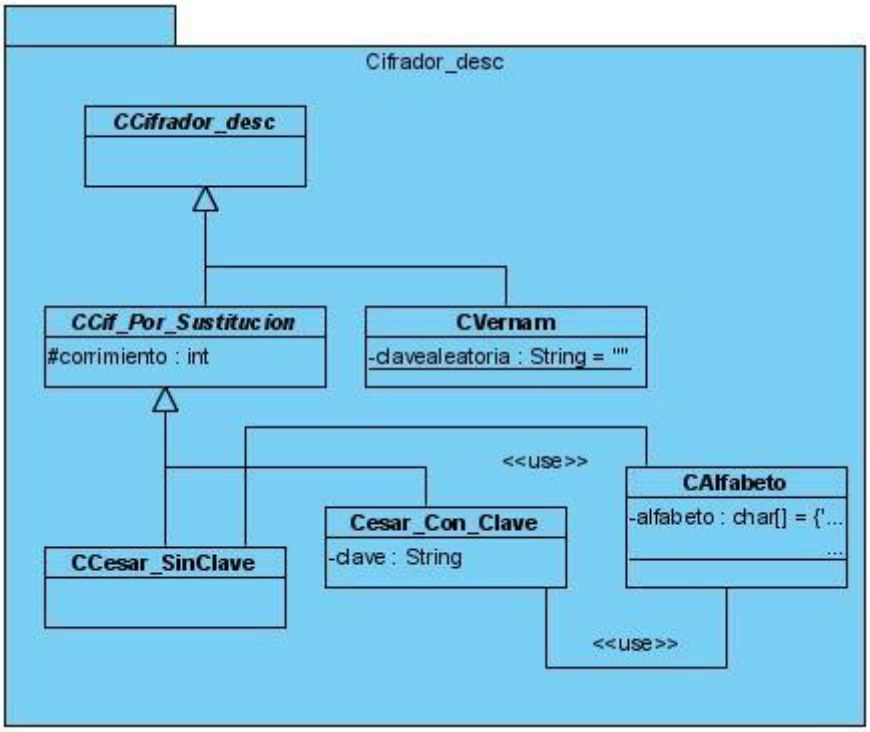
Anexo 11: Paquete de diseño Filtros



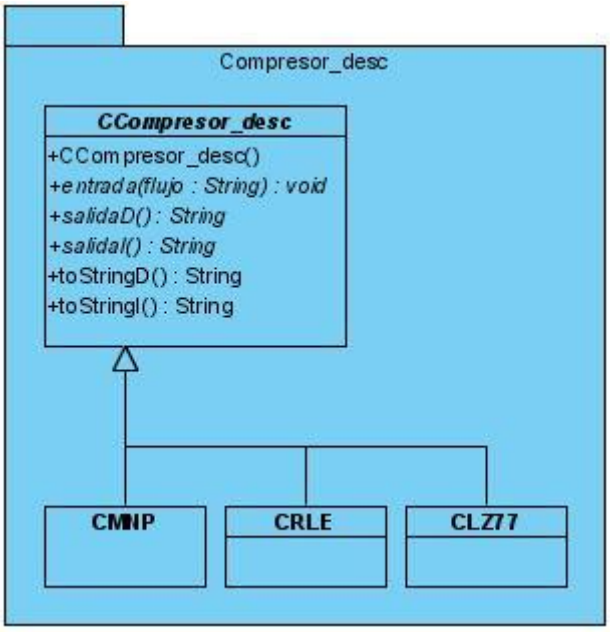
Anexo 12: Paquete de diseño Visual



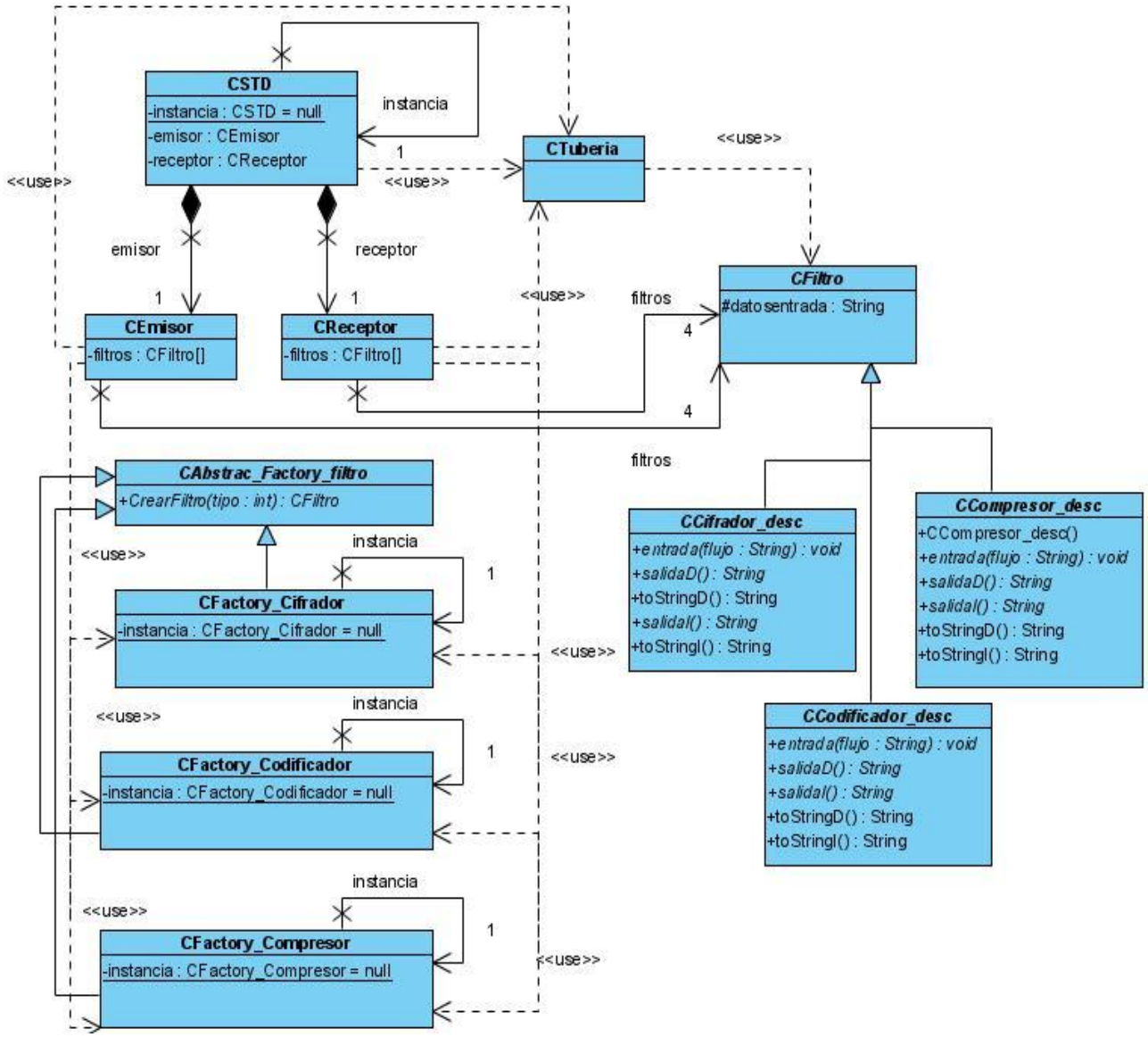
Anexo 13: Paquete de diseño Codificador_dec



