Universidad de las Ciencias Informáticas
Facultad VIII

Título: Software de Migración de ADABAS a Oracle

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

Autor(es): Arniel Serrano Hernández
Armando Alejandro Rodriguez Hernández

Tutor: Ing. Rafael Y. Rodríguez Montero

Ciudad de la Habana
"Año 50 de la Revolución"

"La actitud frente a la vida es mostrar con el ejemplo el camino a seguir, el llevar a las masas con el propio ejemplo cualesquiera que sean las dificultades a vencer en el camino, quien pueda mostrar el ejemplo de su trabajo repetido durante días sin esperar de la sociedad otra cosa que el reconocimiento a sus méritos, de constructor de esta nueva sociedad, tiene derecho a exigir a la hora del sacrificio."

Che.

DECLARACIÓN DE AUTORÍA

Declaro que somos los únicos autores de este trabajo y autorizo al proyecto CICPC de la Facultad 8 de la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio.

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

Autor(es):
Arniel Serrano Hernández.
Armando Alejandro Rodriguez Hernández

Tutor:
Ing. Rafael Rodríguez Montero

AGRADECIMIENTOS

A nuestra Revolución.

A todos los profesores que influyeron en nuestra formación como futuros profesionales.

A los igualitos de la facultad 8, Danner y Danier, por aplicar todo su conocimiento en nuestro favor.

A Lenna, por brindarnos su incondicional apoyo en la elaboración de este trabajo.

DEDICATORIA

A mis padres, por el amor que me han dado.
A mi hermano, por ser un ejemplo a seguir en mi vida y enseñarme a luchar incansablemente por un objetivo.
A mi novia, por estar a mi lado en los momentos difíciles y apoyarme.
A nuestro tutor Rafael, por brindar tanto esfuerzo en este trabajo y ayudarnos incondicionalmente.
A mis amigos, en especial a Giordis por enseñarme a confiar en mí.
A mis profesores por prepararme como profesional.
A toda mi familia, por estar a mi lado siempre.
A todos mis primos, Yarlenis, Uliser, a quienes les debo parte de mi vida.
A la revolución, por darme la posibilidad de ser un profesional.

Arniel

A Dios ante todo.
A mi mamá por apoyarme siempre.
A mi abuela querida que amo con todo mi ser.
A mi abuelo por tenerme siempre en sus oraciones.
A mis tíos y tías, que más que eso han sido padres y madres.
A mi padre de crianza por ser todo un ejemplo.
A mi bisabuela por su sonrisa tan linda.
A mi hermano que quiero con todas mis fuerzas.
A todos esos pequeños primos que tengo para los cuales me he esforzado para ser mejor cada día.
A mi familia que es todo para mí.
A Maikel, Charlie, Yendy, Greylan, Adriana, Deya, Henry, Carlos C, Alexis, Carlos E, por demostrar ser verdaderos amigos.
A mis primos de la UCI, y mis amigos de toda la vida.

A. Alejandro

RESUMEN

Cuando se habla de migración de datos se refiere al proceso de traspaso de información (*datos*) entre bases de datos. La migración de los datos consiste en convertir los datos desde un sistema de base de datos a otro. Esta migración conlleva a la creación de tablas o modificación de las existentes, cambios en algunos tipos de datos y trasladar la información sin perderla hacia la otra, perdurando así de un gestor hacia otro. Actualmente la mayoría de SGBS incluyen herramientas de ayuda a la migración más o menos fiables, pero no son capaces de contener dicha funcionalidad para todos los tipos de SGBS. El software de migración de ADABAS a Oracle se encarga de leer la información contenida en los ficheros generados por el Natural y migrar toda esta información para el Oracle. El sistema no solamente migra dicha información, sino también elimina una serie de datos que se declaran como obsoletos para la migración, además realiza una previa normalización de la misma para ganar en manejo de información y optimizar el almacenamiento de los datos. El proceso de ejecución de las inserciones lo hace a través de hilos de ejecución para ganar un mayor rendimiento en el proceso de traspaso de la información. Finalmente se obtiene una base de datos gestionada por Oracle, con estructura diferente a la antigua BD, con todos los datos que con el paso del tiempo se recopilaron y sin pérdida de integridad de los mismos.

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

INTRODUCCIÓN

La migración de los datos consiste en mover, convertir, trasladar datos desde un sistema gestor de base de datos a otro. La misma es un gran desafío porque involucra una multitud de problemas de conversión causados por las considerables incompatibilidades que pueden existir entre las BD y la necesidad de preservar las propiedades de ejecución de las aplicaciones de BD existentes.

Adaptable Database System (ADABAS) es un sistema de gestión propietario que no cumple con los estándares internacionales para el manejo de datos, tal como lo hace el Structured Query Language (SQL), además de restringir al cliente de conocer la estructura de los ficheros de base de datos, elemento que hace más difícil cualquier trabajo con los datos en dicho gestor, por lo que los resultados obtenidos son bajos en correspondencia al nivel de respuesta y acción que requiere dicha empresa.

A menudo se desestima la complejidad de la conversión de base de datos y no se toman en cuenta todas las dificultades envueltas en este proceso. Como resultado, se utilizan herramientas que aseguran la conversión parcial y tienen que tratar con las tareas principales manualmente. Pero debido al largo número de objetos (*tablas, procedimientos, etcétera*.) en base de datos, el manejo manual de cualquier problema inesperado aparece en el transcurso del proyecto y puede llevar a demorar y dramáticamente arriesgar el costo de migración.

En el actual contrato de la Universidad de las Ciencias Informáticas (UCI) con el Cuerpo de Investigaciones Científicas, Penales y Criminalísticas (CICPC) de la República de Venezuela se necesita la implementación de un nuevo sistema para mejorar sus funcionalidades, dicha organización posee como sistema gestor de base de datos Adaptable Database System (ADABAS), software que por cuestiones de antigüedad no posee vías de comunicación comunes con el Oracle, además de ser considerado como un software no relacional, elemento que lo hace incompatible con cualquier otro homólogo.

Para migrar los datos existentes en el Cuerpo de Investigaciones Científicas, Penales y Criminalísticas (CICPC), dada la sensibilidad de la información y capacidad que ocupan los mismos sería humanamente imposible realizar dicha operación de forma manual, por lo que se necesita de un mecanismo automático que realice dicha actividad garantizando integridad, seguridad y rapidez al

mismo tiempo. Además, nunca antes se ha implementado un software de migración de datos con dichas características.

Para esto se propone como problema científico:

¿Cómo realizar una migración de los datos existentes en la BD que actualmente usa el Cuerpo de Investigaciones Científicas, Penales y Criminalísticas en Adaptable Database System para Oracle sin pérdidas de información e integridad de los datos?

El objeto de estudio está orientado hacia la migración de datos de bases de datos incompatibles.

El campo de acción de este trabajo se centra en el software de migración de datos de ADABAS a Oracle.

El objetivo general es desarrollar un software, pasando por todo el ciclo de desarrollo que propone RUP, que sea capaz de realizar una migración de los datos del sistema que se utiliza actualmente en el Cuerpo de Investigaciones Científicas, Penales y Criminalísticas en Adaptable Database System para Oracle en el cual estará el sistema en desarrollo.

Para dar cumplimiento a este objetivo se desarrollarán diferentes tareas:

- Realizar diseño teórico de la investigación.
- Realizar diseño metodológico de la investigación.
- Realizar un estudio de las particularidades del ADABAS.
- Realizar un estudio acerca de las características de una migración de datos.
- Realizar modelo de dominio de la aplicación.
- Identificar y describir los requerimientos del sistema.
- Realizar el modelo de análisis y diseño del sistema.
- Definir e implementar la arquitectura a utilizar en la aplicación a desarrollar.
- Realizar la implementación del sistema.
- Probar la integridad de los datos después de la migración.
- Estructurar el documento de tesis

Como idea a defender se plantea que si se desarrolla un software de migración de Adaptable Database System para Oracle en el Cuerpo de Investigaciones Científicas, Penales y Criminalísticas se asegura que se logre una migración sin pérdida de datos e integridad de los mismos.

El presente trabajo ha sido organizado de la siguiente manera:

Capítulo 1: En este Capítulo se realiza una breve descripción del negocio, se realiza un estudio y análisis de las tecnologías apropiadas para desarrollos de este tipo de software, escogiendo la mejor opción para la construcción del sistema propuesto.

Capítulo 2: Después de haber analizado e identificado cada uno de los requerimientos imprescindibles para el funcionamiento del software, este capítulo se propone mostrar a partir de estos requisitos el Diagrama de Caso de Uso, el cual representa las relaciones de los actores que interactúan con el sistema y el flujo de actividades con las que interactúan.

Capítulo 3: En este capítulo se realiza el diseño de la propuesta de solución, seleccionando una arquitectura y modelándose los artefactos que contribuyen al desarrollo de este Software. Se analiza cómo va a estar distribuido el sistema y cómo quedarán estructurados los módulos del sistema.

Capítulo 4: Desde que el implementador empieza a desarrollar una aplicación, se va encontrando a su paso con posibles errores que de ocurrir pudieran afectar el rendimiento de la aplicación, una forma de ver si la aplicación está libre de estos errores es testeando el código o las interfaces. El tratamiento de errores también forma parte del proceso de producción, la buena ejecución del tratamiento de errores conduce a un software que en la fase de pruebas generará menos resultados insatisfactorios.
Este capítulo, tratará fundamentalmente sobre los tipos de pruebas que es necesario realizar para que una aplicación sea probada ya sea en tiempo de compilación (*cuando se está compilando el programa*) o en tiempo de ejecución (*cuando se está ejecutando la aplicación*).

Capítulo

1

1 CAPÍTULO1: FUNDAMENTACIÓN TEÓRICA

1.1 Introducción

En este Capítulo se realiza una breve descripción del negocio, se realiza un estudio y análisis de las tecnologías apropiadas para desarrollos de este tipo de software, escogiendo la mejor opción para la construcción del software propuesto.

1.2 Sistemas de migración existentes en la actualidad

En la actualidad no existen muchos sistemas de migración de datos, pues la mayoría de los Sistemas Gestores de Base de Datos (SGBS) incluyen herramientas de ayuda a la migración, elemento que hace esta tarea un poco más accesible. No obstante el proceso de migración de datos es lo suficientemente delicado como para realizarlo en un entorno de pruebas, contemplando toda la casuística posible en cuanto a tipos de datos a manejar, tablas involucradas y sus relaciones, etcétera. Y así sólo en el momento en el que estemos seguros de que la migración se ha realizado con éxito, sin problemas de interpretación de datos ni pérdida de ellos, se puede preceder a realizarse en un entorno de producción.

1.3 Problemas de recogida de datos actual

Dentro de algunos de los sistemas de migración de datos que se han creado se encuentran los creados para migrar BD privadas a MySQL: Inspirer SQLWays, Embarcadero DT/Studio y Microsoft DTS, no sucediendo así para sistemas antiguos como el ADABAS, el cual no posee hasta el momento alguna herramienta con tal funcionalidad.

La migración de datos puede consumir gran parte del tiempo del personal provocando costosas pérdidas de datos e interrupciones. Es por eso que un software de migración tiene que cumplir con una serie de características principales tales como:

- Excelente protección de datos, seguridad de las actualizaciones de datos durante la migración.
- Consistencia de los datos en el proceso de traslado de la información.

- Mantener la estructura de la base de datos.
- Eficiencia del procesamiento de datos.
- Alta compatibilidad del software que se emplea, así como de los gestores de bases de datos que se migran.

La necesidad de migrar grandes cantidades de datos desde la actual BD de CICPC en Venezuela hacia la nueva BD creada por el proyecto es en sí el real motivo de este producto a desarrollar.

1.4 Sistemas Gestores de Base de Datos

Un Sistema Gestor de Base de Datos (SGBD) es un tipo de software muy específico o conjuntos de ellos que permite manejar de manera clara, sencilla y ordenada altos volúmenes de datos. Por su rapidez, efectividad y buen manejo de grandes flujos de información en poco tiempo, se han colocado entre las primeras necesidades a la hora de optimizar servicios y productos. Por la gran demanda de soluciones informáticas han surgido muchos gestores de base de datos, siendo estos programas quienes permitan manipular grandes volúmenes de datos, poseer un lenguaje de consulta para realizar sus tareas y administrar todos los procesos que se ejecuten en la BD. (ERROR500 2004)

Existe una amplia gama de SGBD con características propias, no obstante los principales componentes de cualquier gestor de la base de datos son los siguientes:

- Control de autorización: Este módulo comprueba que el usuario tiene los permisos necesarios para llevar a cabo la operación que solicita.
- Procesador de comandos: Una vez que el sistema ha comprobado los permisos del usuario, se pasa el control al procesador de comandos.
- Control de la integridad: Cuando una operación cambia los datos de la base de datos, este módulo debe comprobar que la operación a realizar satisface todas las restricciones de integridad necesarias.
- Optimizador de consultas: Este módulo determina la estrategia óptima para la ejecución de las consultas.
- Gestor de transacciones: Este módulo realiza el procesamiento de las transacciones.
- Planificador (*Scheduler*): Este módulo es el responsable de asegurar que las operaciones que se realizan concurrentemente sobre la base de datos tienen lugar sin conflictos.
- Gestor de recuperación: Este módulo garantiza que la base de datos permanece en un estado consistente en caso que se produzca un fallo.
- Gestor de buffer: Este módulo es el responsable de transferir los datos entre memoria principal y los dispositivos de almacenamiento secundario. A este módulo también se le denomina gestor de datos.

Los principales objetivos de un Gestor de Base de Datos son ante todo evitar la redundancia, eliminando así la maleabilidad, y mejorar así los elementos de seguridad de los datos e intimidad. (MARQUÉS, A. 2001)

Además los sistemas de Gestión de Base de Datos presentan múltiples características las cuales cumplen con distintos objetivos:

- Abstracción de la información: Ahorran a los usuarios, detalles acerca del almacenamiento físico de los datos. No influye si una base de datos ocupa uno o cientos de archivos, este hecho se hace transparente al usuario. Así, se definen varios Niveles de Abstracción.
- Independencia: La independencia de los datos consiste en la capacidad de modificar el esquema (*físico o lógico*) de una base de datos sin tener que realizar cambios en las aplicaciones que se sirven de ella.
- Redundancia mínima: Un buen diseño de una base de datos logrará evitar la aparición de información repetida o redundante. De entrada, lo ideal es lograr una redundancia nula; no obstante, en algunos casos la complejidad de los cálculos hace necesaria la aparición de redundancias.
- Consistencia: En aquellos casos en los que no se ha logrado esta redundancia nula, será necesario vigilar que aquella información que aparece repetida se actualice de forma coherente, es decir, que todos los datos repetidos se actualicen de forma simultánea.
- Seguridad: La información almacenada en una base de datos puede llegar a tener un gran valor, por lo que disponen de un complejo sistema de permisos a usuarios y grupos de usuarios, que permiten otorgar diversas categorías de permisos.
- Integridad: Se trata de adoptar las medidas necesarias para garantizar la validez de los datos almacenados. Es decir, se trata de proteger los datos ante fallos de hardware, datos introducidos por usuarios descuidados, o cualquier otra circunstancia capaz de corromper la información almacenada.
- Respaldo y recuperación: Los SGBD deben proporcionar una forma eficiente de realizar copias de seguridad de la información almacenada en ellos, y de restaurar a partir de estas copias los datos que se hayan podido perder.
- Control de la concurrencia: En la mayoría de entornos (*excepto quizás el doméstico*), lo más habitual es que sean muchas las personas que acceden a una base de datos, bien para recuperar información o almacenarla. Y es también frecuente que dichos accesos se realicen de forma simultánea. Así pues, un SGBD debe controlar este acceso concurrente a la información, que podría derivar en inconsistencias.
- Tiempo de respuesta: Lógicamente, es deseable minimizar el tiempo que el SGBD tarda en dar la información solicitada y en almacenar los cambios realizados.

Existen SGBD muy potentes tanto en la clasificación de software propietario como software libre. Como propietarios podemos encontrar Oracle y Microsoft SQL Server que lideran el mercado por sus altas prestaciones dado muchos años de experiencia mientras que como opción libre se encuentra, entre otros, MySQL, gestor muy utilizado en la web por su simplicidad de uso, y PostgreSQL que si bien no soporta alguno de los aspectos más avanzados si representa una opción muy confiable y comprometedora. (SILBERSCHATZ, A)

1.4.1 Oracle

Oracle es un sistema de gestión de base de datos relacional (*RDBMS por el acrónimo en inglés Relational Data Base Management System*). Se considera uno de los sistemas de bases de datos más completos. Es un sistema gestor de base de datos robusto, tiene muchas características que nos garantizan la seguridad e integridad de los datos; que las transacciones se ejecuten de forma correcta, sin causar inconsistencias; ayuda a administrar y almacenar grandes volúmenes de datos; estabilidad, escalabilidad y es multiplataforma.

Entre sus características podemos encontrar:

- Versatilidad

  Por su cualidad de multiplataforma se pueden crear códigos en Java o en C++ utilizando dicha base de datos tanto en Windows como en Linux. Exporta los datos y los migra desde una plataforma a la otra sin causar problemas de integridad o pérdida de datos, migra con su propio software tanto de SQL Server como de MySQL funcionando sobre cualquier plataforma libre, por lo que es muy útil para todos los programadores.
- Potencia

  Ofrece un rendimiento mucho mayor que cualquier otra plataforma de Base de Datos. Permite al administrador asignar sus propias zonas de memoria a los datos y cualidades, se controla en todo momento; tanto el crecimiento como el rendimiento de los distintos esquemas que componen una BD sobre Oracle, aunque esto suponga un problema, ya que los administradores deben estar pendientes de su configuración para no sufrir fallos debido a algún problema de almacenamiento.
- Seguridad

  La seguridad de Oracle como sistema gestor de base de datos es alta, pues al no tener 100 porciento de integración con Windows lo hace invulnerable a los defectos que posee el sistema operativo, además de poseer un sistema de seguridad muy avanzado.
- Complejidad

  Complejo para los administradores que no estén familiarizados con las bases de datos, su alto rendimiento es directamente proporcional a su nivel de complejidad.