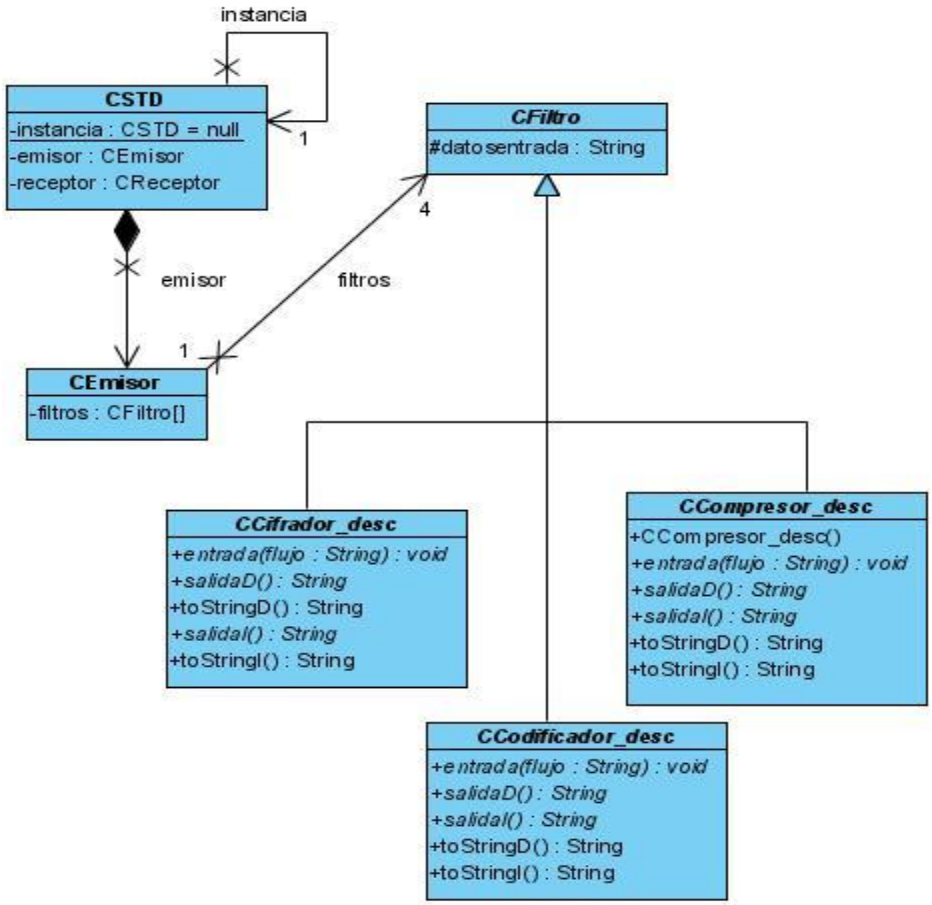
Anexo 14: Paquete de diseño Cifrador_dec



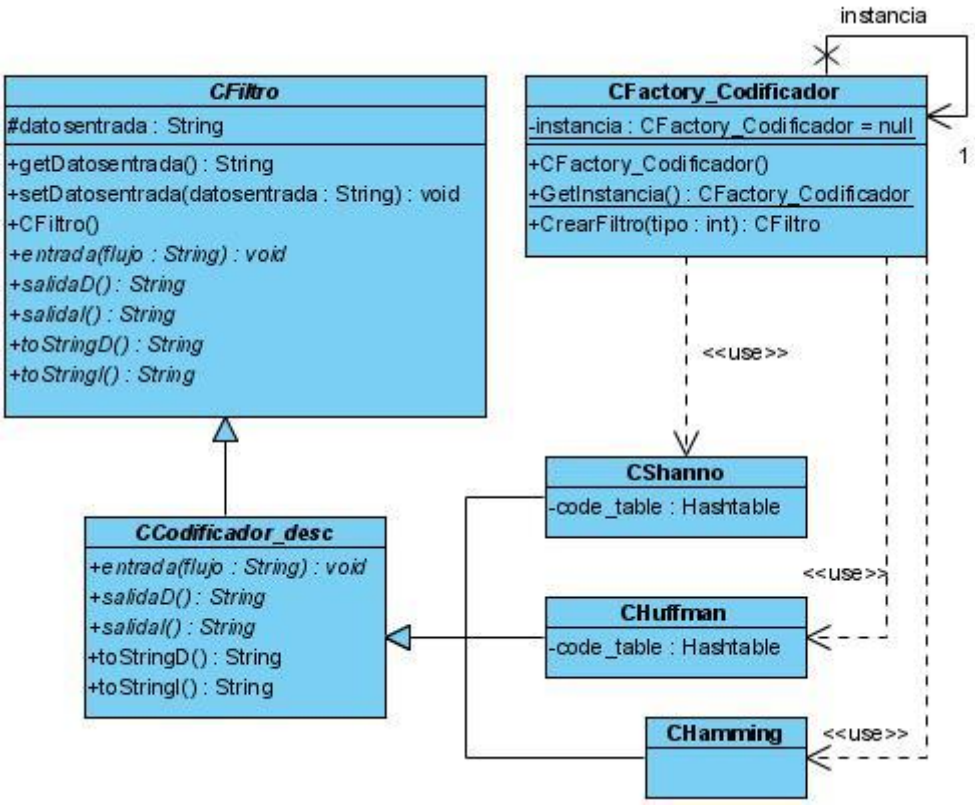
Anexo 15: Paquete de diseño Compresor_desc