¿Qué tipo de almacenamiento ofrece? Por una parte se puede asignar una porción de memoria que alojará el contenido del esquema (*datos*), y otra porción que almacenará las propiedades (*enlaces, claves, etcétera*). Se puede almacenar en segmentos de memoria distintos para evitar fallos, incorrecciones o conflictos con otros esquemas. En cuanto a la velocidad de ejecución, Oracle se muestra variable dependiendo del software y plataforma que lo ejecute. Además, al poseer su propio control de acceso a zonas de memoria, gestiona los elementos de forma más organizada, impidiendo que se entremezclen conceptos. (CORPORATION, Oracle)

Otro punto importante es la ayuda disponible en la página oficial de Oracle, en ella se puede encontrar varios manuales online con solución a todas las dudas que te puedan aparecer. Además de los foros de discusión en los cuales se postean todo tipo de preguntas y respuestas con relación a cualquier proceso.
Como todo software propietario su licencia posee altos valores en el mercado, por lo que se convierte en un poderoso gestor de muy alto precio en el mercado.

1.4.2 Adaptable Database System (ADABAS)

Es una base de datos jerárquica de alto rendimiento creada por la empresa alemana Software AG, en el año 1969.
ADABAS es una Base de datos de lista invertida. Ha sido descrita como “no relacional” aunque “casi relacional” en sus características. He aquí algunas diferencias:

- Archivos, no tablas, como la mayor unidad organizacional.
- Registros, no filas, como la unidad contenedora dentro de la unidad organizacional.
- Campos, no columnas, como componentes de una unidad de contenido.
- No tiene un motor de SQL embebido, por lo que un mecanismo externo de consulta debe ser provisto.
- La lectura sucia es el modo estándar de operación.
- Soporta tablas embebidas, por ejemplo en el caso de archivos anidados.

Se ha probado que es muy exitoso en proveer acceso eficiente a los datos y en mantener la integridad en la base de datos. Actualmente, ADABAS posee una gestión de datos ilimitada en cuanto a volúmenes de datos y sincronización de datos estructurados y no estructurados o para ambientes de alto procesamiento de transacciones analíticas en línea. Ya no tiene que preocuparse por el tiempo de inactividad que se produce cuando se alcanzan los límites de almacenamiento y no hay necesidad de realizar esfuerzos de integración costosos y complejos.

La base de datos ADABAS asegura un acceso muy rápido, que no consigue DB2 ni Oracle en grandes volúmenes de datos, ya que la estructura de ADABAS de archivos y no tablas lo favorece ante los

demás gestores en cuanto a velocidad de procesamiento. En cuanto la interacción, tanto para administración como para realizar cualquier tipo de operación sobre dicho gestor no existen ventanas ni ningún tipo de eventos, lo que dificulta mucho más su manejo, además de la poca información existente tanto en la Web como en la documentación con relación al mismo. Fue diseñado para funcionar con mínimos gastos de personal. (WIKIPEDIA F 2008)

1.5 Metodologías

Uno de los principales problemas en la actualidad en el desarrollo de software es seleccionar la metodología más adecuada que posibilite obtener los resultados óptimos que se desean; o sea, cómo trabajar eficientemente evitando las catástrofes que conllevan al fracaso de un gran porcentaje de proyectos a nivel mundial. Una metodología tiene como principal objetivo aumentar la calidad del software que se produce en todas y cada una de sus fases de desarrollo. Se analizará seguidamente tres de las más conocidas, y sus características.

1.5.1 Lenguaje Unificado de Modelado (UML)

El Lenguaje Unificado de Modelado (UML) como soporte de lenguaje orientado a objeto para el modelado de aplicaciones.

En todas las disciplinas de la ingeniería se hace evidente la importancia de los modelos ya que describen el aspecto y la conducta de “algo”. Ese “algo”, puede existir, estar en un estado de desarrollo o estar, todavía, en un estado de planeación. UML es, probablemente, una de las innovaciones conceptuales en el mundo tecnológico del desarrollo de software que más expectativa ha generado a lo largo de muchos años, comparable con la aparición e implantación de los lenguajes COBOL10, Basic, Pascal, C++, y actualmente con los más recientes Java, XML, C#. (HERNANDEZ ORALLO 2001)

UML es ya un estándar de la industria del software, pero no solo de la industria sino, que en general, de cualquier industria que requiera la construcción de modelos como condición previa para el diseño y posteriormente para la construcción de prototipos.

UML ha nacido como un lenguaje, pero es mucho más que un lenguaje de programación. En realidad se ha diseñado y construido un lenguaje que ha nacido con una madurez sólida si se le compara, incluso con los últimos desarrollos de HTML, C#, Java, Ajax, Xml, los lenguajes por excelencia del mundo de la Internet. (LARMAN 1999)

UML ayuda a los usuarios a entender la realidad desde un punto de vista de la tecnología y la posibilidad de que reflexione antes de invertir y gastar grandes cantidades de dinero en proyectos que

no estén seguros en su desarrollo, reduciendo el costo y el tiempo empleado en la construcción de los módulos que construirán el software. (RUMBAUGH 2000)

El Lenguaje Unificado de Modelado (UML) ha ganado en utilización actualmente, por ser la mezcla eficiente de una gran cantidad de estándares internacionales. Su base esta soportada por tres metodologías procedentes de la unión de tres grandes creadores, James. Rumbaugh, Grady. Boosh e Ivar. Jacobson. Logrando así un lenguaje de excelencia para modelar, que es el procesamiento que realizan los ingenieros para el diseño de software previo a su construcción.

Posee una gran cantidad de propiedades que han sido las que, realmente, ha contribuido a hacer de UML el estándar de la industria en la actualidad.

Algunas de las propiedades de UML como lenguaje de modelado son:

- Es un lenguaje distribuido y adecuado a las necesidades de conectividades actuales y futuras. Ampliamente utilizado por la industria del software.
- Reemplaza a decenas de notaciones empleadas por otros lenguajes.
- Modela estructuras complejas.
- Las estructuras más importantes que soporta tienen su fundamento en la tecnología orientada a objeto, tales como objetos, clases, componentes y nodos.
- Comportamiento del sistema: casos de usos, diagramas de secuencia, de colaboración, que sirve para evaluar el estado de las máquinas.

El modelar sirve, no solamente para los grandes sistemas, sino en aplicaciones de pequeño tamaño se obtienen beneficios del modelado, sin embargo es un hecho que entre más grande y complejo es el sistema, más importante es el papel que juega el modelado por una simple razón: "El hombre hace modelos de sistemas complejos porque no puede entenderlos en su totalidad".(RUMBAUGH 2000)

1.5.2 Proceso Unificado del Rational (RUP)

El Proceso Unificado de Rational o RUP es un proceso para la Ingeniería de Software. Proporciona un acercamiento disciplinado a asignar tareas y responsabilidades dentro de una organización de desarrollo. Su meta es asegurar la producción de software de calidad superior que satisface las necesidades de sus usuarios finales dentro de un tiempo y presupuesto predecibles.
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. El ciclo de vida que se desarrolla por cada iteración, es llevada bajo dos disciplinas.

El proceso unificado de desarrollo, RUP, es el resultado de la evolución e integración de diferentes metodologías de desarrollo de software. Permite sacar el máximo provecho de los conceptos asociados a la orientación a objetos y al modelado visual. Cuenta con las mejores prácticas del modelo de desarrollo de un software en particular. (PRESSMAN 2005)

- Desarrollo de software de forma iterativa.
- Manejo de requerimientos.
- Utiliza arquitectura basada en componentes.
- Modela el software de forma visual, usando UML.
- Verifica la calidad del software.
- Controla los cambios.
- Dirige las tareas de cada desarrollador por separado y del equipo como un todo.
- Especifica los artefactos que deben desarrollarse en cada fase de desarrollo del software.

Características de RUP:

- Creado por Jacobson, Rumbaugh y Booch.
- Unifica los mejores elementos de metodologías anteriores.
- Preparado para desarrollar grandes y complejos proyectos.
- Orientado a Objetos.
- Utiliza el UML como lenguaje de modelado para preparar todos los esquemas de un sistema de software. Es una parte esencial del Proceso Unificado de desarrollo de software y fueron desarrollados paralelamente por las mismas personas, haciendo que su integración sea un éxito.

Además de las características anteriores RUP posee tres características que lo convierten en único:

- Guiado por Casos de Uso.
- Iterativo e Incremental.
- Centrado en la Arquitectura.

RUP divide el proceso de desarrollo en ciclos, teniendo un producto al final de cada ciclo, cada ciclo se divide en fases que finalizan con un hito donde se debe tomar una decisión importante:

- Inicio: se hace un plan de fases, se identifican los principales casos de uso y se identifican los riesgos.
- Elaboración: se hace un plan de proyecto, se completan los casos de uso y se eliminan los riesgos.

- Construcción: se concentra en la elaboración de un producto totalmente operativo y eficiente y el manual de usuario.
- Transición: se implementa el producto en el cliente y se entrena a los usuarios. Como consecuencia de esto suelen surgir nuevos requisitos a ser analizados.

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 y es la más adaptable para proyectos de largo plazo. (JACOBSON, I. 2000)

1.5.3 Programación Extrema (XP)

La Programación Extrema (XP), mejor conocida por su nombre en inglés Extreme Programming, es una de las llamadas Metodologías Ágiles de desarrollo de software más exitosas de los tiempos recientes.

La programación extrema o eXtreme Programming (XP) es un enfoque de la ingeniería de software formulado por Kent Beck, autor del primer libro sobre la materia, Extreme Programming Explained: Embrace Change (1999). Es la más destacada de los procesos ágiles de desarrollo de software. Al igual que éstos, la programación extrema se diferencia de las metodologías tradicionales principalmente en que pone más énfasis en la adaptabilidad que en la previsibilidad.

Los defensores de XP consideran que los cambios de requisitos sobre la marcha son un aspecto natural, inevitable e incluso deseable del desarrollo de proyectos. Creen que ser capaz de adaptarse a los cambios de requisitos en cualquier punto de la vida del proyecto es una aproximación mejor y más realista que intentar definir todos los requisitos al comienzo del proyecto e invertir esfuerzos después en controlar los cambios en los requisitos. Se puede considerar la programación extrema como la adopción de las mejores metodologías de desarrollo de acuerdo a lo que se pretende llevar a cabo con el proyecto, y aplicarlo de manera dinámica durante el ciclo de vida del software.

Las características fundamentales del método son:

- Desarrollo iterativo e incremental: pequeñas mejoras, unas tras otras.
- Pruebas unitarias continuas, frecuentemente repetidas y automatizadas, incluyendo pruebas de regresión. Se aconseja escribir el código de la prueba antes de la codificación.
- Programación en parejas: se recomienda que las tareas de desarrollo se lleven a cabo por dos personas en un mismo puesto. Se supone que la mayor calidad del código escrito de esta manera (*el código es revisado y discutido mientras se escribe*) es más importante que la posible pérdida de productividad inmediata.

- Frecuente integración del equipo de programación con el cliente o usuario. Se recomienda que un representante del cliente trabaje junto al equipo de desarrollo.
- Corrección de todos los errores antes de añadir nueva funcionalidad. Hacer entregas frecuentes.
- Refactorización del código, es decir, reescribir ciertas partes del código para aumentar su legibilidad y mantenibilidad, pero sin modificar su comportamiento. Las pruebas han de garantizar que en la refactorización no se ha introducido ningún fallo.
- Propiedad del código compartida: en vez de dividir la responsabilidad en el desarrollo de cada módulo en grupos de trabajo distintos, este método promueve el que todo el personal pueda corregir y extender cualquier parte del proyecto. Las frecuentes pruebas de regresión garantizan que los posibles errores serán detectados.
- Simplicidad en el código: es la mejor manera de que las cosas funcionen. Cuando todo funcione se podrá añadir funcionalidad si es necesario. La programación extrema apuesta que es más sencillo hacer algo simple y tener un poco de trabajo extra para cambiarlo si se requiere, que realizar algo complicado y quizás nunca utilizarlo.

La simplicidad y la comunicación son extraordinariamente complementarias. Con más comunicación resulta más fácil identificar qué se debe y qué no se debe hacer. Mientras más simple es el sistema, menos tendrá que comunicar sobre este, lo que lleva a una comunicación más completa, especialmente si se puede reducir el equipo de programadores. (MOLPECEES 2003)

1.5.4 Desarrollo Guiado por Funcionalidad (FDD)

FDD es un proceso diseñado por Peter Coad, Erich Lefebvre y Jeff De Luca y se podría considerar a medio camino entre RUP y XP, aunque al seguir siendo un proceso ligero es más similar a este último. FDD está pensado para proyectos con tiempo de desarrollo relativamente cortos (*menos de un año*). Se basa en un proceso iterativo con iteraciones cortas (*aproximadamente 2 semanas*) que producen un software funcional que el cliente y la dirección de la empresa pueden ver y monitorizar.
Las iteraciones se deciden en base a features (*de ahí el nombre del proceso*) o funcionalidades, que son pequeñas partes del software con significado para el cliente.

Un proyecto que sigue FDD se divide en 5 fases:

- Desarrollo de un modelo general
- Construcción de la lista de funcionalidades
- Plan de releases en base a las funcionalidades a implementar
- Diseñar en base a las funcionalidades

- Implementar en base a las funcionalidades

Las primeras tres fases ocupan gran parte del tiempo en las primeras iteraciones, siendo las dos últimas las que absorben la mayor parte del tiempo según va avanzando el proyecto, limitándose las primeras a un proceso de refinamiento. Las funcionalidades a implementar en un release se dividen entre los distintos subgrupos del equipo, y se procede a implementarlas. (MOLPECEES 2003)

1.6 Herramientas CASE

Hoy en día, se ha hecho habitual que cada vez que se desarrolle un software por pequeño o grande que se sea, se utilizan las herramientas CASE, con el fin de automatizar los aspectos clave de todo el proceso de desarrollo de un sistema, desde el principio hasta el final e incrementando su calidad.

Los medios con los que siempre se ha realizado el intercambio de información de diseño e ideas usando la notación UML han sido populares: pizarras, cuadernos y trozos de papel, etcétera. Pero UML se utiliza mejor a través de una herramienta de modelado, la cual puede ser usada para capturar, guardar, rechazar, integrar automáticamente información, y diseño de documentación. (ROSSI 2004)

Una característica que UML brinda para beneficiar a los modeladores es escoger una herramienta de modelado. Como una buena caja de herramientas, una buena herramienta de modelado ofrece todas las herramientas necesarias para conseguir hacer eficientemente varios trabajos, sin dejarte nunca sin la herramienta correcta. (ROSSI 2004)

Las herramientas de modelado deberían soportar las siguientes funcionalidades:

- Soporte para toda la notación y semántica de UML.
- Soporte para una cantidad considerable de técnicas de modelado y diagramas para complementar UML, incluyendo tarjetas CRC, modelado de datos, diagramas de flujo, y diseño de pantallas de usuario. Posibilidad de reutilizar información obtenida por otras técnicas todavía usadas, como modelado tradicional de procesos.
- Facilitar la captura de información en un repositorio subyacente permitiendo la reutilización entre diagramas.
- Posibilidad de personalizar las propiedades de definición de elementos subyacentes de modelos UML.
- Permitir a varios equipos de analistas trabajar en los mismos datos a la vez.

- Posibilidad de capturar los requisitos, asociarlos con elementos de modelado que los satisfagan y localizar cómo han sido satisfechos los requisitos en cada uno de los pasos del desarrollo.
- Posibilitar la creación de informes y documentación personalizados en tus diseños, y la salida de estos informes en varios formatos, incluyendo HTML para la distribución en la Internet o Intranet local.
- Posibilidad para generar y realizar ingeniería inversa (*por ejemplo C++, Java, etcétera*.) para facilitar el análisis y diseño interactivo, para volver a usar código o librerías de clase existentes, y para documentar el código. (ROSSI 2004)

1.6.1 Rational Rose (Enterprise Edition)

Es la herramienta CASE que comercializan los desarrolladores de UML (Booch, Rumbaugh y Jacobson) y que soporta de forma completa la especificación del UML. Esta herramienta propone la utilización de cuatro tipos de modelos para realizar un diseño del sistema, utilizando una vista estática, otra dinámica de los modelos del sistema, una lógica y otra física; que permite crear y refinar estas vistas creando de esta forma un modelo completo que representa el dominio del problema y el sistema de software.

En el Rational Rose (EE) se añaden nuevas características:

- Característica de control por separado de componentes modelo que permite una administración más granular y el uso de modelos.
- La generación de código Ada, ANSI C ++, C++, CORBA, Java y Visual Basic, con capacidad de sincronización modelo- código configurables.
- Soporte Enterprise Java Beans™ 2.0.
- Capacidad de análisis de calidad de código.
- Modelado UML para trabajar en diseños de base de datos, con capacidad de representar la integración de los datos y los requerimientos de aplicación a través de diseños lógicos y físicos.
- Capacidad de crear definiciones de tipo de documento XML (DTD) para el uso en la aplicación
- Integración con otras herramientas de desarrollo de Rational.
- Capacidad para integrarse con cualquier sistema de control de versiones SCC-compliant, incluyendo a Rational ClearCase.
- Publicación web y generación de informes para optimizar la comunicación dentro del equipo.

Los tipos de modelos son:

- Desarrollo Iterativo
- Generador de Código
- Ingeniería Inversa
- Trabajo en Grupo (CORPORATION, Rational)

1.6.2 ER/Studio

Es una herramienta de modelado para diseñar bases de datos. Ayuda a descubrir, documentar y reutilizar los datos activos. Con un soporte de ida y vuelta de bases de datos, los arquitectos de datos tienen el poder para analizar a donde las fuentes de datos existentes tan bien como el diseño e implementación de bases de datos de alta calidad.