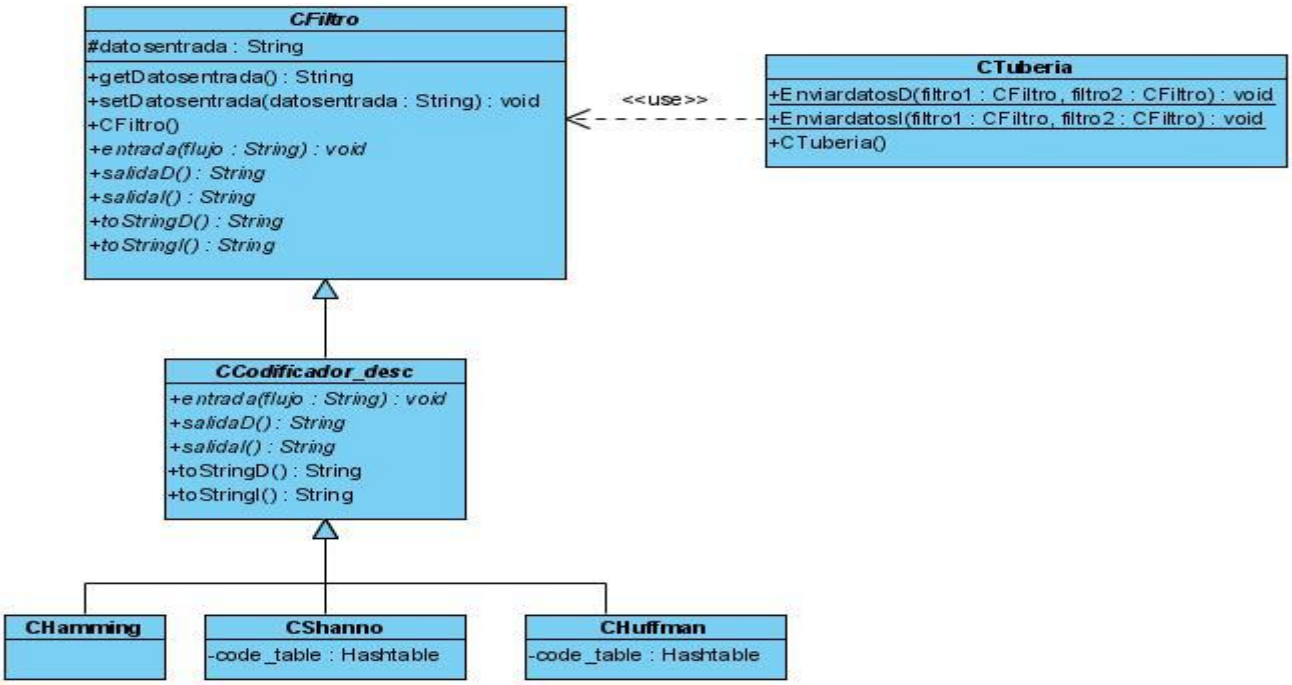
Anexo 16 Diagrama de clases del diseño CU Procesar Mensaje



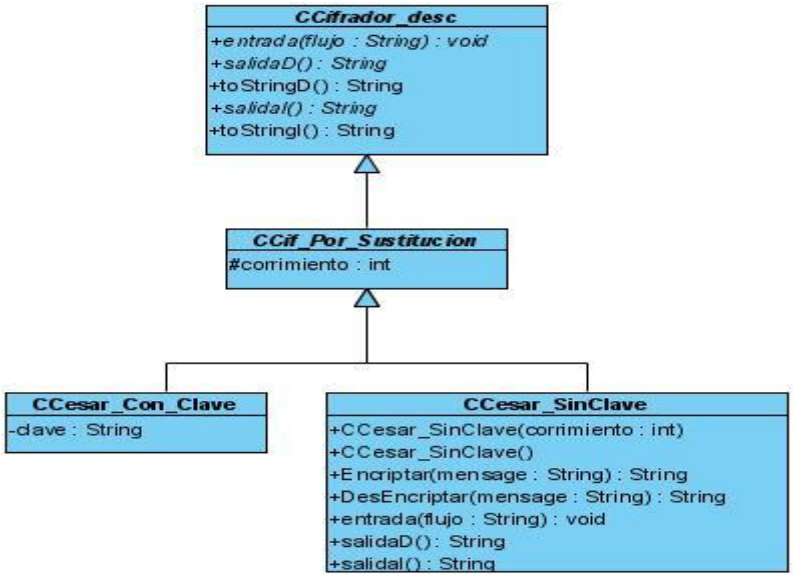
Anexo 17 Diagrama de clases del diseño CU Mostrar Pasos



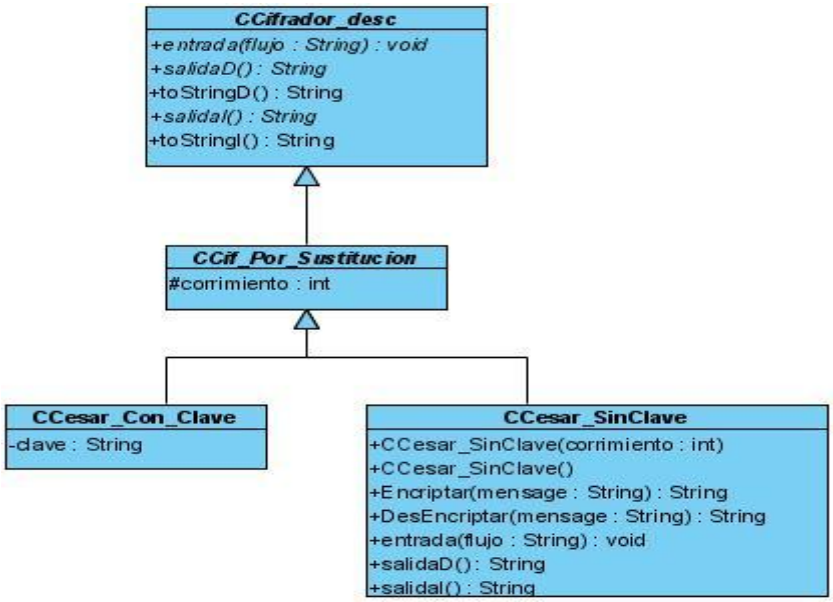
Anexo 18 Diagrama de clases del diseño CU Codificar



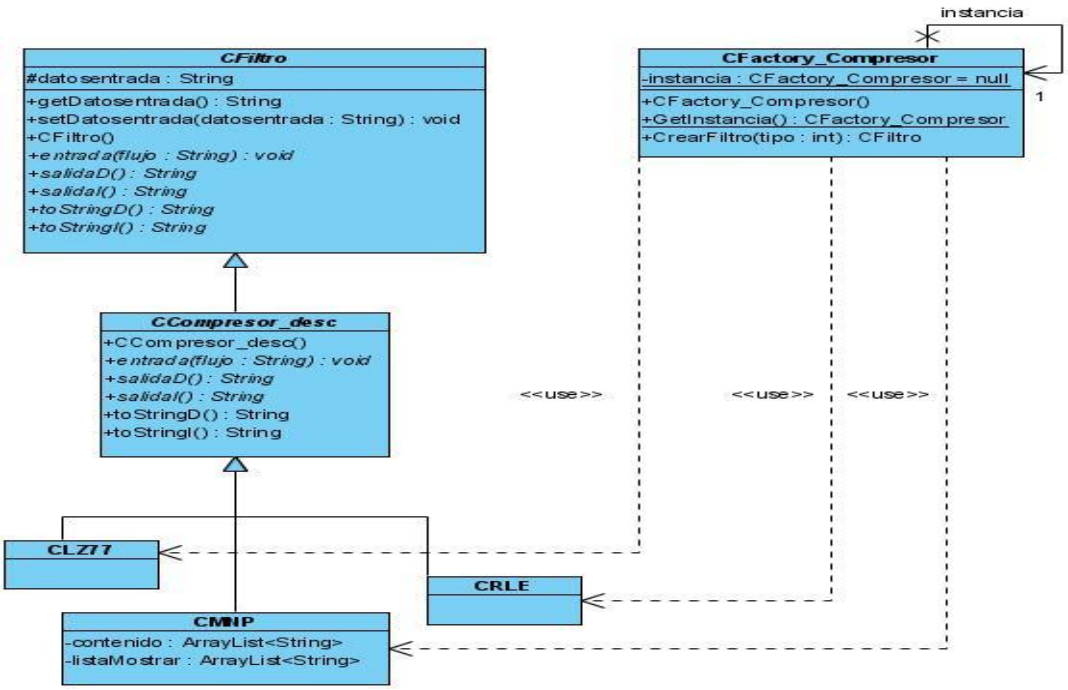
Anexo 19 Diagrama de clases del diseño CU Descodificar



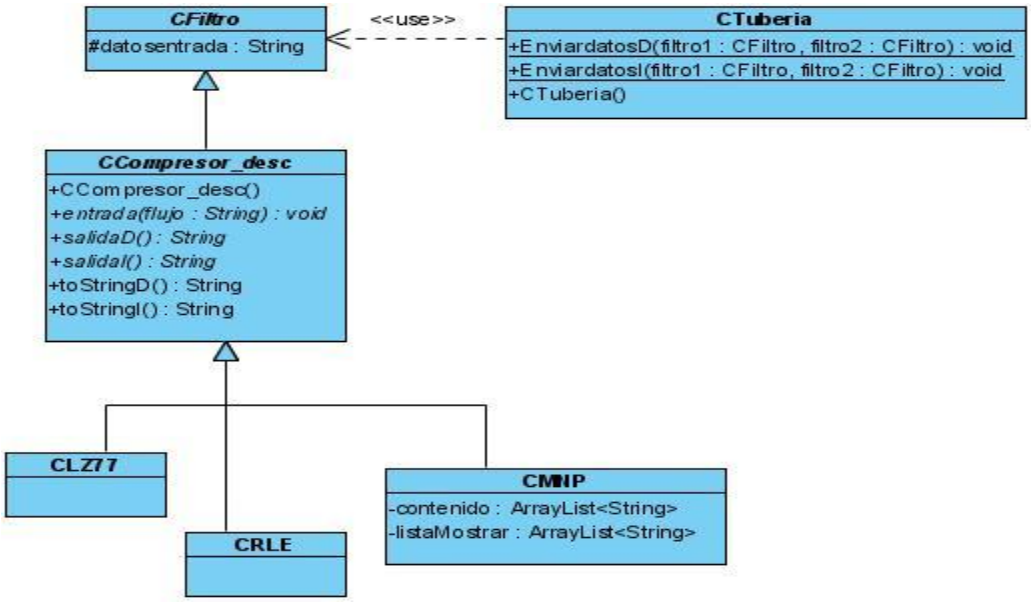
Anexo 20 Diagrama de clases del diseño CU Cifrar



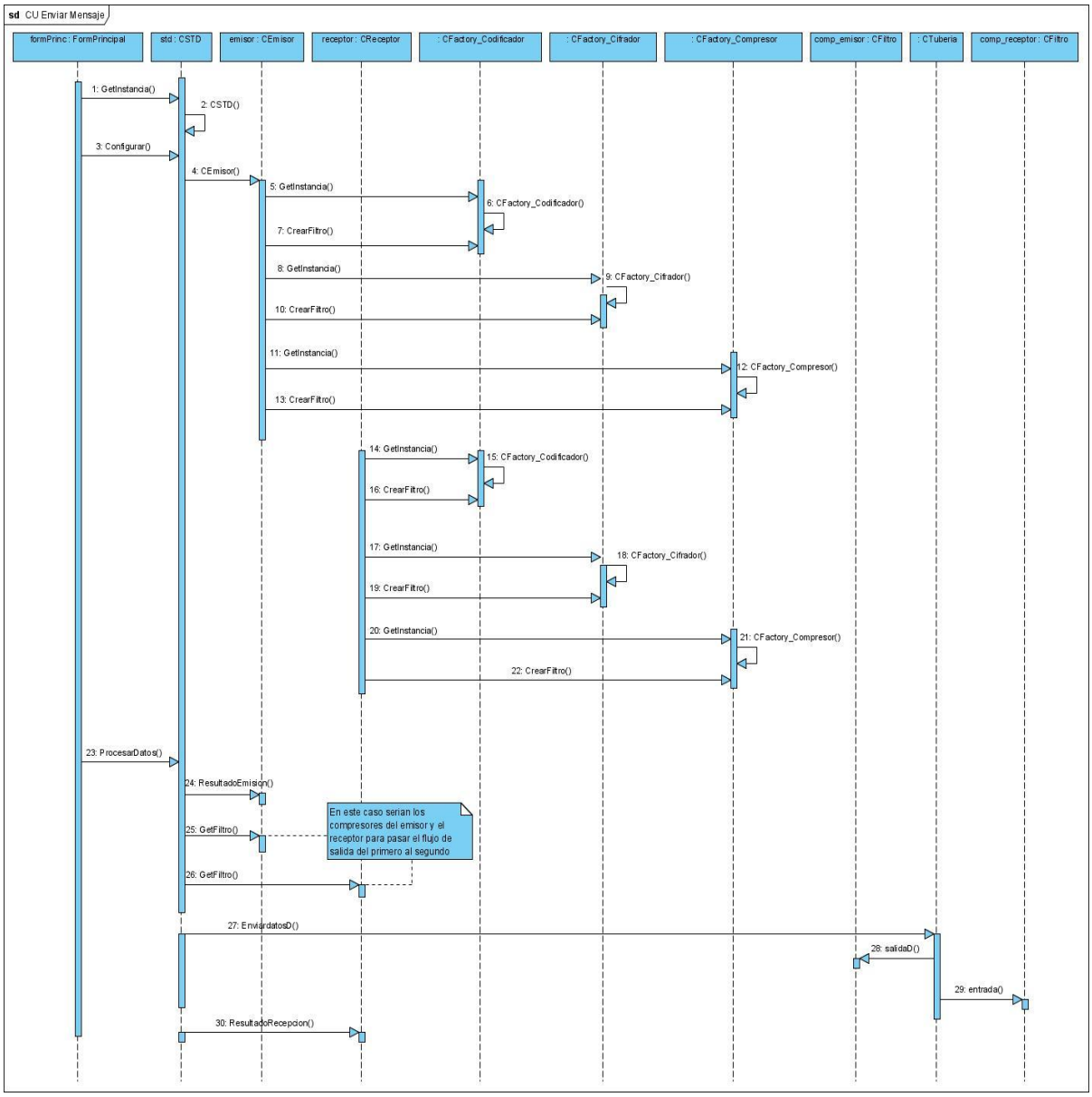
Anexo 21 Diagrama de clases del diseño CU Descifrar