Esta herramienta ofrece las siguientes características: Ambiente de diseño orientado al modelo, soporte del ciclo de vida de bases de datos completas, administración del modelo empresarial, soporte para integración y almacenes de datos o data warehouse, y diseño de base datos con calidad. Además de incorporarse nuevas características como:

- Asistente fácil de usar para la construcción de esquemas XML directamente desde un modelo de datos físico o lógico, permitiendo a los arquitectos de datos incorporar los mismos estándares en el desarrollo SOA que ellos utilizaron previamente para la construcción de las bases de datos
- Utilería para el nombramiento de estándares y un editor para el mapeo de diferentes tipos de datos que automáticamente transforma el modelo de metadatos, permitiendo a los arquitectos de datos ser más productivos para mover modelos entre las capas de los diseños lógicos y físicos
- Utilería para re-arquitectura y conversión del modelo de datos que mejora significativamente la calidad de los metadatos cuando se importan desde plataformas tales como herramientas de modelado, plataformas de BI, ETL, formatos estándar de intercambio de archivos y herramientas de modelado (ER/Studio)

Además la nueva versión 7.0, aporta las siguientes características:

- Actualización del diseño de la base de datos física:
    - Gestión de la seguridad: Gestión de usuarios, permisos y roles en cualquier modelo físico o lógico permitiendo propagar los permisos desde el modelo lógico al físico una vez que se crea éste y sincronizar los usuarios, roles y permisos entre los modelos y una base de datos.
    - Pronóstico de la capacidad: Pronosticar el número de filas y ratios de crecimiento de filas para las tablas y calcular los requisitos de almacenaje futuros.
    - Generación de sentencias ALTER SQL: Generar automáticamente código ALTER cuando se compara dos modelos físicos o se compara un modelo físico con un archivo SQL

- Actualización de la gestión del modelo
    - Herencia de Dominios: ER/Studio 7.0 soporta la herencia de estructuras de dominios. Deriva nuevos dominios de uno ya existente para una plataforma específica o permite construir conjuntos de dominios relacionados.
    - Mapeos definidos por el usuario: ER/Studio 7.0 permite al ingeniero de software definir sus propios mapeos entre un modelo físico y lógico
    - Añadir nuevos modelos**:** pudiendo importar modelos ER/Studio o DT/Studio usando el asistente "Add New Physical Model" , ó bien utilizando técnicas de ingeniería inversa importando modelos ya existentes o a partir de un archivos SQL
    - Actualización del Submodelo**:** Ofrece un control detallado sobre la construcción y mantenimiento de submodelos, permitiendo añadir y borrar objetos relacionados

1.6.3 Visual Paradigm for UML (Enterprise Edition)

Herramienta de modelado diseñada para un gran número de usuarios, incluyendo ingenieros de sistemas, analistas de sistemas, analistas de negocio, arquitectos de sistemas. Además se integra con IDEs (*Eclipse, JBuilder, NetBeans, IntelliJ IDEA, JDeveloper and WebLogic Workshop*) para soportar la fase de implementación de desarrollo de software.

La transición desde el análisis al diseño y después a la implementación es cuidadosamente integrada dentro de la herramienta CASE, de esta manera se reduce significativamente el esfuerzo en todas las etapas del ciclo de vida del desarrollo del software. Incluye además un conjunto de herramientas que soportan Object-Relational Mapping (ORM), como Hibernate, lo cual incluye generación completamente orientada a objetos, listo para usar librerías para obtener y modificar registros de base de datos para una gran variedad de gestores de base de datos, y la sincronización entre los diagramas de clases y los diagramas de entidad-relación (ERD).

Presenta además generación de código e ingeniería inversa del código. Permite generar los EJB y así mismo los descriptores de despliegue para varios servidores de aplicación. Distinto a muchas herramientas de modelado pueden extenderse sus diagramas hechos desde el Visio y de Rational Rose. (CORPORATION, VisualP)

1.6.4 CASE Studio

Modelador de base de datos altamente profesional y adaptable que permite a diseñadores y desarrolladores de base de datos crear y mantener visualmente diagramas de entidad relación (DER), diagramas de flujo de datos (DFD) y generar scripts de SQL para diferentes bases de datos de forma

automática. Provee un soporte completo para más de 20 bases de datos, Por ejemplo. Oracle, BD2, MSSQL, Sybase, MySQL, Firebird, PostgreSQL, etcétera.

Principales características de CASE Studio:

- Generación automática de diagramas entidad-relación de SQL (DDL) Scripts para ingeniería en Reversa.
- Generación a detalle en HTML y RTF documentación de diagramas de Flujo de datos exportables en formato XML en la versión de administrador, modelos en editor.

Su principal característica, es su potente sistema de ingeniería inversa, que permite identificar y estructurar bases de datos ya existentes para poder trabajar con ellas sin problemas. (CORPORATION, CaseS)

1.7 Herramientas de desarrollo

1.7.1 Microsoft Visual Studio

Es un entorno de desarrollo integrado (*IDE, por sus siglas en inglés*) para sistemas Windows. Soporta varios lenguajes de programación tales como Visual C++, Visual C#, Visual J#, ASP.NET y Visual Basic .NET, aunque actualmente se han desarrollado las extensiones necesarias para muchos otros.

Visual Studio permite a los desarrolladores crear aplicaciones, sitios y aplicaciones web, así como servicios web en cualquier entorno que soporte la plataforma .NET. Así se pueden crear aplicaciones que se intercomuniquen entre estaciones de trabajo, páginas web y dispositivos móviles.
Visual Studio 2005 también añade soporte de 64-bit. Aunque el entorno de desarrollo sigue siendo una aplicación de 32 bits Visual C++ 2005 soporta compilación para x86-64 (*AMD64 e Intel 64*) e IA-64 (*Itanium*). El SDK incluye compiladores de 64 bits así como versiones de 64 bits de las librerías.
Se incluye un diseñador de implantación, que permite que el diseño de la aplicación sea validado antes de su implantación. También se incluye un entorno para publicación web y pruebas de carga para comprobar el rendimiento de los programas bajo varias condiciones de carga.

Este entorno de desarrollo posee muchas ventajas ante otros, pues no muestran total compatibilidad con SQL 2005, Visual Studio posee elementos a su favor al eliminar todo tipo de recurrencia, incluye las herramientas de bajo nivel como los punteros dándoles la terminología de “dato no seguro” y su no manipulación, por lo que hace fiable y segura las aplicaciones creadas bajo esta herramienta, además de su poderoso motor de corrección y detección de errores, elemento que para el programador es de gran importancia.

Dentro de las principales características de Visual Studio se puede encontrar que facilita el diseño de formularios Windows a través de una superficie de diseño grafico que permite al programador crear una interfaz de usuario. Para el acceso a distintos tipos de datos cuenta con clases y componentes visuales cuenta con herramientas de depuración de código que puede utilizarse en todos los lenguajes de soportados por Visual Studio. (CORPORATION Microsoft. 2008)

1.7.2 SharpDevelop

Entorno de desarrollo integrado libre para los lenguajes de programación C#, Visual Basic .NET.
Es usado típicamente por aquellos programadores de los citados lenguajes, que no desean o no pueden usar el entorno de desarrollo de Microsoft, el Microsoft Visual Studio. Hay disponible uno para Mono/Gtk#, llamado MonoDevelop, el cual funciona en otros sistemas operativos.
Para el completado automático de código, la aplicación incorpora sus propios parsers. La versiones de esta aplicación pueden importar proyectos de Visual Studio .NET y ya algunas son capaces de editarlos directamente.

Dentro de sus características encontramos que:

- Incorpora un diseñador de Windows forms.
- Depurador incorporado.
- Conversor bidireccional entre C# y Visual Basic .NET, y unidireccional hacia Boo.
- Escrito enteramente en C#.
- Compilación de código directamente dentro del entorno de desarrollo integrado.
- Complementos para C++.
- Pre visualización de documentación XML.
- Gran integración con plantillas a la hora de añadir o crear ficheros, proyectos o compiladores.
- Escritura de código C#, ASP.NET, ADO.NET, XML y HTML.
- Llaves inteligentes en la escritura de código.
- Soporte para plantillas de código.
- Extensible mediante herramientas externas, o complementos. (CORPORATION, SharpD)

Aunque la herramienta posee tantos elementos a su favor no alcanza aún los niveles de facilidad, manejo e integridad del Mismo Visual Studio. NET de Microsoft.

1.7.3 Delphi (Developer Studio)

Entorno de desarrollo de software diseñado para la programación de propósito general con énfasis en la programación visual. En Delphi utiliza como lenguaje de programación una versión moderna de Pascal llamada Object Pascal. En sus diferentes variantes, permite producir archivos ejecutables para Windows, Linux y la plataforma .NET.

Un uso habitual de Delphi es el desarrollo de aplicaciones visuales y de bases de datos cliente-servidor y multicapas. Debido a que es una herramienta de propósito múltiple, se usa también para proyectos de casi cualquier tipo, incluyendo aplicaciones de consola, aplicaciones de web (*por ejemplo servicios web, CGI, ISAPI, NSAPI, módulos para Apache*), servicios COM y DCOM, y servicios del sistema operativo. (MARTEENS, IAN. 2006)

Developer Studio 2006 incluye en el mismo entorno de desarrollo los lenguajes:

- Delphi para Win32
- Delphi para.NET
- C# para.NET
- C++

1.8 Lenguajes Procedulares

1.8.1 Procedural Language/ Structured Query Language (PL/SQL)

Lenguaje de programación embebido en Oracle. El PL/SQL soporta todas las consultas y manipulación de datos que se usan en SQL, pero incluye nuevas características que no poseen sus homólogos:

- El manejo de variables.
- Estructuras modulares.
- Estructuras de control de flujo y toma de decisiones.
- Control de excepciones.

El lenguaje PL/SQL está incorporado en:

- Servidor de la base de datos.
- Herramientas de Oracle (*Forms, Reports*,...).

En un entorno de base de datos los programadores pueden construir bloques PL/SQL para utilizarlos como procedimientos o funciones, o bien pueden escribir estos bloques como parte de scripts SQL*Plus.

Los programas o paquetes de PL/SQL se pueden almacenar en la base de datos como otro objeto, y todos los usuarios que estén autorizados tienen acceso a estos paquetes. Los programas se ejecutan en el servidor para ahorrar recursos a los clientes. (MARTEENS, IAN. 2006)
Para manejar el lenguaje se utiliza el PL/SQL Developer, ambiente integrado para el desarrollo, prueba, depuración de errores y optimización de PL/SQL. Contiene ayuda sensitiva al contexto, descripciones de bases de datos de objetos, sintaxis resaltada, edición y búsqueda de datos, browser gráfico y muchas otras características que le hacen la vida más fácil al usuario. (Developer, PL/SQL)

1.8.2 Natural

En la filosofía del lenguaje, el lenguaje natural es el lenguaje hablado y/o escrito por humanos para propósitos generales de comunicación, para distinguirlo de otros como puedan ser una lengua construida, los lenguajes de programación o los lenguajes usados en el estudio de la lógica formal, especialmente la lógica matemática.

El término lenguaje natural se refiere al estudio de las propiedades computacionales y de otro tipo implicadas en la comprensión, producción y uso de las lenguas naturales.

El Natural se ha diseñado para proporcionar un significativo retorno de su inversión en infraestructura de las tecnologías, a partir de nuevas y potentes herramientas compatibles con SOA, servicios Web, entornos de desarrollo de código abierto y Rich Internet Applications basadas en AJAX. Proporciona notables mejoras, desde soporte para SOA hasta una perfecta integración de las aplicaciones Natural con aplicaciones Java y .NET. El mismo se considera un lenguaje de 4ta generación. (*NaturalAdabas* 2006)

1.9 Lenguajes de Programación

1.9.1 C# (C Sharp)

Lenguaje de programación orientado a objetos desarrollado y estandarizado por Microsoft como parte de su plataforma .NET, que después fue aprobado como un estándar por la ECMA e ISO.
Su sintaxis básica deriva de C/C++ y utiliza el modelo de objetos de la plataforma.NET el cual es similar al de Java aunque incluye mejoras derivadas de otros lenguajes (*más notablemente de Delphi y Java*). C# fue diseñado para combinar el control de lenguajes de bajo nivel como C y la velocidad de programación de lenguajes de alto nivel como Visual Basic.

El lenguaje que incluye mejoras tales como tipos genéricos, métodos anónimos, iteradores, tipos parciales y tipos anulables. (SECO, J. 2001)

Aunque C# forma parte de la plataforma.NET, esta es una interfaz de programación de aplicaciones; mientras que C# es un lenguaje de programación independiente diseñado para generar programas sobre dicha plataforma. Aunque aún no existen, es posible implementar compiladores que no generen programas para dicha plataforma, sino para una plataforma diferente como Win32 o UNIX. Microsoft Visual Studio .NET, es el IDE por excelencia de este lenguaje. (ARCHER, T. 2001)

1.9.2 C++

El C++ es un lenguaje de programación, diseñado a mediados de los años 1980, por Bjarne Stroustrup, como extensión del lenguaje de programación C.
Se puede decir que C++ es un lenguaje que abarca tres paradigmas de la programación: la programación estructurada, la programación genérica y la programación orientada a objetos.
Actualmente existe un estándar, denominado ISO C++, al que se han adherido la mayoría de los fabricantes de compiladores más modernos. Las principales características del C++ son las facilidades que proporciona para la programación orientada a objetos y para el uso de plantillas o programación genérica (*templates*).

Posee una serie de propiedades difíciles de encontrar en otros lenguajes de alto nivel:

- Posibilidad de redefinir los operadores (*sobrecarga de operadores*)
- Identificación de tipos en tiempo de ejecución (RTTI) (STROUSTRUP B.1998)

C++ está considerado por muchos como el lenguaje más potente, debido a que permite trabajar tanto a alto como a bajo nivel, sin embargo es a su vez uno de los que menos automatismos trae (*obliga a hacerlo casi todo manualmente al igual que C*) lo que dificulta mucho su aprendizaje.

1.9.3 Java

Lenguaje de programación orientado a objetos desarrollado por Sun Microsystems a principios de los años 90. El lenguaje en sí mismo toma mucha de su sintaxis de C y C++, pero tiene un modelo de objetos más simple y elimina herramientas de bajo nivel, que suelen inducir a muchos errores, como la manipulación directa de punteros.

La implementación original y de referencia del compilador, la máquina virtual y las librerías de clases de Java fueron desarrolladas por Sun Microsystems en 1995. Desde entonces, Sun ha controlado las especificaciones, el desarrollo y evolución del lenguaje a través del Java Community Process, si bien

otros han desarrollado también implementaciones alternativas de estas tecnologías de Sun, algunas incluso bajo licencias de software libre.

Sun Microsystems liberó la mayor parte de sus tecnologías Java bajo la licencia GNU GPL, de acuerdo con las especificaciones del Java Community Process, de tal forma que prácticamente todo el Java de Sun es ahora software libre (*aunque la biblioteca de clases de Sun que se requiere para ejecutar los programas Java todavía no es software libre*).

El lenguaje Java se creó con cinco objetivos principales:

- Debería usar la metodología de la programación orientada a objetos.
- Debería permitir la ejecución de un mismo programa en múltiples sistemas operativos.
- Debería incluir por defecto soporte para trabajo en red.
- Debería diseñarse para ejecutar código en sistemas remotos de forma segura.
- Debería ser fácil de usar y tomar lo mejor de otros lenguajes orientados a objetos, como C++.

Para conseguir la ejecución de código remoto y el soporte de red, los programadores de Java a veces recurren a extensiones como CORBA (*Common Object Request Broker Architecture*), Internet Communications Engine u OSGi respectivamente.
Java fue creado para abrir una nueva vía en la gestión de software complejo, y es por regla general aceptado que se ha comportado bien en ese aspecto. Sin embargo no puede decirse que Java no tenga grietas, ni que se adapta completamente a todos los estilos de programación, todos los entornos, o todas las necesidades.

En un sentido estricto, Java no es un lenguaje absolutamente orientado a objetos. Por motivos de eficiencia, Java ha relajado en cierta medida el paradigma de orientación a objetos, y así por ejemplo, no todos los valores son objetos.

El código Java puede ser a veces redundante en comparación con otros lenguajes. Esto es en parte debido a las frecuentes declaraciones de tipos y conversiones de tipo manual (*casting*). También se debe a que no se dispone de operadores sobrecargados, y a una sintaxis relativamente simple.
A diferencia de C++, Java no dispone de operadores de sobrecarga definidos por el usuario. Sin embargo esta fue una decisión de diseño que puede verse como una ventaja, ya que esta característica puede hacer los programas difíciles de leer y mantener. (Java)

1.10 Selección de las Tecnologías a utilizar en la propuesta solución

Luego de los resultados arrojados por la comparación de herramientas se ha llegado a la conclusión de que las seleccionadas dentro de su funcionalidad son las mejores para obtener un software de calidad y con buenos resultados.

- Oracle como sistema gestor de BD por sus características excepcionales de sistema gestor, por su estabilidad y escalabilidad, seguridad e integridad de los datos, además de ser el gestor utilizado por el cliente.
- Para el trabajo con los datos, e implementación de sentencias SQL se utilizará PL/SQL Developer 7.14, este permite crear consultas, ejecutar códigos, y una serie de opciones importantes referentes al servidor de base de datos.
- Microsoft Visual Studio como IDE por excelencia de desarrollo del lenguaje C#, por su fiabilidad, por su poderoso motor de corrección y detección de errores y por su buen desempeño en la creación de aplicaciones de escritorio.
- Dentro de las metodologías se llego a la conclusión de que va a utilizar para el desarrollo de la aplicación RUP, ya que las características de dicha metodología son las que más favorecen la creación de un software de buena calidad bajo las circunstancias que se presentan en nuestro ambiente de desarrollo.
- La herramienta de modelado para diseñar bases de datos ER/Studio por su poderoso soporte a bases de datos completas, además de su fácil manejo y de su ambiente de diseño orientado al modelo además permite visualizar toda la estructura de la base de datos, crear relaciones y hacer ingenierías inversas a la base de datos, es popular por su fácil manejo y claridad en el código generado.