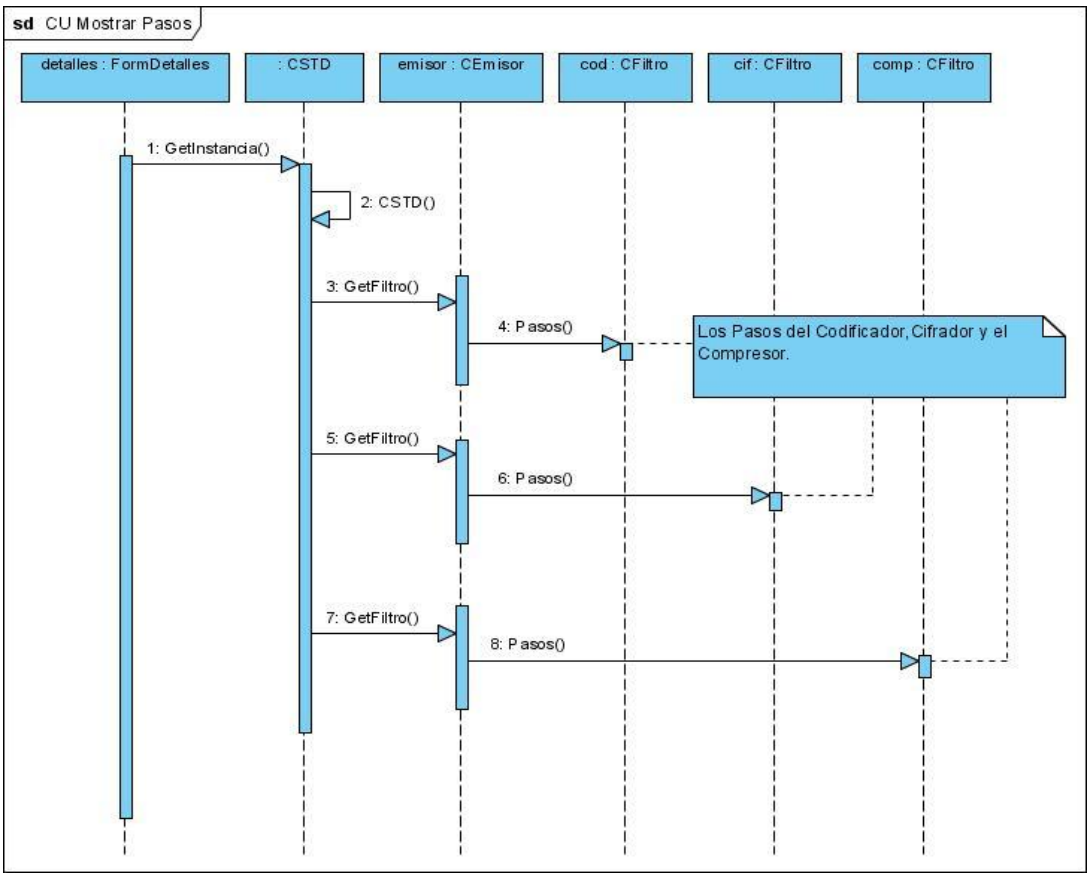
Anexo 22 Diagrama de clases del diseño CU Comprimir



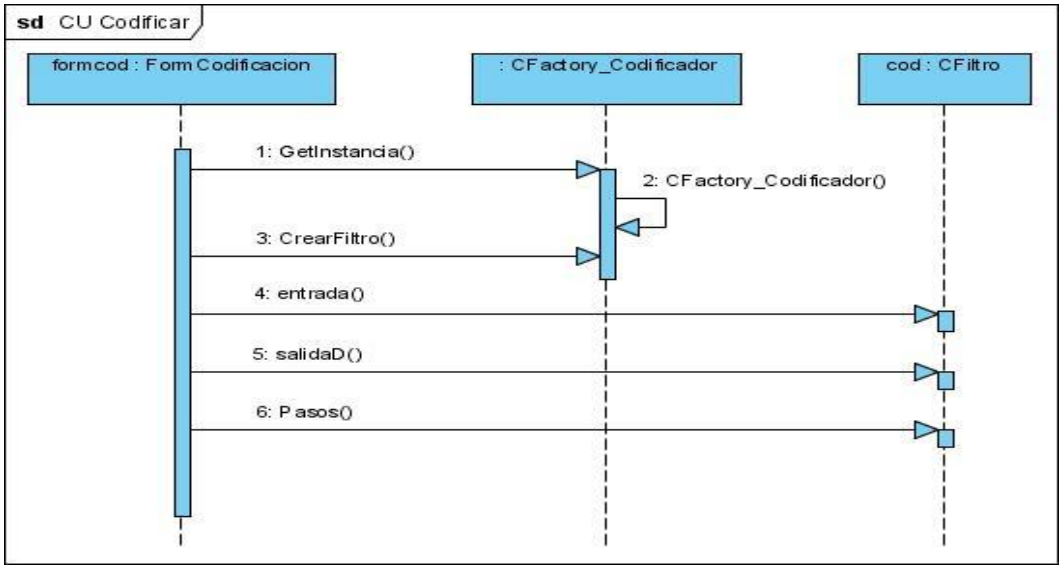
Anexo 23 Diagrama de clases del diseño CU Descomprimir



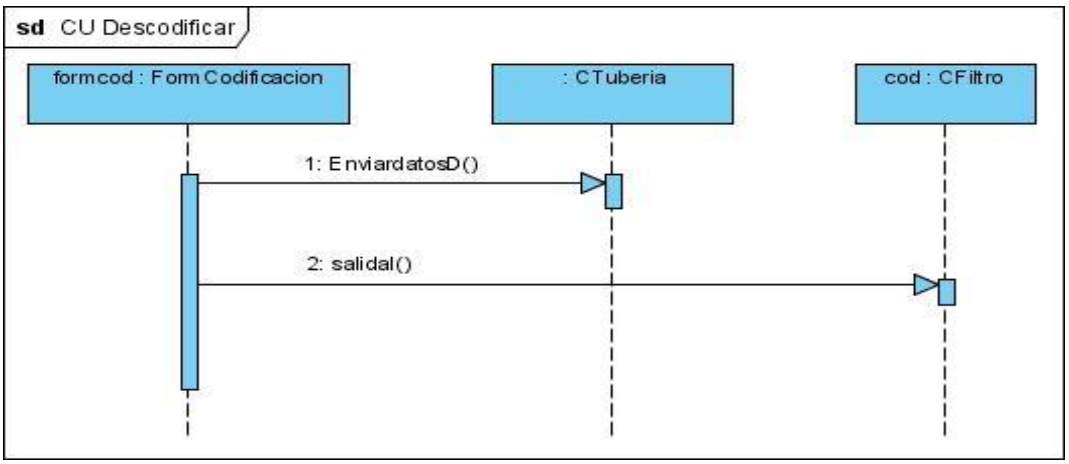
Anexo 24 Diagrama de iteración CU Procesar Mensaje