Todas estas herramientas cuidadosamente seleccionadas para la obtención del producto, proporcionan seguridad, eficiencia y un buen rendimiento de la aplicación a implementar.

1.11 Conclusiones

En este capítulo se han analizado y estudiado algunas de las tecnologías existentes de gran importancia para la elaboración de software. Además quedó bien reflejada la descripción del negocio, para con esto lograr una mejor automatización de estos procesos. Por último se elaboró un resumen final donde se realiza una selección de las tecnologías que son apropiadas y correctas para la construcción del Software de Migración de ADABAS a Oracle.

Capítulo

2

2 CAPÍTULO2: CARACTERÍSTICAS DEL SISTEMA

2.1 Introducción

Después de haber analizado e identificado cada uno de los requerimientos imprescindibles para el funcionamiento del software, este capítulo se propone mostrar a partir de estos requisitos el Diagrama de Caso de Uso, el cual representa las relaciones de los actores que interactúan con el sistema y el flujo de actividades con las que interactúan.

2.2 Objeto de automatización

Para este trabajo no se partió de ningún proceso que existiera anteriormente, se llevó a cabo con la idea de desarrollar todo el proceso de migración de datos de ADABAS a Oracle con el fin de lograr un traspaso de datos de forma eficiente y sin pérdidas de información. Se automatiza la migración de datos y estructura dada la necesidad de no realizar todo este tedioso proceso de forma manual.

La lectura desde el software de los ficheros de texto extraídos con natural del ADABAS se realiza para eliminar todo tipo de incoherencia y así también generar un nuevo fichero con un formato legible para el software, dicha estructura contiene ya relaciones, tablas, tipos de datos y longitud de los mismos, por lo que dicha operación garantiza una fiable transacción de datos desde las antiguas tablas hacia las nuevas por crear.

Luego de conectarse a la nueva base de datos en Oracle un nuevo módulo de generación será el encargado de realizar las operaciones pertinentes para la clonación en dicha BD, la estructura a generar no tiene la misma forma pues entre estos gestores existe diferentes jerarquías, además se eliminan ciertos elementos inservibles y no funcionales dentro de dicha estructura, y se crean nuevas tablas para una mejor optimización de los datos.

Otro módulo a nivel de la capa de datos se encargará entonces de realizar toda la migración de datos que se encuentra dentro de las tablas antiguas hacia las generadas por los procesos anteriores, el cual garantizará tanto integridad como seguridad de los mismos, por lo que todo el proceso automático culminaría con esta acción, también se le muestra al operador las tablas que no contienen datos en su interior y se le da la posibilidad de buscar su origen y realizar el proceso de llenado de forma manual, por lo que todos estos procesos automatizados garantizaran la culminación de los propósitos a cumplir.

2.3 Modelo de dominio

El Modelo de Dominio (o Modelo Conceptual) es una representación visual de los conceptos u objetos del mundo real significativos para un problema o área de interés. Representa clases conceptuales del dominio del problema. Representa conceptos del mundo real, no de los componentes del software.

Una clase conceptual puede ser una idea o un objeto físico (símbolo, definición y extensión). Es un diagrama de clases en el que se muestran:

- Conceptos u objetos del dominio del problema: clases conceptuales.
- Asociaciones entre las clases conceptuales.
- Atributos de las clases conceptuales.

En el modelo de dominio no se muestra comportamiento. Las clases conceptuales pueden tener atributos pero no métodos.

Cualquiera que sea la solución de los casos de uso que se haya elegido, los conceptos e ideas propias del dominio del problema son las mismas; un mismo modelo de dominio contempla cualquiera de las soluciones analizadas. El modelo de dominio es global, es decir se realiza para todos los casos de uso y no para uno en particular.

Sus características son:

- No posee cronología entre las acciones.
- No se diferencia entre las acciones dentro y fuera del sistema.
- Es un diagrama global, no por casos de uso.
- No es completo: es esquemático y con sus asociaciones resumidas. (BARRIENTOS, E. 2005)

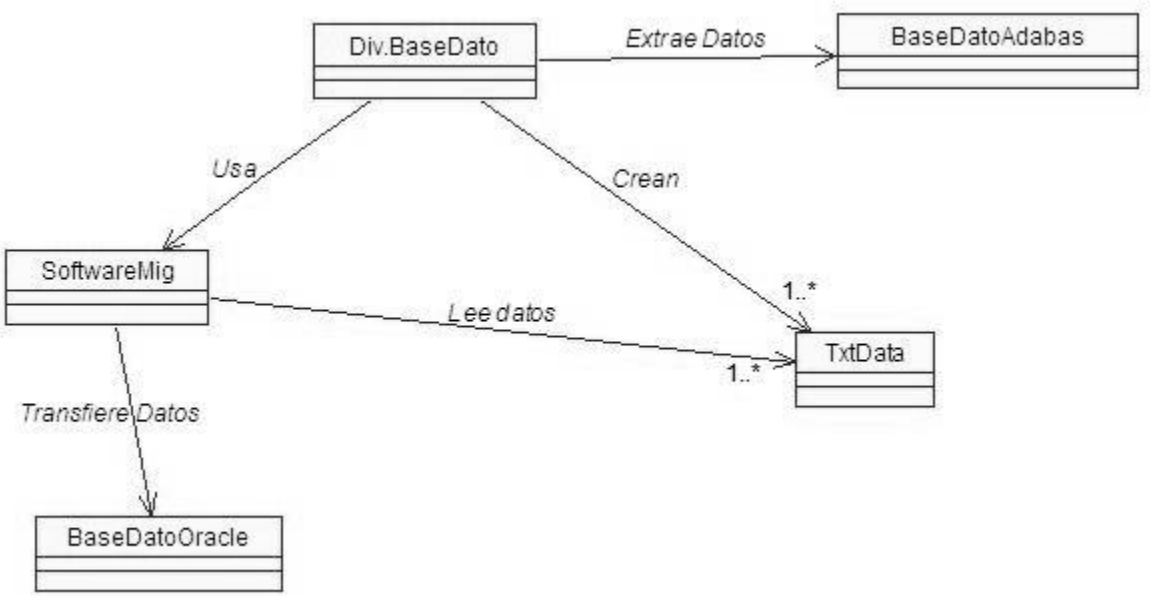

Figura 1. Modelo de Dominio

Glosario de términos empleados en el Modelo de Dominio.

Div.BaseDato: División de Base de Datos, estructura organizacional del CICPC encargada de administrar y controlar servidores de BD.

BaseDatoAdabas: Antigua base de datos que posee como sistema gestor ADABAS, de este se extraen la estructura y los datos.

SoftwareMig: Software de migración, aplicación encargada de automatizar el proceso y garantizar total coherencia e integridad en la operación.

TxtData: Archivos de extensión .txt en los que se encuentra la información referida tanto a estructura como contenido de las tablas a generar y migrar.

BaseDatoOracle: Nueva base de datos que posee como sistema gestor Oracle, hacia este se migran estructura y datos.

2.4 Actores del Sistema

Un actor es una idealización de una persona externa, de un proceso, o de una cosa que interactúa con un sistema, un subsistema, o una clase. Un actor caracteriza las interacciones que los usuarios exteriores pueden tener con el sistema, pueden ser definidos en jerarquías de generalización, en las cuales una descripción abstracta del actor es compartida y aumentada por una o más descripciones

específicas del actor. Un actor puede ser un ser humano, otro sistema informático, o un cierto proceso ejecutable. Se dibuja a un actor como una persona pequeña con trazos lineales y el nombre debajo de él. En la Tabla 1 se hace una descripción de los actores del sistema que se desarrolla.

Tabla 1. Descripción de Actores

Nombre del actor	*Descripción*
Operador	Persona que realiza la migración de los datos, contiene en su conocimiento el lugar donde se encuentran los datos y la estructura de la cual se realizará la migración, es capaz de conectarse con el gestor de base de datos para proceder a hacer la migración. Conoce a fondo el uso del software y lo manipula para obtener los resultados esperados.

2.5 Requerimientos funcionales

- **RF1 -** Cargar Estructura.
  - **RF1.1** Mostrar cuadro de diálogo.
  - **RF1.2** Guardar Información del camino.
  - **RF1.3** Mostrar camino seleccionado.
  - **RF1.4** Cargar el fichero en memoria.
  - **RF1.5** Eliminar los caracteres extraños.
  - **RF1.6** Convertir a caracteres legibles.
  - **RF1.7** Eliminar información irrelevante.
  - **RF1.8** Crear fichero temporal.
  - **RF1.9** Ubicar fichero temporal.

- **RF2 -** Preparar Base de Datos.
  - **RF2.1** Mostrar cuadro de diálogo.
  - **RF2.2** Crear conexión con la base de datos especificada.
  - **RF2.3** Leer el fichero de estructura generado previamente.
  - **RF2.4** Construir script con la DDL.
  - **RF2.5** Realizar una previa normalización.

- **RF2.6** Ejecutar los script generados previamente.

- **RF3 -** Generar script de migración.
  - **RF3.1** Leer el camino donde se encuentran los datos.
  - **RF3.2** Asignar un nombre de fichero de datos para cada tabla.
  - **RF3.3** Hacer búsqueda de los ficheros.
  - **RF3.4** Cargar en memoria la información de los ficheros.
  - **RF3.5** Convertir la información leída a Codificación UTF-8.
  - **RF3.6** Leer la estructura de la tabla.
  - **RF3.7** Comenzar a generar los scripts.
  - **RF3.8** Lanzar un mensaje.

- **RF4 -** Migrar Datos a Oracle Automáticamente.
  - **RF4.1** Comenzar a almacenar hilos de ejecución.
  - **RF4.2** Proceder a ejecutar los scripts.
  - **RF4.3** Habilitar opción de migración manual.

- **RF5 -** Migrar Datos a Oracle Manualmente.
  - **RF5.1** Brindar serie de opciones.
  - **RF5.2** Almacenar camino de los ficheros no encontrados.
  - **RF5.3** Leer la dirección de los ficheros.
  - **RF5.4** Codificar los ficheros a UTF-8.
  - **RF5.5** Crear serie de scripts con la información.
  - **RF5.6** Proceder a ejecutar los scripts.

- **RF6 -** Mostrar Reporte de Migración Manual.
  - **RF6.1** Cargar la información recopilada.
  - **RF6.2** Mostrar serie de información.
  - **RF6.3** Guardar el reporte.

- **RF7 -** Completar Migración.
  - **RF7.1** Mostrar información.
  - **RF7.2** Eliminar ficheros temporales.

- **RF7.3** Modificar el nombre de las tablas.
- **RF7.4** Cerrar conexiones y recursos utilizados.

2.6 Definición de los requerimientos no funcionales

Los requerimientos no funcionales son propiedades o cualidades que el producto debe tener. Debe pensarse en estas propiedades como las características que hacen al producto atractivo, usable, rápido o confiable. Los requerimientos no funcionales forman una parte significativa de la especificación. Son importantes para que clientes y usuarios puedan valorar las características no funcionales del producto, pues si se conoce que el mismo cumple con la toda la funcionalidad requerida, las propiedades no funcionales, como cuán usable, seguro, conveniente y agradable, pueden marcar la diferencia entre un producto bien aceptado y uno con poca aceptación. (RUMBAUGH, J. 2000)

A continuación se detallan los requerimientos no funcionales del sistema.

1- Apariencia o Interfaz externa:

- **RNF1.1** Diseño sencillo de la aplicación, de fácil manejo para cualquier operador.
- **RNF1.2** Construcción hecha a través de botones y pestañas para el acceso rápido a las funcionalidades.

2- Usabilidad:

- **RNF2.1** El sistema podrá ser usado por una persona que tenga conocimientos del manejo de una computadora y del software en específico.

3- Rendimiento:

- **RNF3.1** Tiempos de migración agilizada por ejecución de varios hilos de sentencias.
- **RNF3.2** El tiempo de respuesta está dado por el volumen de datos a migrar.

4- Soporte:

- **RNF4.1** Se requiere de un servidor de base de datos con las siguientes características:

  RNF4.1.1 Soporte para medianos volúmenes de datos y alto procesamiento.

- **RNF4.2** Framework 2.0 o superior.
- **RNF4.3** Los ficheros de migración tienen que cumplir con los estándares de extracción del Natural.

5- Portabilidad:

- **RNF5.1** La herramienta propuesta será usada bajo el sistema operativo Windows 2000 en adelante.
- **RNF5.2** La nueva Base de Datos se encontrará en Oracle 10g release 2 bajo un sistema operativo HP-UX 11.23

6- Seguridad:

- **RNF6.1** Autenticarse en la base de datos y restringir los privilegios del usuario a la parte de la migración.
- **RNF6.2** Garantizar que solamente tenga acceso el personal calificado para la migración.
- **RNF6.3** Garantizar integridad de la información original de los ficheros de la fuente de datos.
- **RNF6.4** Protección de los demás esquemas de la base de datos en el momento de la migración.

7- Confiabilidad:

- **RNF7.1** El gestor de bases posee formas de recuperación de la información en caso de pérdidas de datos o mal proceso de migración.

8- Funcionalidad:

- **RNF8.1** Reducir al mínimo la carga del sistema en el momento en que se realiza la migración.
- **RNF8.2** Ejecución inmediata de los scripts siempre que el gestor de base de datos soporte más transacciones.

9- Software:

- **RNF9.1** Microsoft Visual Studio 2005.
- **RNF9.2** Oracle Database 10g o superior.
- **RNF9.3** Sistema operativo Windows.

10- Hardware:

- **RNF10.1** Servidor de Bases de Datos localizado con requerimientos mínimos de 3gb de almacenamiento para sus ficheros y 5gb para la Base de Datos, 1gbit RAM o superior y 2.4 o superior de velocidad del microprocesador.

2.7 Diagrama de Caso de Uso

Un caso de uso es una secuencia de transacciones que son desarrolladas por un sistema en respuesta a un evento que inicia un actor sobre el propio sistema. Los diagramas de casos de uso sirven para especificar la funcionalidad y el comportamiento de un sistema mediante su interacción con los usuarios y/o otros sistemas. O lo que es igual, un diagrama que muestra la relación entre los actores y los casos de uso en un sistema. Los diagramas de casos de uso se utilizan para ilustrar los requerimientos del sistema al mostrar cómo reacciona una respuesta a eventos que se producen en el mismo. (WIKIMEDIA 2007)

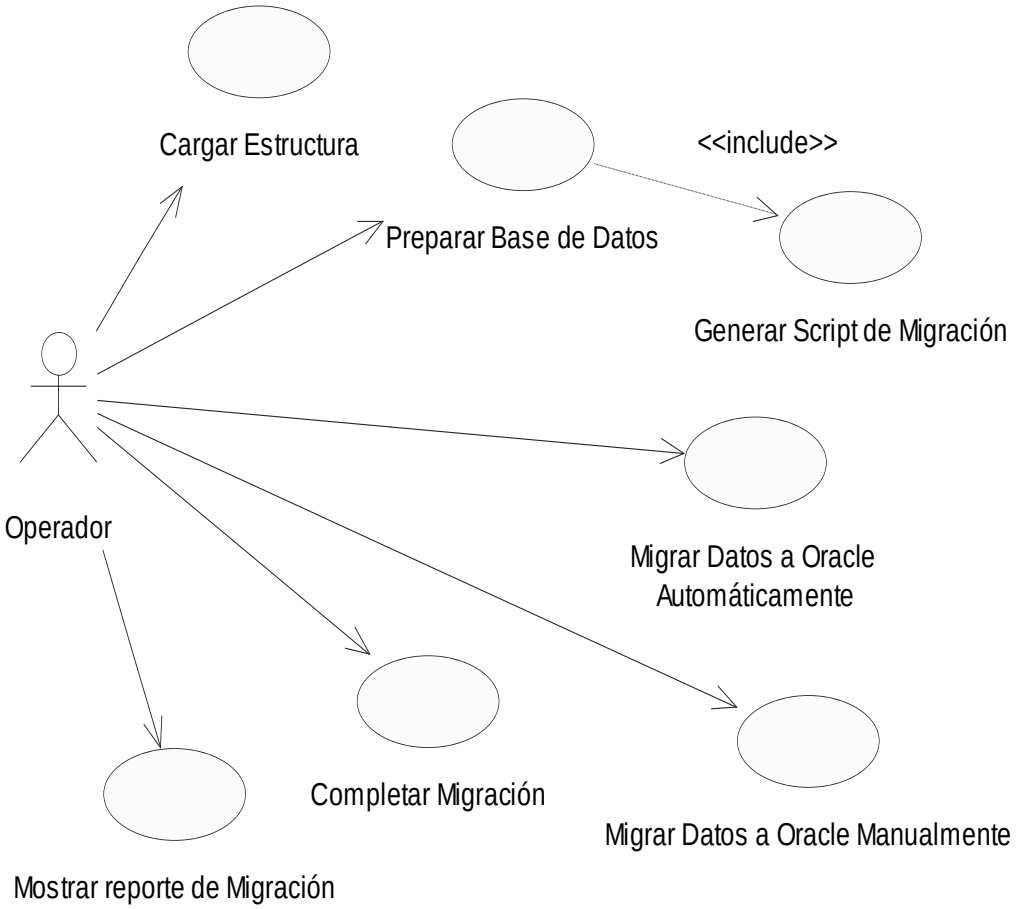

Figura 2. Diagrama de Caso de Uso

2.8 Descripción de los Casos de Uso

Nombre CU: Cargar estructura

Objetivo:

Crear un fichero temporal con solo la información relevante para la migración, el cual será el fichero a utilizar en lo adelante.