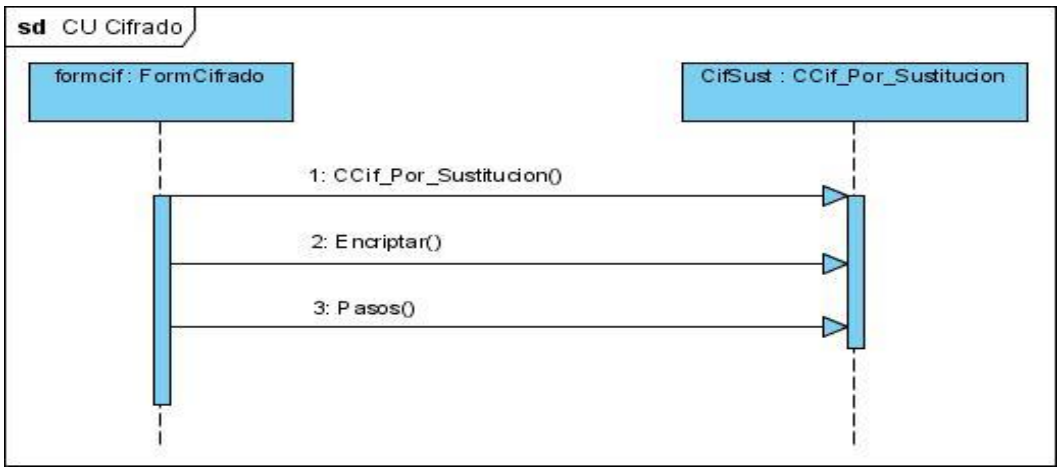
Anexo 25 Diagrama de iteración CU Mostrar Pasos



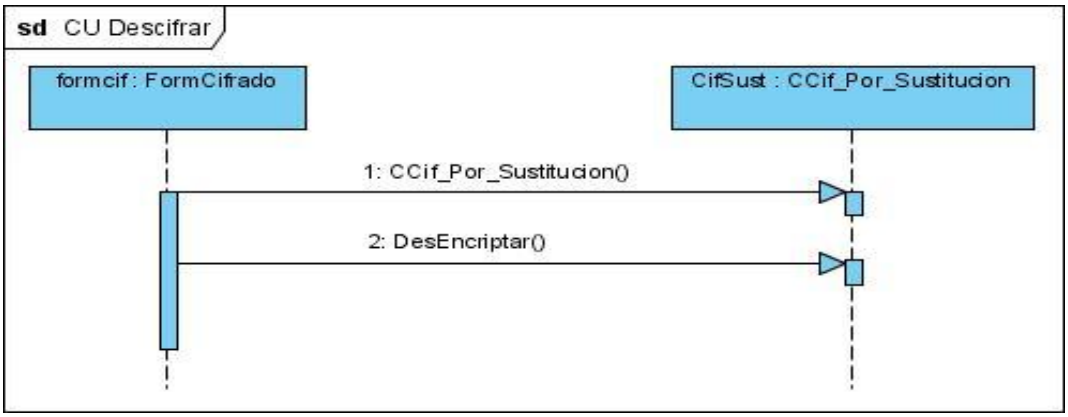
Anexo 26 Diagrama de iteración CU Codificar



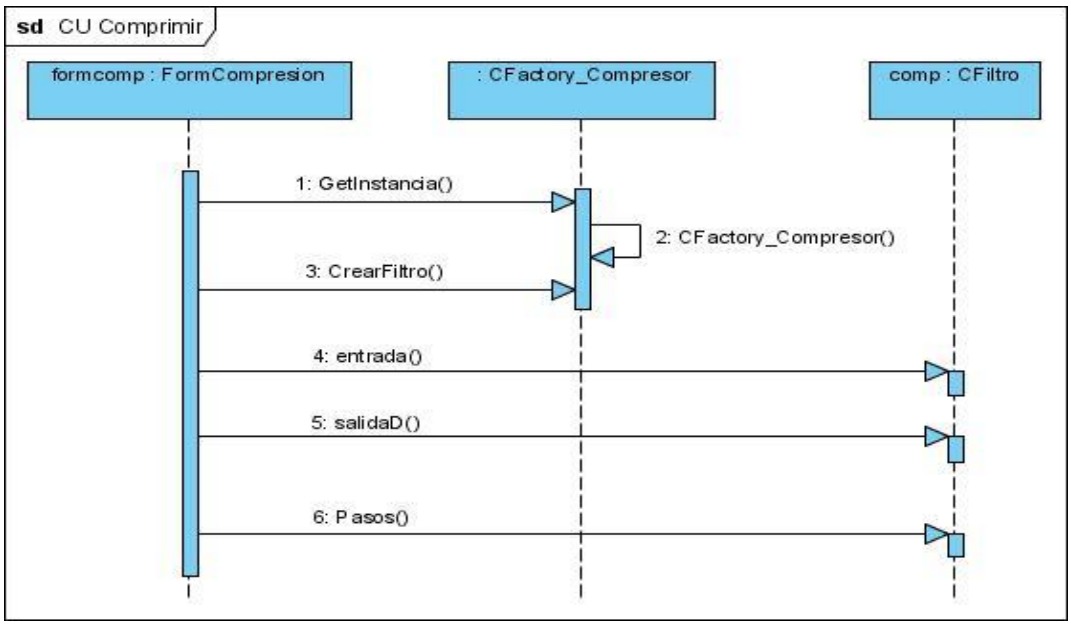
Anexo 27 Diagrama de iteración CU Descodificar



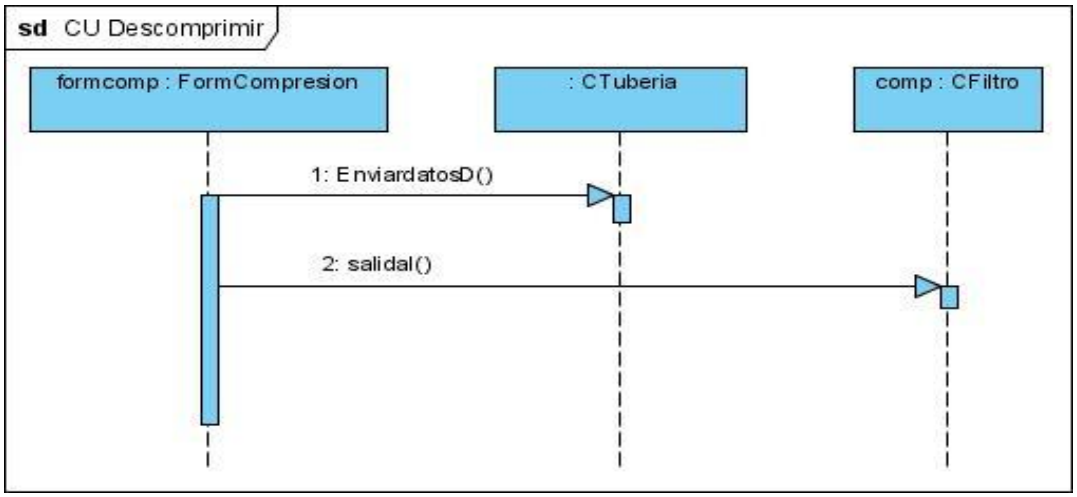
Anexo 28 Diagrama de iteración CU Cifrar



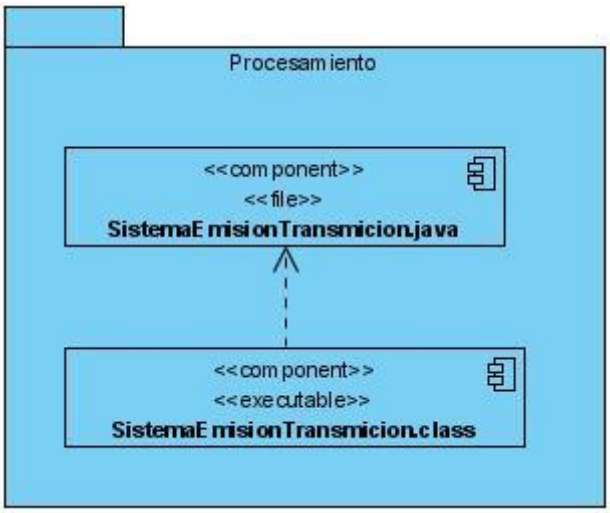
Anexo 29 Diagrama de iteración CU Descifrar



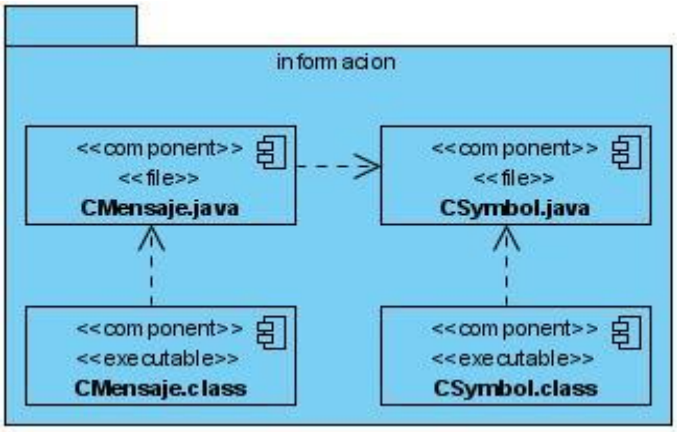
Anexo 30 Diagrama de iteración CU Comprimir



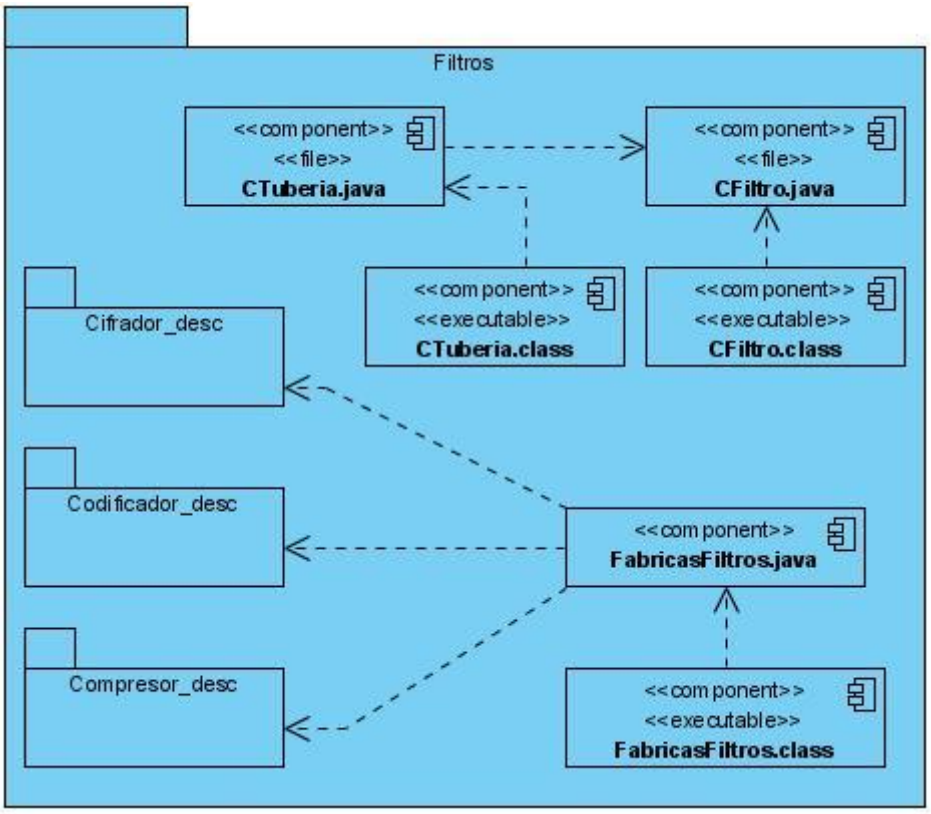
Anexo 31 Diagrama de iteración CU Descomprimir



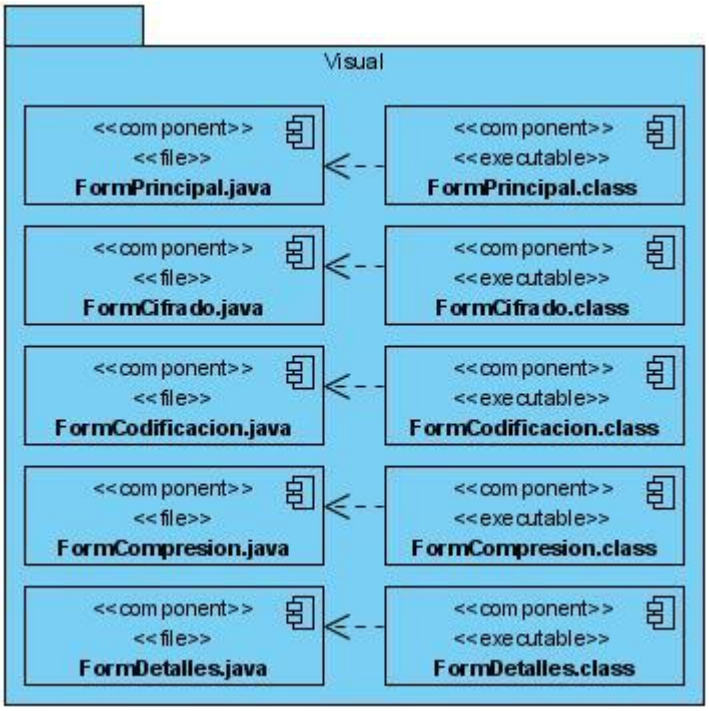
Anexo 32: Paquete de implementación Procesamiento