Actores:

Operador (Inicia)

Resumen:

Este CU es el encargado de leer el fichero de estructura (DDM) generado por el NATURAL y luego genera un fichero temporal que solo contiene la información relevante para la migración, eliminando la irrelevante, con este fichero es con el que se trabajará para los restantes pasos en la migración.

Complejidad:

Alta.

Precondiciones:

Debe existir un fichero de texto, el cual debe ser extraído del NATURAL y contenga la estructura de la base de datos del ADABAS.

Poscondiciones:

Queda creado un fichero, que contiene la información relevante para la migración.

Flujo básico de eventos	
Acciones del actor	**Respuesta del sistema**
1. El Operador selecciona del menú principal la opción de “Cargar el fichero de estructura”.	1.1. El Sistema muestra un cuadro de diálogo que permite que se seleccione el camino donde se encuentra el fichero de estructura.
2. El Operador selecciona el camino donde se encuentran los datos generados por el NATURAL.	2.1. El Sistema guarda la información del camino donde se encuentran los datos, además de mostrar en pantalla el camino seleccionado.
3. El Operador selecciona el camino donde se encuentra el fichero de estructura.	3.1. El Sistema guarda la información del camino donde se encuentra el fichero y carga el fichero en memoria, además de mostrar en pantalla el camino seleccionado. 3.2. El Sistema procede a eliminar todos los caracteres extraños presentes en el fichero y los convierte a caracteres legibles mientras se pueda. 3.3. El Sistema procede a eliminar todo la información irrelevante para la migración (*archivos vacíos, archivos sin importancia, etcétera*.). 3.4. El Sistema crea un fichero temporal en el cual se almacena la información que se utilizará en los próximos pasos, el mismo estará ubicado en el mismo lugar donde se encuentra el antiguo fichero de estructura. 3.5. Termina la ejecución del caso de uso.

Tabla 2. Cargar estructura

Nombre CU Preparar base de datos

Objetivo

Crear la estructura de las tablas de la base de datos en Oracle.

Actores

Operador (Inicia)

Resumen

Este caso de uso es el encargado de crear una nueva base de datos en Oracle con todas sus tablas, extrayendo la información del fichero temporal limpio, previamente creado**.**

Complejidad

Media.

Precondiciones

Debe haberse ejecutado el caso de uso “Cargar estructura”.

Poscondiciones

Queda creada una base de datos en Oracle, que contiene las tablas extraídas del fichero de estructura.

Flujo básico de eventos

Acciones del actor	**Respuesta del sistema**
1. El Operador selecciona del menú principal la pestaña de “Procesamiento”.	1.1. El Sistema muestra un cuadro de diálogo que le permite realizar varias opciones.
2. El Operador selecciona la opción de conectarse a la base de datos, suministrando la entrada de datos correspondiente (*Nombre de la base de datos, Usuario, Contraseña*).	2.1. El Sistema crea una conexión con la base de datos especificada, en caso de estar correctos los parámetros de entrada.

3. El Operador selecciona la opción de “Generar Tablas”.	3.1. El Sistema procede a leer el fichero de estructura generado anteriormente y construye varios scripts SQL que contendrán la estructura (DDL) de tablas con las cuales se trabajará en nuestra base de datos. 3.2. El sistema realiza una previa normalización de las tablas cargadas. 3.3. El Sistema ejecuta dentro de la base de datos los scripts anteriormente generados y genera así las nuevas tablas donde se almacenarán los datos. 3.4. Termina la ejecución del caso de uso.

Flujo alterno de eventos

2.1. En caso que los datos de entrada contengan error se emite un mensaje de error.

Tabla 3. Preparar base de datos

Nombre CU Generar script de migración

Objetivo

Crear una serie de scripts SQL que contendrán la información a incluir en la base de datos.

Actores

CU: Preparar base de datos (Inicia).

Resumen

Este CU se ejecuta una vez concluida la ejecución del CU: Preparar base de datos, el mismo es el encargado de generar una lista de scripts, preparados para ser ejecutado en el servidor que contienen la información a incluir en la base de datos, leída de los ficheros de texto extraídos del ADABAS.

Complejidad

Alta.

Precondiciones

Debe haberse ejecutado el CU: “Preparar base de datos”, además de haberle proporcionado al software el camino donde se encuentran los ficheros de datos del ADABAS.

Poscondiciones

Queda creada una serie de scripts que contienen la información a ser incluida en la base de datos.

Flujo básico de eventos

Acciones del actor	**Respuesta del sistema**
1. Comienza con la selección de llenado de las tablas creadas por el CU anterior	1.1. El Sistema lee el camino donde se encuentran los datos creando un nombre de fichero de datos para cada tabla.
	1.2. El sistema hace una búsqueda de todos los ficheros de datos creados anteriormente, en caso que existan.
	1.3. El sistema carga en memoria la información del fichero de datos de la tabla correspondiente, y convierte a codificación UTF-8 los caracteres leídos.
	1.4. El Sistema procede a leer la estructura de la tabla y comienza a generar un scripts leyendo la información del fichero cargado en memoria y creando las sentencias SQL a almacenar para adicionarlas a la base de datos.
	1.5. El sistema lanza un mensaje el cual pregunta si está listo para pasar a la próxima etapa de la migración.
	1.6. Termina el caso de uso.

Flujo alterno de eventos

1.2. En caso que el fichero de datos de la tabla no se encuentre entonces esta tabla se almacenará en una lista de tablas perdidas para posteriormente suministrarle el camino manualmente.

1.5. Si se cancela el mensaje el sistema detiene el proceso de migración y elimina las tablas que acaba de crear.

Tabla 4. Generar script de migración

Nombre CU Migrar datos a Oracle automáticamente.

Objetivo

El sistema almacena en la base de datos la información que contienen los scripts.

Actores

Operador (Inicia).

Resumen

Este CU es el encargado de ejecutar automáticamente todos los scripts de migración previamente generados, insertando todos los datos correspondientes en la base de datos creada.

Complejidad

Media.

Precondiciones

Debe haberse ejecutado el CU: “Generar scripts de migración”.

Poscondiciones

Queda almacenada toda la información en la base de datos de los scripts anteriormente generados.

Flujo básico de eventos

Acciones del actor	**Respuesta del sistema**
1. El usuario acepta el mensaje de pasar al próximo nivel de la migración	1.1. El Sistema comienza a almacenar en hilos de ejecución las sentencias que se encuentran en memoria.
	1.2. El sistema procede a ejecutar los scripts en la base de datos.
	1.3. En caso de detectarse inconsistencia en el nombre de los ficheros se habilita la opción de migración manual.
	1.4. Termina la ejecución del caso de uso.

Flujo alterno de eventos

1. En caso de no aceptarse el mensaje ir al flujo básico 1.4.

Tabla 5. Migrar datos a Oracle automáticamente

Nombre CU Migrar datos a Oracle manualmente.

Objetivo

El sistema adiciona en la base de datos las tablas que no encontró de forma automática.

Actores

Operador (Inicia).

Resumen

Este CU es el encargado de permitir al Operador de la migración ejecutar manualmente todos los scripts de migración previamente creados, en el caso que hayan tenido errores en la migración automática, insertando todos los datos correspondientes en la base de datos en Oracle.

Complejidad

Media.

Precondiciones

Debe haberse ejecutado el CU: "Migrar datos a Oracle automáticamente".

Debe haberse suministrado el camino de los datos a las tablas que no encontró el sistema automáticamente.

Poscondiciones

Queda almacenada toda la información en la base de datos de los script que no encontró el sistema.

Flujo básico de eventos

Acciones del actor	**Respuesta del sistema**
1. El usuario selecciona la pestaña de "Mig Manual".	1.1. El Sistema brinda una serie de opciones.

2. El usuario selecciona el camino de los ficheros de datos del ADABAS.	2.1. El sistema almacena el camino donde se encuentran cada uno de los ficheros que no encontró de forma automática.
3. El usuario presiona el botón “Comenzar Migración Manual”	3.1. El sistema lee la dirección de los ficheros suministrados anteriormente. 3.2. El sistema codifica los ficheros leídos a UTF-8. 3.3. El sistema crea una serie de scripts SQL que contienen la información de los ficheros leídos. 3.4. El sistema procede a almacenar la información en la base de datos de los scripts guardados en memoria. 3.5 Termina el caso de uso.

Tabla 6. Migrar datos a Oracle manualmente

CU Mostrar reporte de migración.

Objetivo

El sistema muestra una serie de información acerca de la migración.

Actores

Operador (Inicia).

Resumen

Este caso de uso se encarga de mostrar un pequeño reporte que informa de los resultados de la migración, mostrando posibles errores ocurridos y genera un fichero con dicha información.

Complejidad

Baja.

Precondiciones

Debe haberse ejecutado el CU: “Migrar datos a Oracle automáticamente”.

Poscondiciones

Queda generado un reporte que se muestra en el software y se almacena en un fichero.

Flujo básico de eventos

Acciones del actor	**Respuesta del sistema**
1. El Operador selecciona la pestaña de “Información”	1.1. El sistema carga la información recopilándola de lo sucedido en el transcurso de la migración.
	1.2. El Sistema muestra una serie de información de lo sucedido en el proceso de migración.
2. El Operador presiona el botón “Guardar reporte”	2.1. El sistema muestra un cuadro de diálogo para que seleccione el camino donde desea guardar el reporte.
3. El Operador selecciona el camino del reporte.	3.1. El sistema almacena el camino donde posteriormente se almacenará el reporte.
	3.2. El sistema procede a guardar la información del reporte.
	3.3. Termina el caso de uso.

Tabla 7. Mostrar Reporte de migración

CU Completar migración.

Objetivo

Crear un esquema de la base de datos final.

Actores

Operador (Inicia).

Resumen

Este CU es el encargado de eliminar todas las trazas creadas y hacer cambios en los nombres de las tablas, dejando la base de datos funcionando**.**

Complejidad

Baja.

Precondiciones

Debe haberse ejecutado el CU: “Migrar datos a Oracle automáticamente”.

Debe haberse ejecutado el CU: “Migrar datos a Oracle manualmente”.

Poscondiciones

La base de datos queda lista para su explotación.

Flujo básico de eventos

Acciones del actor	**Respuesta del sistema**
1. El Operador presiona la pestaña “Completar Migración”.	1.1. El Sistema muestra una serie de información acerca de lo que va a hacer.
2. El Operador presiona el botón “Terminar”	2.1. El sistema procede a eliminar todos los ficheros temporales creados.
	2.2. El sistema procede a modificar los nombres de las tablas creadas para crear uniformidad con las tablas de la base de datos en desarrollo.
	2.3. El sistema cierra todas las conexiones abiertas y recursos utilizados.

2.4. Termina el caso de uso.

Tabla 8. Completar migración

2.9 Conclusiones

En este Capítulo se ha realizado una breve descripción del actor del sistema y sus responsabilidades, identificando los requerimientos funcionales, los cuales son de gran importancia para el desarrollo del sistema, ya que constituyen la funcionalidad del mismo, así como presentando el diagrama de caso de uso, que refleja un esquema de lo que se debe implementar. También se identificaron los requerimientos no funcionales, los que son de gran importancia ya que son propiedades o cualidades que el sistema debe cumplir y además se detallaron todos los casos de usos, siendo estos de gran importancia para la construcción del sistema propuesto.

Capítulo

3

3 CAPÍTULO3: DESCRIPCIÓN DE LA SOLUCIÓN PROPUESTA

3.1 Introducción

En el presente capítulo se realiza el diseño de la propuesta de solución, seleccionando una arquitectura y modelándose los artefactos que contribuyen al desarrollo de este Software. Se analiza cómo va a estar distribuido el sistema y como quedará estructurado en forma de módulos el sistema.

3.2 Descripción de la arquitectura

Una Arquitectura de Software, también denominada Arquitectura Lógica, consiste en un conjunto de patrones y abstracciones coherentes que proporcionan el marco de referencia necesario para guiar la construcción del software de un sistema de información. La Arquitectura de Software establece los fundamentos para trabajar en una línea común que permite alcanzar los objetivos y necesidades del sistema de información. (RUMBAUGH, J. 2000)

3.2.1 Arquitectura 3 Capas

El Software de Migración será desarrollado sobre la arquitectura de 3 capas o niveles. Este Patrón define cómo organizar el modelo de diseño en capas. Los componentes de una capa solo pueden hacer referencia a componentes en capas inmediatamente inferiores, esto simplifica la comprensión y la organización de sistemas complejos.

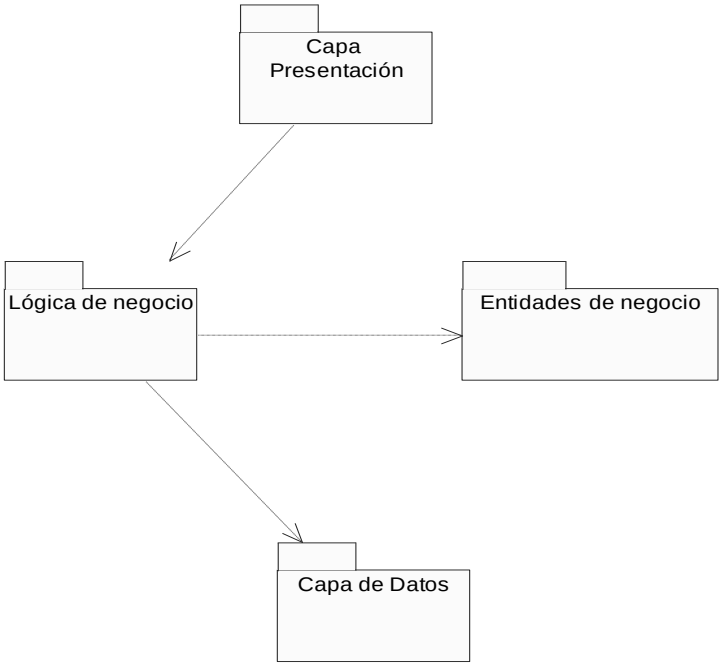

Figura 3. Arquitectura de 3 Capas

- La capa de presentación o interfaz de usuario.

En este caso está formada por los formularios y los controles que se encuentran en los formularios. Capa con la que interactúa el usuario.

- La capa de negocio.

Esta capa está formada por las entidades y clases que manejan la lógica de negocio, llamadas clases de control, que representan objetos que van a ser manejados o consumidos por toda la aplicación.

- La capa de acceso a datos.

Esta contiene las clases que interactúan con la Base de Datos, estas clases altamente especializadas, realizan todas las operaciones con la BD de forma transparente para la capa de negocio.

El sistema estará dividido en 4 módulos fundamentales, los cuales se describirán a continuación:

- **Módulo de Lectura**

Este módulo es el encargado de leer el fichero de estructura del ADABAS y luego genera un fichero temporal que solo contiene la información relevante para la migración, eliminando la irrelevante, con este fichero es con el que trabajará.

- **Módulo de procesamiento**

Este módulo es el encargado de crear la nueva BD en Oracle con todas sus tablas, extrayendo la información del fichero temporal limpio, previamente creado.

- **Módulo de generación**

Este módulo es el encargado de generar una lista de scripts, preparados para ser ejecutados en el servidor, scripts que contienen la información a incluir en la base de datos, leída de los ficheros de texto extraídos del ADABAS.

- **Módulo de migración**

Este módulo es el encargado de ejecutar automáticamente todos los scripts de migración previamente generados, insertando todos los datos correspondientes en la base de datos creada, en caso de ser automáticamente y permite además poder hacer manualmente la migración. Además se encarga de mostrar reportes con respecto a la migración.

3.3 Modelo de diseño

En la fase de diseño se modela el sistema de manera que soporte todos los requerimientos, tanto funcionales como no funcionales. Uno de sus propósitos fundamentales es: “Crear una entrada apropiada y un punto de partida para actividades de implementación subsiguientes capturando los requerimientos o subsistemas individuales, interfaces y clases”, se dice que el diseño es un esquema a la implementación. (BARRIENTOS, E. 2005)

La esencia de esta fase es la elaboración de diagramas de interacción, que muestran gráficamente como los objetos se comunican entre ellos a fin de cumplir con los requerimientos. Estos diagramas permiten la realización de los diagramas de clases del diseño, los cuales resumen la definición de las clases que se pueden implementar en el software.