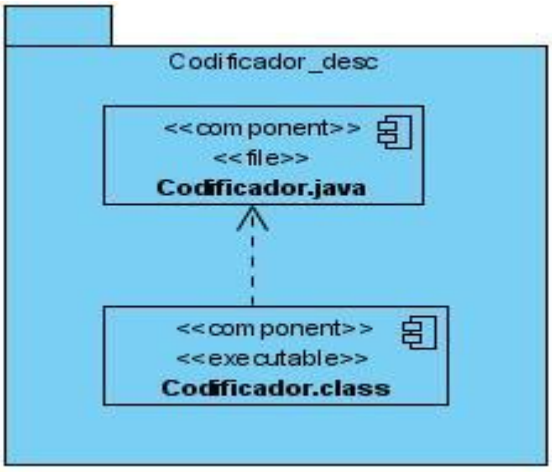
Anexo 33: Paquete de implementación información



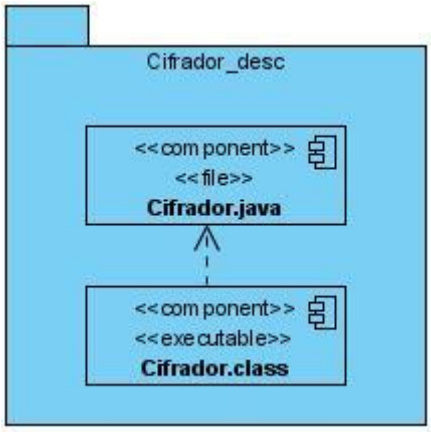
Anexo 34: Paquete de implementación Filtros



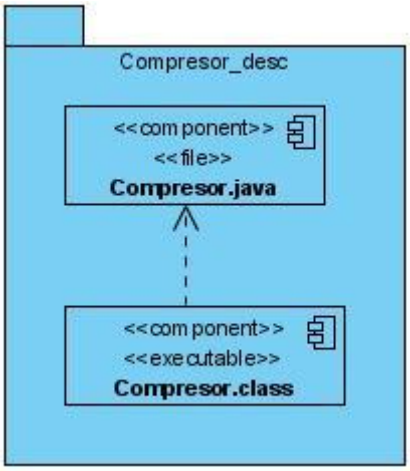
Anexo 35: Paquete de implementación Visual



Anexo 36: Paquete de implementación Codificador_desc



Anexo 37: Paquete de implementación Cifrador_desc



Anexo 38: Paquete de implementación Compresor_desc

Universidad de las Ciencias Informática, Junio 2008
"Año 50 de la Revolución"

A quien pueda interesar:

Mediante este documento hago saber que el software "Simulador PSTD" destinado para usarse como apoyo a la asignatura de Teleinformática I. Desarrollado por los estudiantes de 5to año de la Facultad # 3, Arnolis Rodríguez del Valle y Carlos Matos Carbonell cuenta con las funcionalidades solicitadas y cumple con los requisitos funcionales acordados.

Permite resolver ejercicios haciendo uso de los principales algoritmos de codificación, tanto eficiente como redundante; así como de cifrado y compresión impartidos en las clases de Teleinformática I.

Cuenta con una serie de módulos mediante los cuales se pueden hacer ejercicios de codificación, cifrado y compresión de manera independiente. Muestra los pasos y resultados obtenidos al realizar el proceso y algoritmo seleccionado, siendo esto útil para los profesores, quienes se pueden apoyar en él para impartir las clases y desarrollar algunos ejemplos. Para los estudiantes también juega un papel relevante a la hora de comprobar los ejercicios resueltos en clases o estudio individual. También realiza el proceso completo de transmisión, dando la posibilidad de seleccionar en cada una de las distintas fases los algoritmos a utilizar.

El sistema es multiplataforma, permitiendo así que sea usado por todas las facultades de la Uci, incluyendo la 10, quien tiene sus actividades docentes montada sobre Linux.

Ing. Yohandri Gil Ril
Asesor Técnico Docente del
Departamento de Sistemas Digitales

Anexo 39: Carta de evaluación

GLOSARIO

Debugger: Operación que se realiza en un IDE de desarrollo que permite corregir el código línea por línea viendo los resultados que van obteniendo las variables que intervienen en una parte del código.

Máquina Virtual de Java(JVM): La Máquina virtual Java (en inglés Java Virtual Machine) es un programa nativo, es decir, ejecutable en una plataforma específica, capaz de interpretar y ejecutar instrucciones expresadas en un código binario especial (el Java bytecode), el cual es generado por el compilador del lenguaje Java.

Plug-ins: Un plugin (o plug-in en inglés "enchufar", también conocido como addin, add-in, addon o add-on) es una aplicación informática que interactúa con otra aplicación para aportarle una función o utilidad específica, generalmente muy específica, como por ejemplo servir como driver (controlador) en una aplicación, para hacer así funcionar un dispositivo en otro programa. Esta aplicación adicional es ejecutada por la aplicación principal.

Memoria RAM: Memoria de acceso directo (Random Access Memory). Normalmente se usa este nombre para referirse a memorias en las que se puede leer y también escribir (RWM). En los últimos PC es habitual que se use Fast Page Ram (386 y anteriores), EDO Ram (486 y Pentium) y SDRAM (últimos Pentium, Pentium MMX y superiores).

Microprocesador: El "cerebro" del ordenador. Su velocidad de trabajo se mide en Megahertzios (MHz) y su capacidad de proceso por el número de bits que es capaz de manejar a la vez.

Protocolo: Normas a seguir en una cierta comunicación: formato de los datos que debe enviar el emisor, como debe ser cada una de las respuestas del receptor.

Multitarea: Es cuando un ordenador es capaz de realizar más de una tarea a la vez. Puede ser en paralelo (si tiene más de un procesador) o concurrente (si sólo tiene uno).

GUI: Interfaz gráfica de usuario (Graphical User Interface).

Encapsular: Es la capacidad de que permite mantener oculta la implementación de una abstracción para los usuarios de la misma. El objetivo fundamental es ocultar la implementación para que ninguna parte de un sistema complejo dependa de como se ha implementado otra parte.

Paradigma de programación: Un paradigma de programación es un modelo conceptual para desarrollar programas y su uso se refuerza por el lenguaje que se escoja para realizar un programa en concreto.

Socket: Socket o puerto, permiten la comunicación entre computadoras son el intermediario para que se pueda acceder o no desde un ordenador a otro.

Byte: Unidad en que se mide la información. Un byte equivale a 8 bit y un bit es un 0 o un 1 en el sistema binario.

Scripts: Tipo de programa que consiste de una serie de instrucciones que serán utilizadas por otra aplicación.

Fichero: Conjunto de información que se almacena para consultarse o utilizarse posteriormente.