3.3.1 Diagrama de clases CU Cargar Lectura

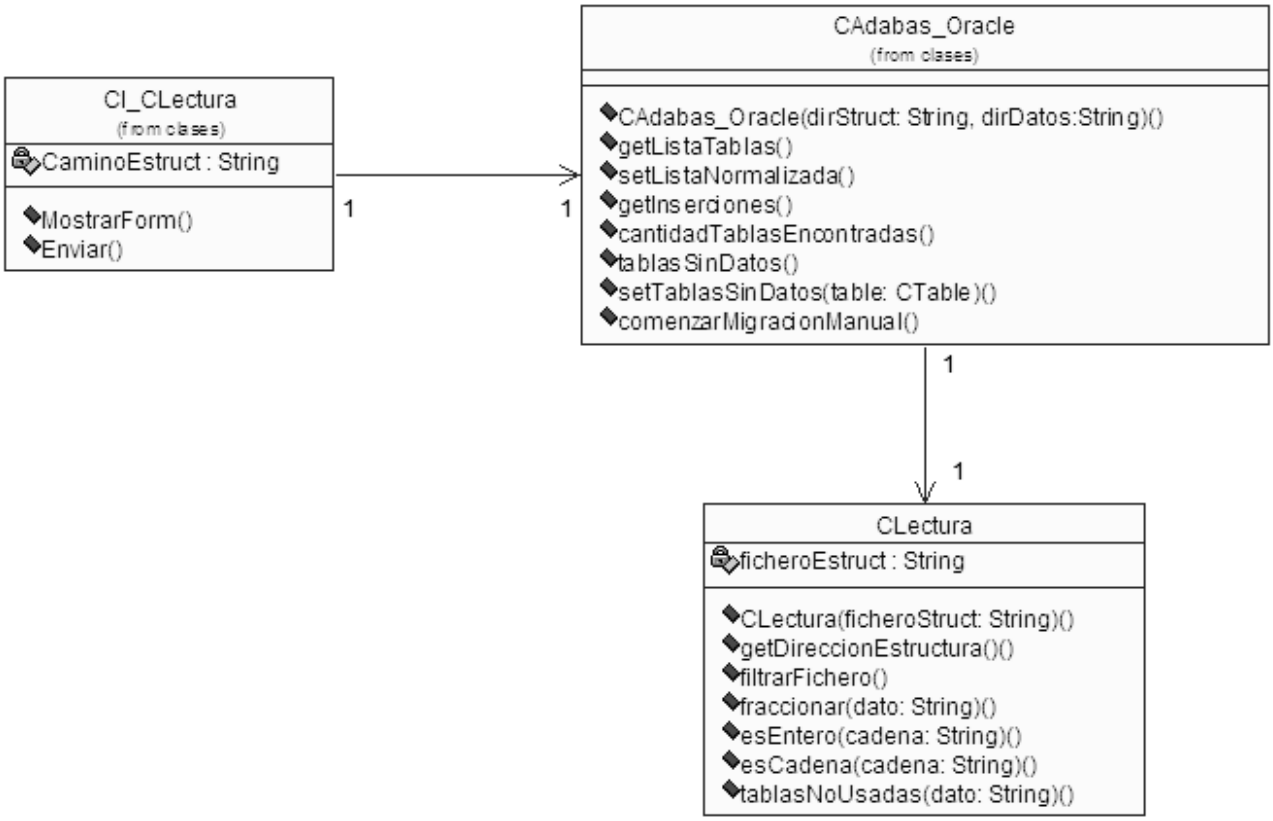

Figura 4 Diagrama de clases CU Cargar Lectura

3.3.2 Diagrama de clases CU Preparar Base de Datos

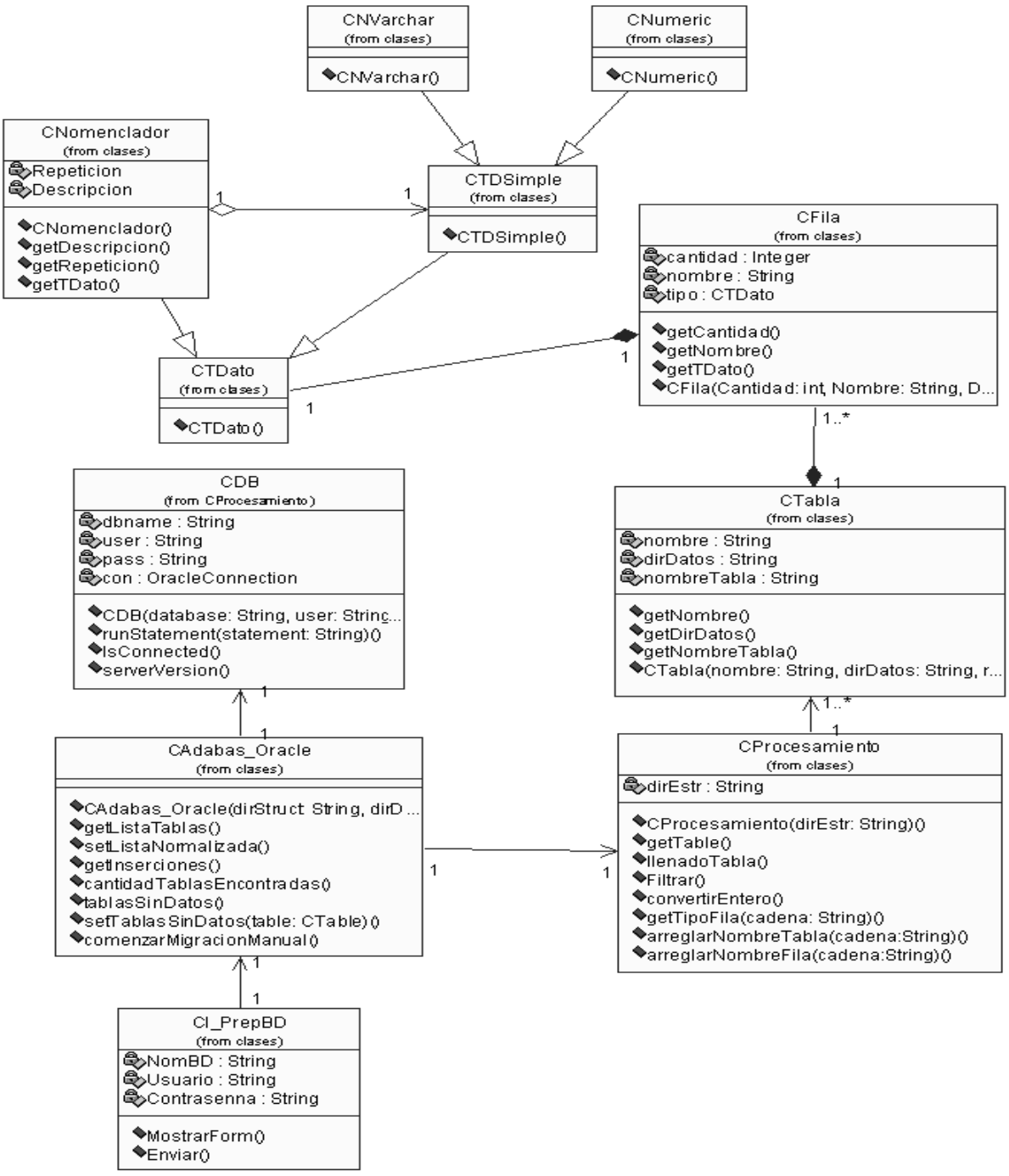

Figura 5 Diagrama de clases CU Preparar Base de Datos

3.3.3 Diagrama de clases CU Generar Script de Migración

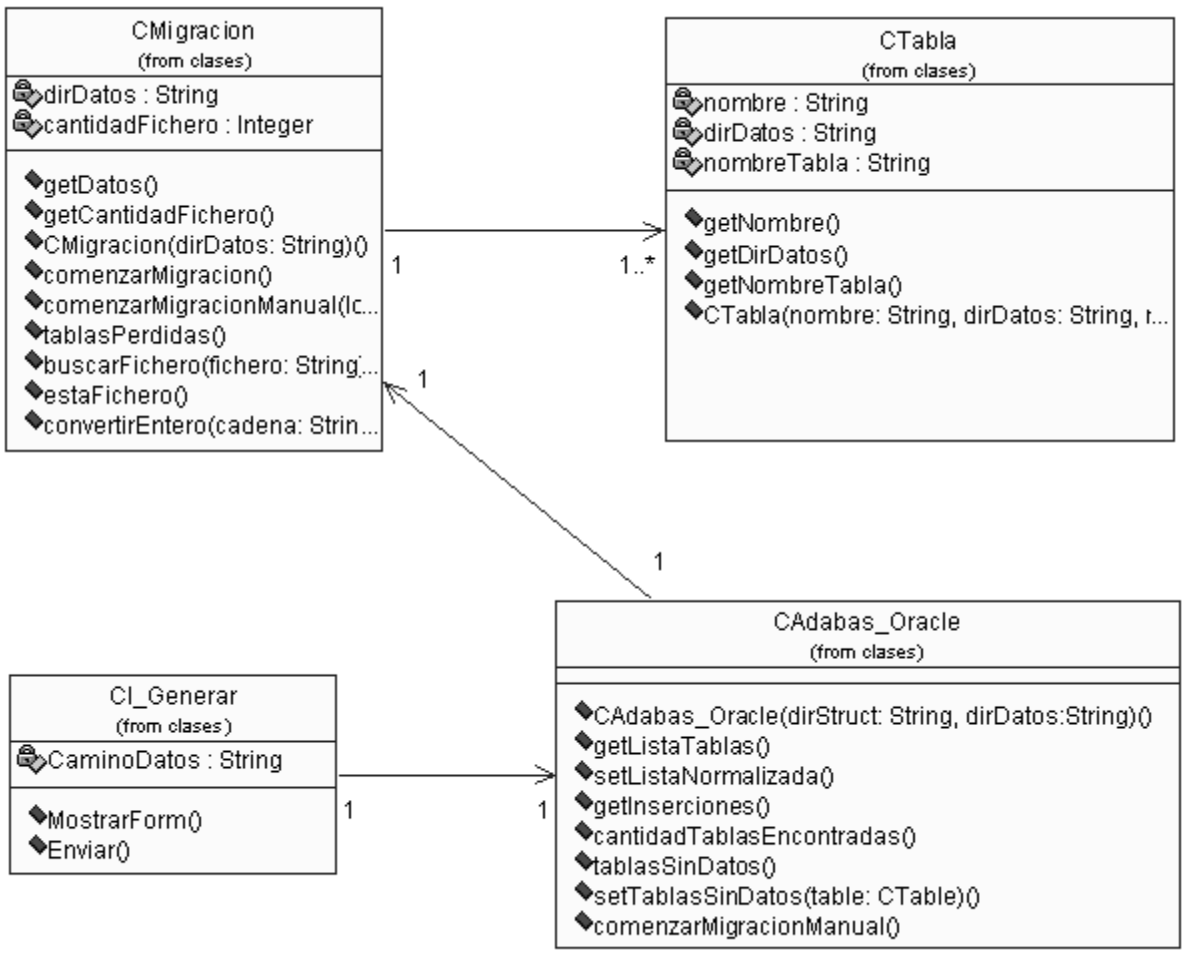

Figura 6 Diagrama de clases CU Generar Script de Migración

3.3.4 Diagrama de clases CU Migrar los datos automáticamente

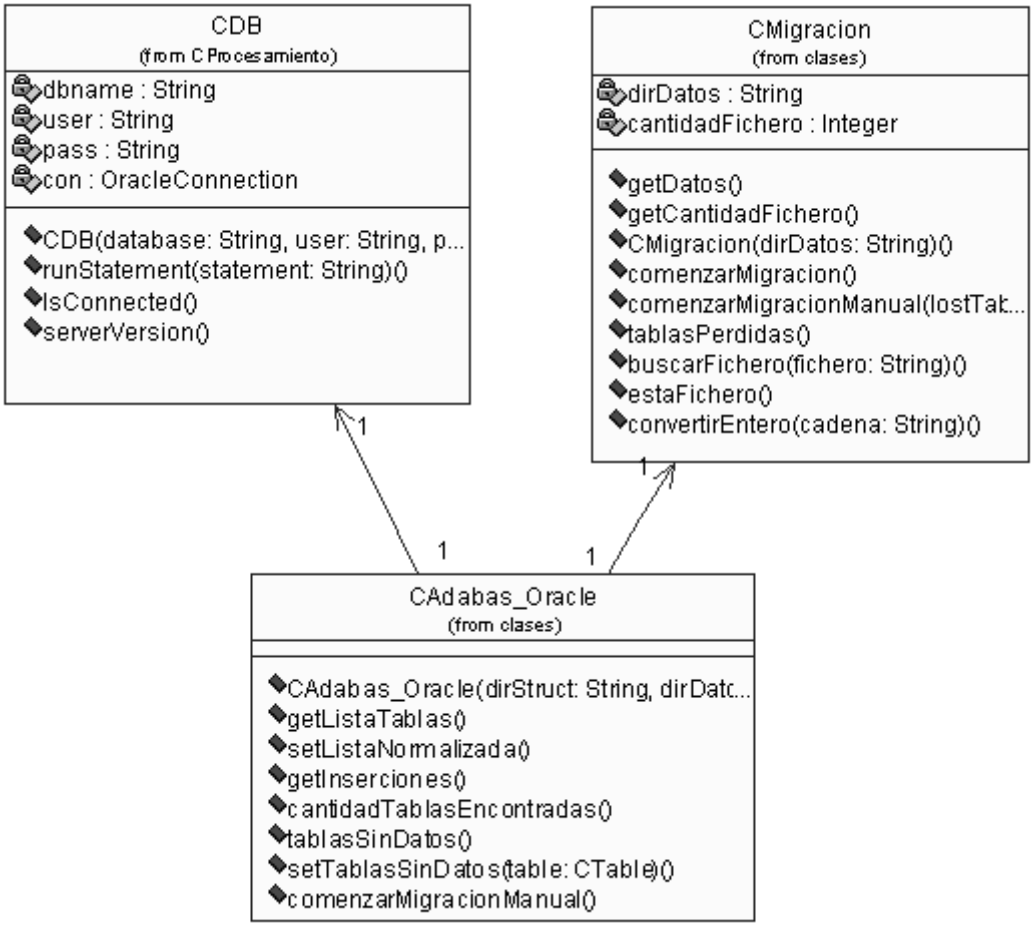

Figura 7 Diagrama de clases CU Migrar los datos automáticamente

3.3.5 Diagrama de clases CU Mostrar Reporte

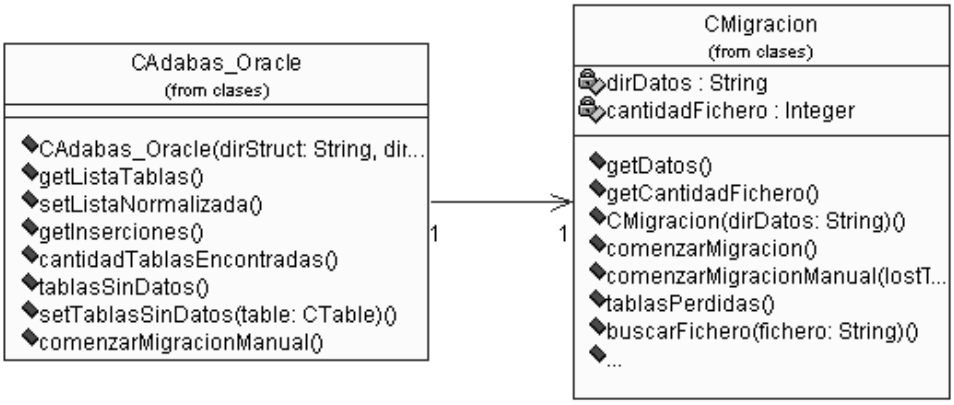

Figura 8 Diagrama de clases CU Mostrar Reporte

3.3.6 Diagrama de clases CU Migrar los datos Manualmente

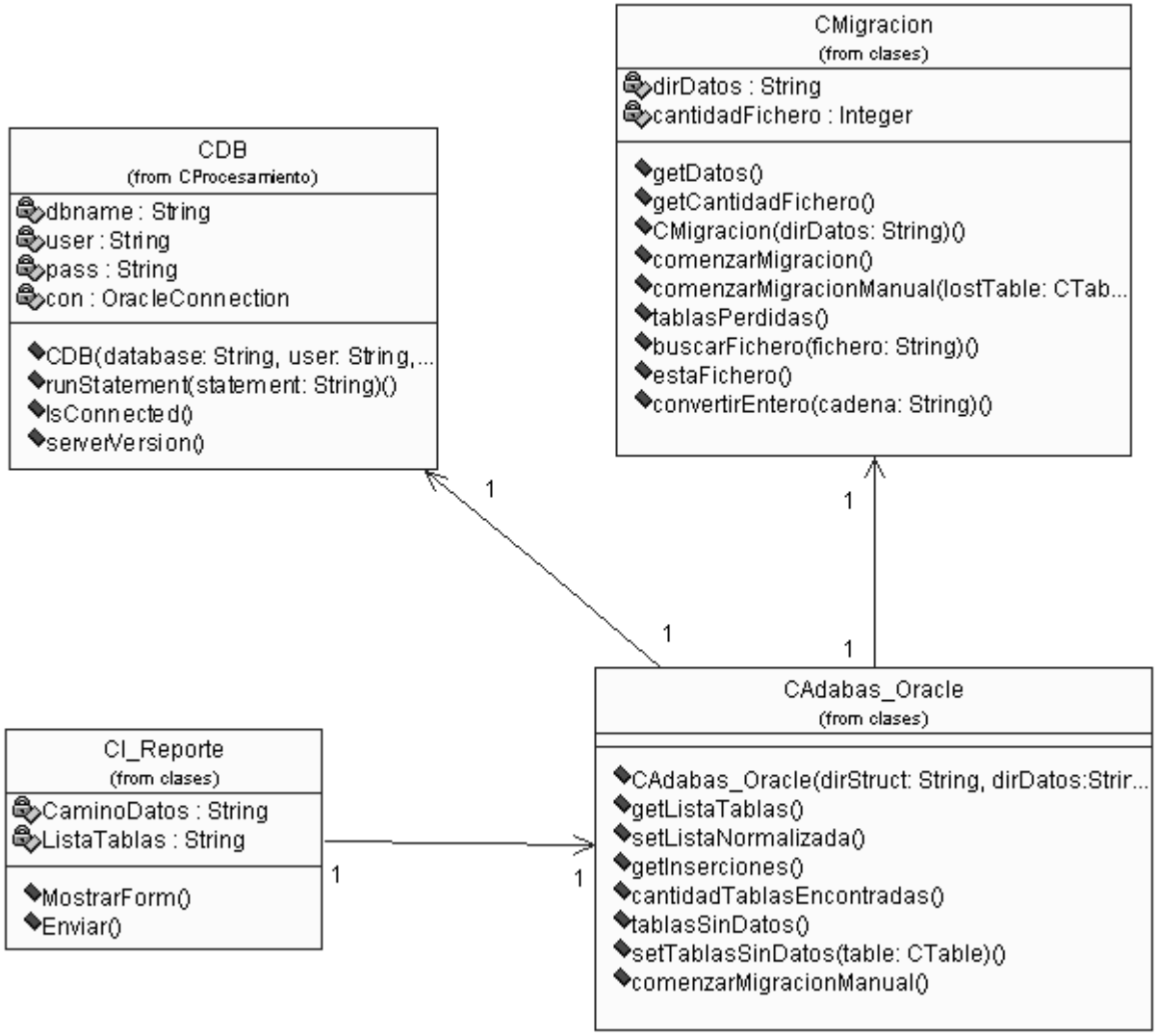

Figura 9 Diagrama de clases CU Migrar los datos Manualmente

3.3.7 Diagrama de clases CU Completar Migración

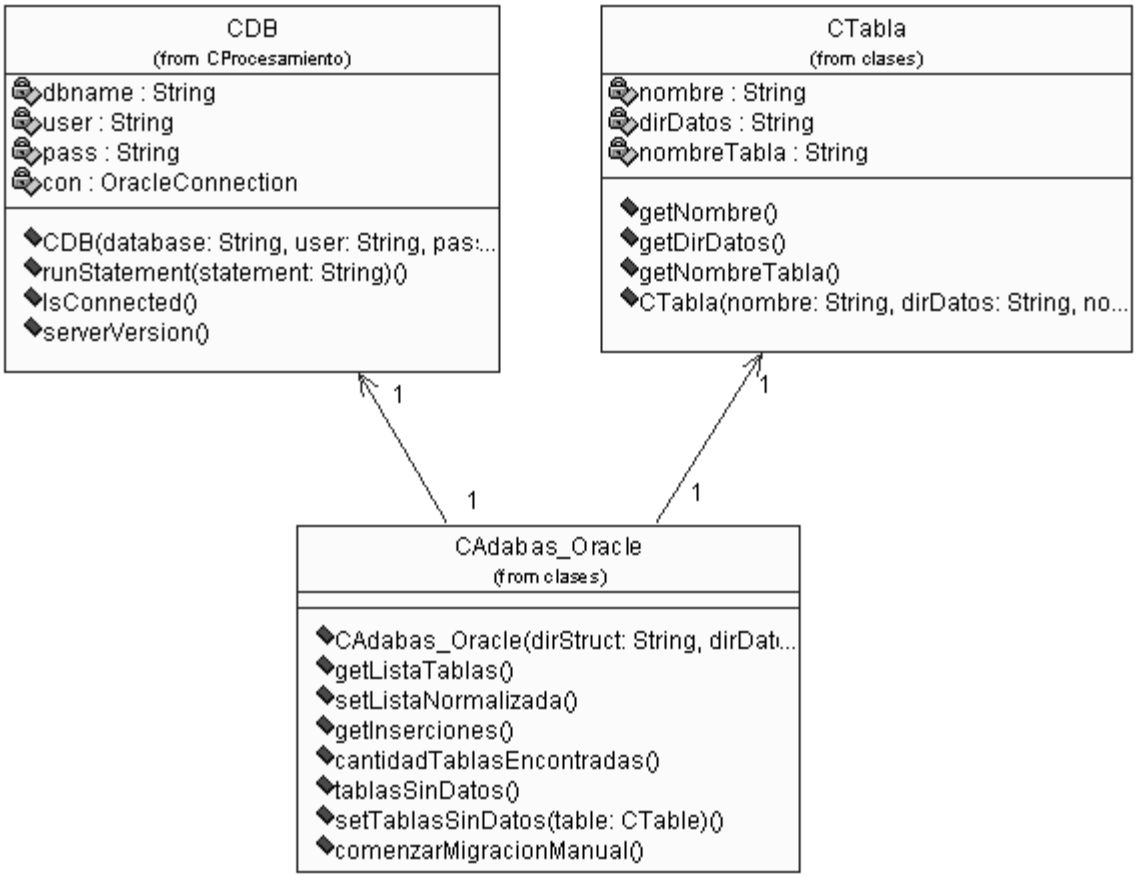

Figura 10 Diagrama de clases CU Completar Migración

3.4 Descripción de las clases

Tabla 9 CTDSimple

Nombre: CTDSimple	
Tipo de clase: Entidad	
Atributo	**Tipo**
Para cada responsabilidad:	
Nombre:	CTDSimple()
Descripción:	Constructor de la clase.

CTDSimple: Esta clase se encarga de establecer uno de los tipos de datos que contiene una fila.

Tabla 10 CTDato

Nombre: CTDato	
Tipo de clase: Entidad	
Atributo	**Tipo**
Para cada responsabilidad:	
Nombre:	CTDato()
Descripción:	Constructor de la clase.

CTDato: Esta clase se encarga de establecer uno de los tipos de datos que contiene una fila.

Tabla 11 CNVarchar

Nombre: CNVarchar	
Tipo de clase: Entidad	
Atributo	**Tipo**
Para cada responsabilidad:	
Nombre:	CNVarchar()
Descripción:	Constructor de la clase.

CNVarchar: Esta clase se encarga de establecer uno de los tipos de datos que contiene una fila.

Tabla 12 CNumeric

Nombre: CNumeric	
Tipo de clase: Entidad	
Atributo	**Tipo**
Para cada responsabilidad:	
Nombre:	CNumeric()
Descripción:	Constructor de la clase.

CNumeric: Esta clase se encarga de establecer uno de los tipos de datos que contiene una fila.

Tabla 13 CNomenclador

Nombre: CNomenclador	
Tipo de clase: Entidad	
Atributo	**Tipo**
Repetición	int
tipoDato	CTDSimple

Descripción	String
Para cada responsabilidad:	
Nombre:	CNomenclador(Descripción: String, Repetición: int, tipoDato: CTDSimple)
Descripción:	Constructor de la clase.
Nombre:	getRepeticion()
Descripción:	Retorna la cantidad de repeticiones que tiene el Nomenclador
Nombre:	getDescripcion()
Descripción:	Retorna el nombre del nomenclador
Nombre:	getTipoDato()
Descripción:	Retorna el tipo de datos del nomenclador

CNomenclador: Esta tabla se encarga de establecer un tipo de datos que contiene una fila.

Tabla 14 CDB

Nombre: CDB	
Tipo de clase: Controladora	
Atributo	**Tipo**
dbname	String
user	String
pass	String
con	OracleConnection
Para cada responsabilidad:	
Nombre:	CDB(database: String, user: String, pass: String)
Descripción:	Constructor de la clase.
Nombre:	Connect()
Descripción:	Realiza la conexión con la base de datos
Nombre:	runStatement(sentencia: String)
Descripción:	Ejecuta una sentencia en el servidor de base de datos.
Nombre:	IsConnected()
Descripción:	Retorna si estas conectado a la Base de Datos.

CDB: Esta clase se encarga de hacer la conexión con la base de datos, además de contener todas las funcionalidades que se ejecutan con el servidor, es el vínculo entre el software y el servidor de base de datos.

Tabla 15 CLectura

Nombre: CLectura	
Tipo de clase: Controladora	
Atributo	**Tipo**
FicheroEstruct	String
Para cada responsabilidad:	
Nombre:	CLectura(aFicheroEstruct: String)
Descripción:	Constructor de la clase.
Nombre:	GetDireccionEstructura()
Descripción:	Retorna la dirección del fichero donde se encuentra la estructura.
Nombre:	FiltrarFichero ()
Descripción:	Filtra el fichero de estructura eliminando las incoherencias creando uno temporal con la información real de la estructura para la migración.
Nombre:	Fraccionar(dato: String)
Descripción:	Fracciona la cadena para mejorar su uso.
Nombre:	EsNumero(cadena: String)
Descripción:	Retorna si una cadena es un número.
Nombre:	EsCadena(cad: String)
Descripción:	Retorna si es una cadena válida para la migración.
Nombre:	TablasNoUsadas(dato: String)
Descripción:	Devuelve si la tabla se usa en la migración.

CLectura: Esta clase recibe como entrada una dirección donde se encuentra el fichero de la estructura de la base de datos y genera como salida un fichero temporal con la estructura de la nueva base de datos que se va a migrar. Decide cuales serán las tablas que no se usarán para la migración además de eliminar todas las incoherencias en el fichero original generado por el NATURAL.

Tabla 16 CProcesamiento

Nombre: CProcesamiento	
Tipo de clase: Controladora	
Atributo	**Tipo**
Tablas	List<CTabla>
DirEstr	String
Para cada responsabilidad:	
Nombre:	CProcesamiento(DirEstr: String)
Descripción:	Constructor de la clase.
Nombre:	GetTable()
Descripción:	Retorna las tablas que extrajo del fichero.
Nombre:	LlenadoTabla ()
Descripción:	Comienza a generar la lista de tablas que lee del fichero generado por la clase de Lectura.
Nombre:	Filtrar(dato: String)
Descripción:	Filtra la cadena que lee de caracteres extraños para la migración.
Nombre:	ConvertirEntero(palabra: String)
Descripción:	Convierte a entero una cadena parámetro.
Nombre:	getTipoFila(cadena: String)
Descripción:	Retorna el tipo de fila.
Nombre:	arreglarNombreTabla(cadena: String)
Descripción:	Define un nuevo nombre para la Tabla generada.
Nombre:	arreglarNombreFila(cadena: String)
Descripción:	Define un nuevo nombre para la Fila generada.

CProcesamiento: Clase de alto peso en el proceso de migración, se encarga de generar toda la estructura a nivel de objetos, recibiendo como entrada un fichero generado por la clase Lectura y dando como salida una lista de tablas con toda la estructura a nivel de objetos de la base de datos final.

Tabla 17 CGenerar

Nombre: CGenerar	
Tipo de clase: Controladora	
Atributo	**Tipo**
Datos	List<CTabla>
Para cada responsabilidad:	
Nombre:	CGenerar(DirEst: String, Datos: List<CTabla>)
Descripción:	Constructor de la clase.
Nombre:	GetDatos()
Descripción:	Retorna los datos que va a generar.
Nombre:	GenerarExecScript()
Descripción:	El retorno es una lista con los script SQL de la estructura que se va a generar en el servidor de base de datos.
Nombre:	estaNomenclador(ntable: String, cantidad: int)
Descripción:	Retorna si se encuentra el nomenclador a crear.

CGenerar: Esta clase tiene como entrada una lista de tablas y a partir de ellas genera una lista de script SQL para ejecutarlos en el servidor de base de datos.

Tabla 18 CMigración

Nombre: CMigracion	
Tipo de clase: Controladora	
Atributo	**Tipo**
dirDatos	String
CantidadFichero	String
Datos	List<CTabla>
lostTables	List<CTabla>
Para cada responsabilidad:	
Nombre:	CMigracion(Tabla: List<CTabla>, dirDatos: String)
Descripción:	Constructor de la clase.

Nombre:	GetDatos ()
Descripción:	Retorna la lista de datos.
Nombre:	GetCantidadFichero ()
Descripción:	Retorna la cantidad de ficheros.
Nombre:	ComenzarMigracion()
Descripción:	Recibe como entrada una lista de Tablas y devuelve una lista de script para ejecutar en el servidor.(Proceso automático)
Nombre:	ComenzarMigracionManual()
Descripción:	Recibe como entrada una lista de Tablas y devuelve una lista de script para ejecutar en el servidor. (Proceso manual)
Nombre:	TablasPerdidas ()
Descripción:	Retorna las tablas que no pudo encontrar los datos en la carpeta de origen de los datos.
Nombre:	setTablasPerdidas ()
Descripción:	Establece los valores para las tablas perdidas.
Nombre:	BuscarFichero(nombre: String)
Descripción:	Busca los ficheros correspondientes a cada una de las tablas en la carpeta donde están los ficheros de datos.
Nombre:	estaFichero(aFich: String)
Descripción:	Retorna si se encuentra el fichero
Nombre:	convertirEntero(palabra: String)
Descripción:	Función para convertir a entero un valor.

CMigracion: Clase de importancia elevada pues es la que se encarga de generar los scripts que contienen todos los datos que se van a insertar dentro de las tablas generadas. Además tiene como funcionalidad determinar y seleccionar las tablas para las que no encontró información, clases que serán devueltas al operador para su futura manipulación.

Tabla 19 CAdabas_Oracle

Nombre: CAdabas_Oracle	
Tipo de clase: Controladora	
Atributo	**Tipo**
Migracion	CMigracion
Datos	List<CTabla>
ListaTablas	List<string>
ListaInserciones	List<string>
Para cada responsabilidad:	
Nombre:	CAdabas_Oracle(DirEst: string, dirDatos: string)
Descripción:	Constructor de la clase.
Nombre:	GetDatos()
Descripción:	Retorna la lista de datos.
Nombre:	GetListaTablas()
Descripción:	Retorna la lista de tablas que se van a migrar
Nombre:	GetInserciones()
Descripción:	Retorna un conjunto de inserciones que se va a ejecutar en el servidor.
Nombre:	cantidadTablasEncontradas()
Descripción:	Retorna la cantidad de tablas que contienen los datos para la migración.
Nombre:	tablasSinDatos()
Descripción:	Retorna las tablas que no contienen datos en la migración.
Nombre:	setTablasSinDatos(List<CTabla> table)
Descripción:	Establece las tablas sin datos.
Nombre:	comenzarMigracionManual(List<CTabla> tablas)
Descripción:	Recibe como parámetro una lista de tablas para procesar una migración manual.

CAdabas_Oracle: Clase controladora para el manejo de todas las funciones a implementar en el sistema.

3.5 Diagramas de secuencia del diseño.

3.5.1 Diagrama de secuencia CU Cargar Lectura.

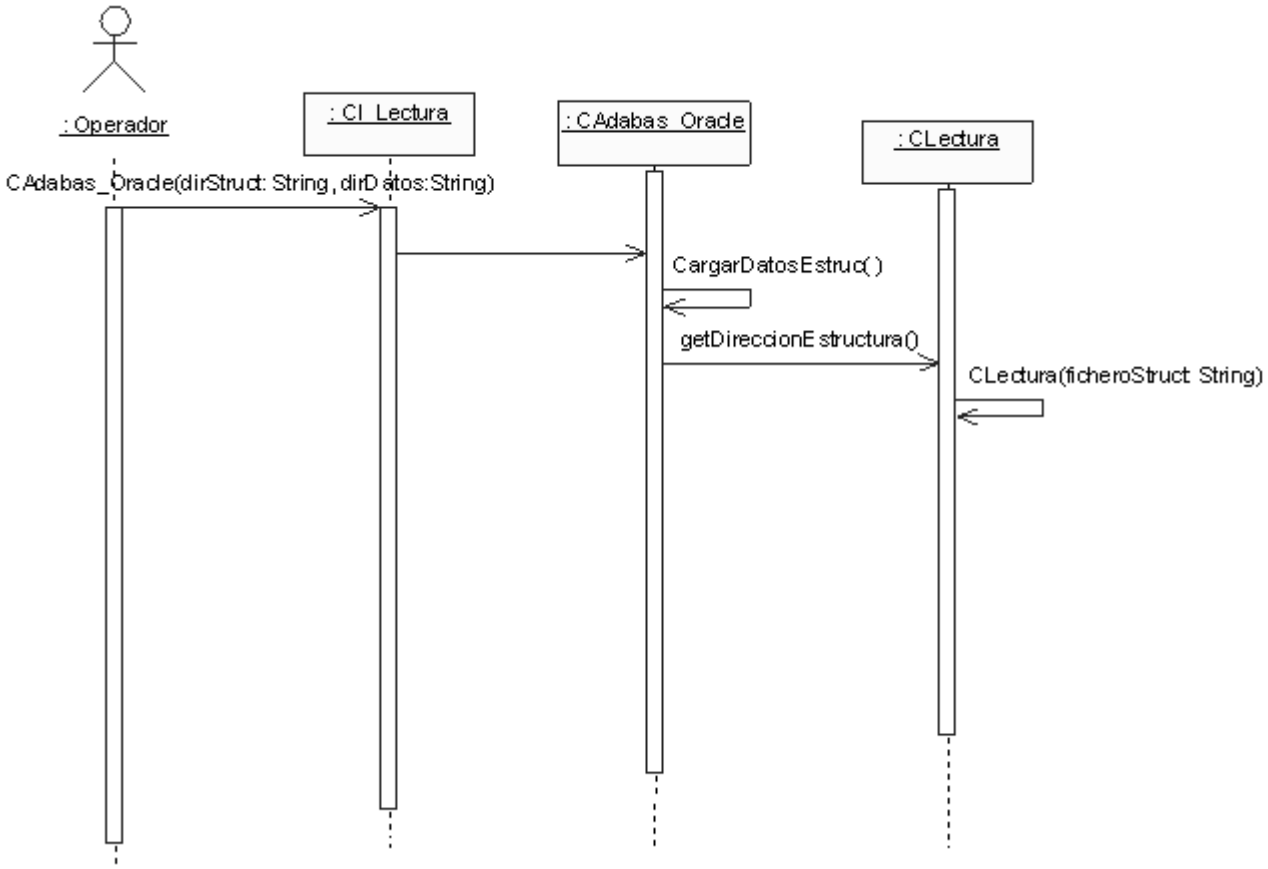

Figura 12. Diagrama de Secuencia CU Cargar Lectura.

3.5.2 Diagrama de secuencia CU Preparar Base de Datos.

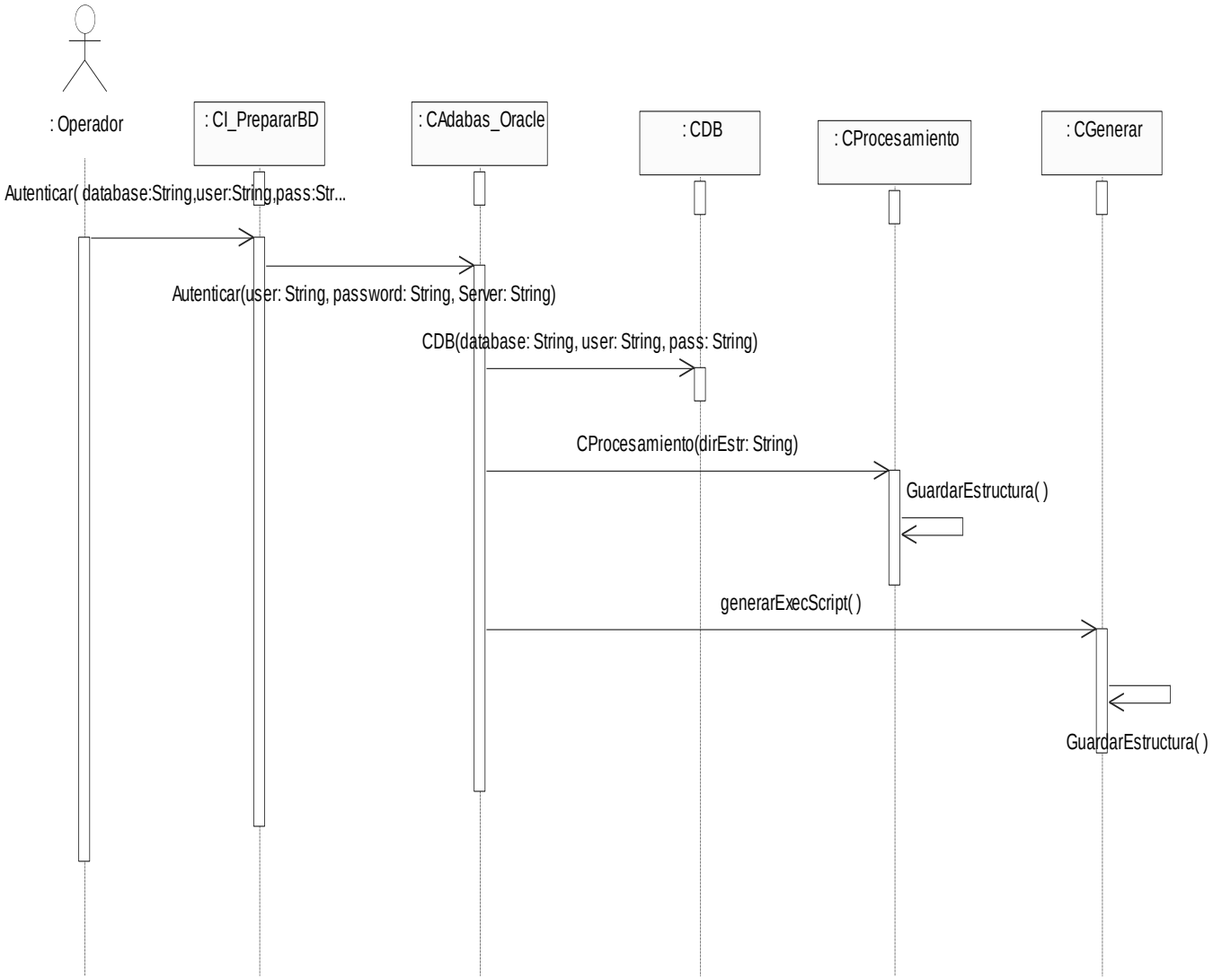

Figura 13. Diagrama de Secuencia CU Preparar Base de Datos.

3.5.3 Diagrama de secuencia CU Generar Script de migración.

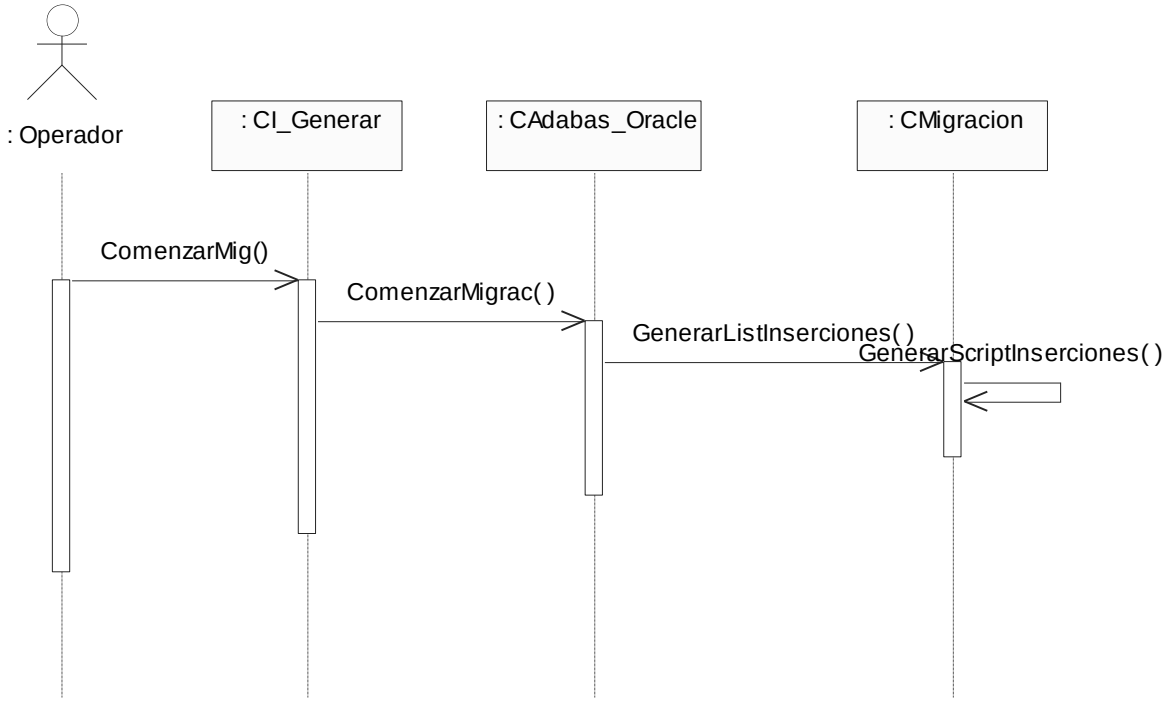

Figura 14. Diagrama de Secuencia CU Generar Script de migración.

3.5.4 Diagrama de secuencia CU Migrar datos Automáticamente.

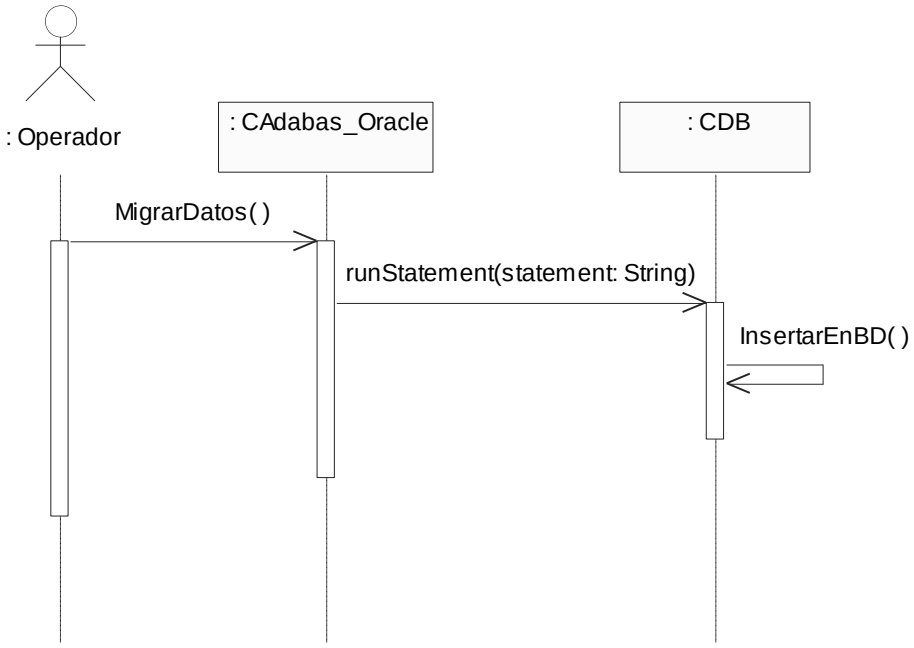

Figura 15. Diagrama de Secuencia CU Migrar los datos Automáticamente.

3.5.5 Diagrama de secuencia CU Mostrar reporte.

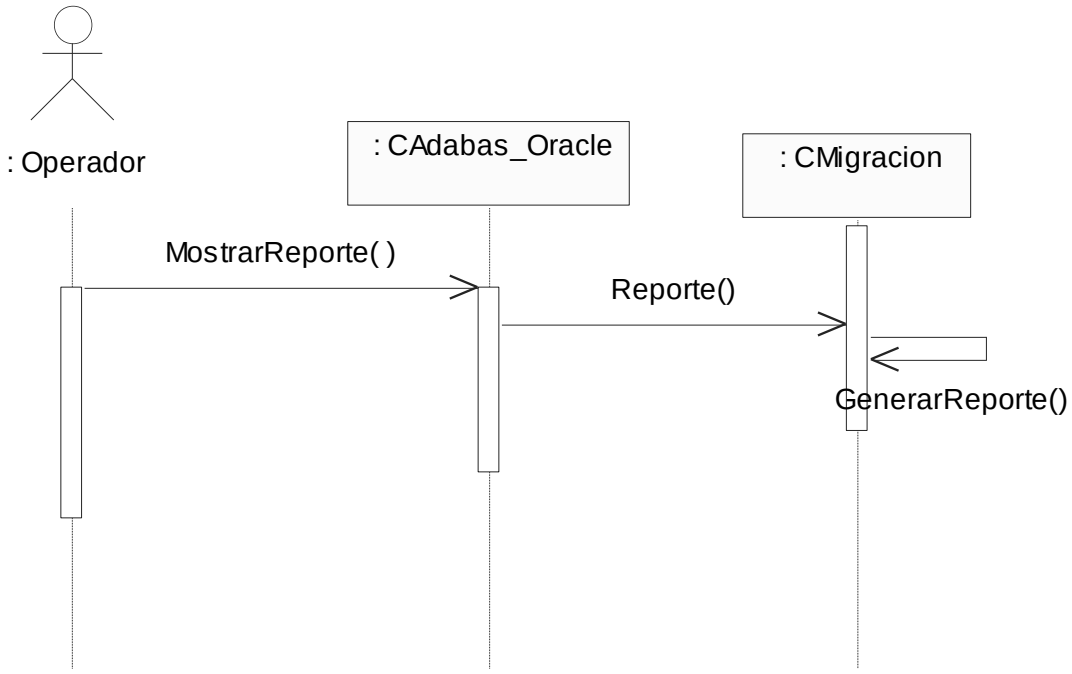

Figura 16. Diagrama de Secuencia CU Mostrar reporte.

3.5.6 Diagrama de secuencia CU Migrar datos Manualmente.

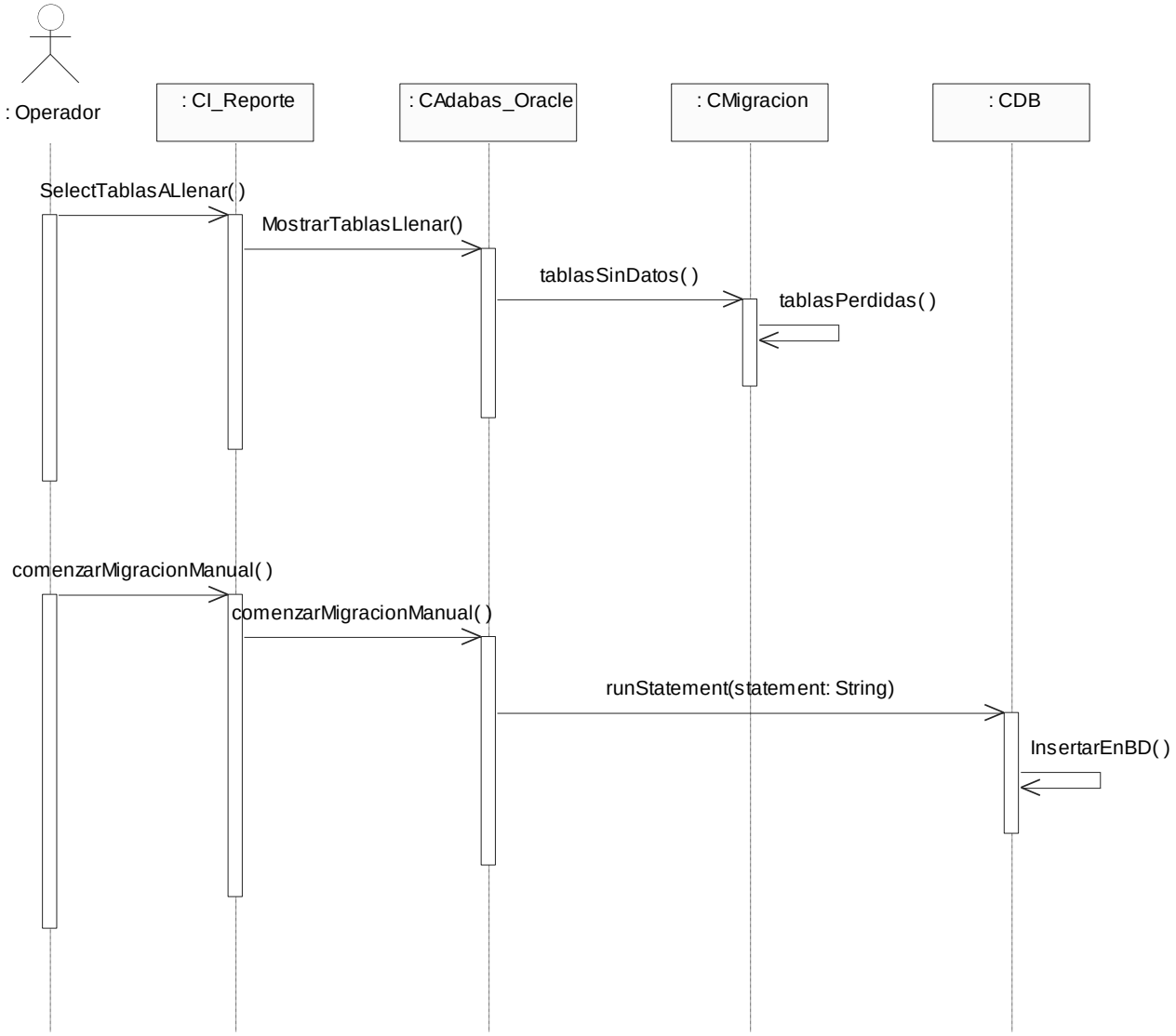

Figura 17. Diagrama de Secuencia CU Migrar datos Manualmente.

3.5.7 Diagrama de secuencia CU Completar migración.

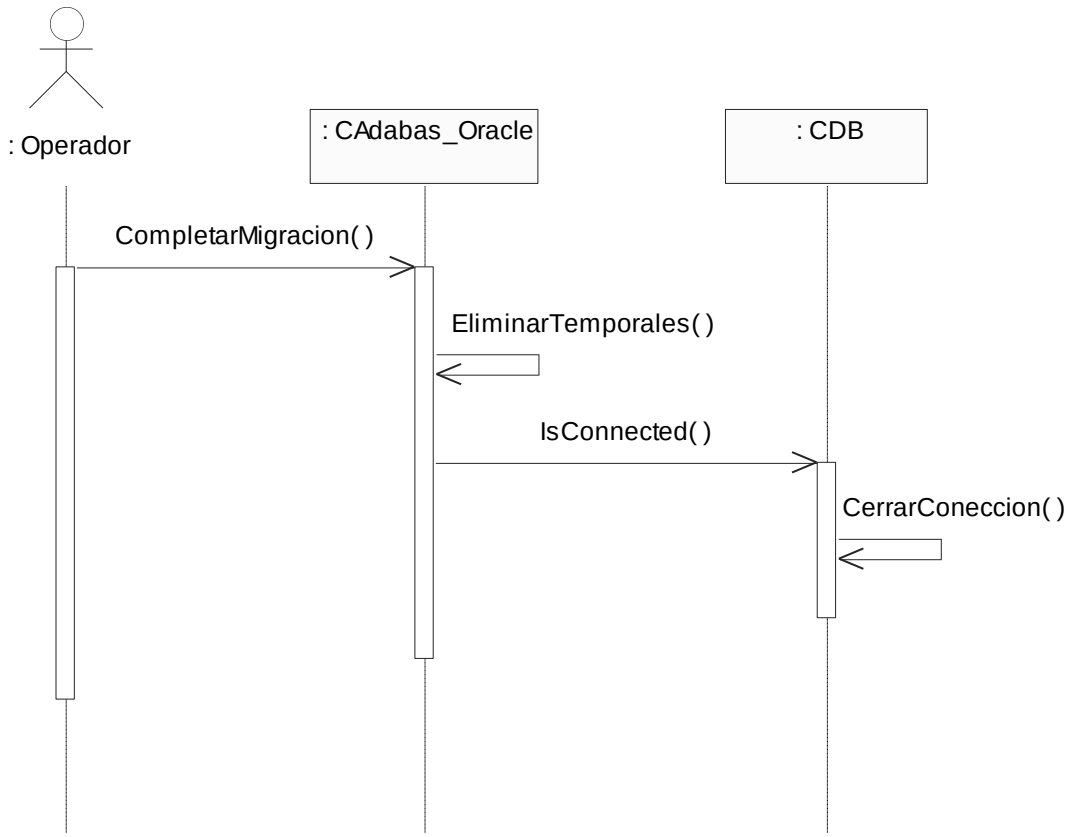

Figura 18. Diagrama de Secuencia CU Completar Migración.

3.6 Diagrama de despliegue

El diagrama de despliegue permite apreciar de forma visual como se encuentran relacionados físicamente los componentes de la aplicación. En este caso se cuenta con una PC cliente conectada mediante el protocolo TCP/IP al Servidor de Base de Datos en que se encuentra instalado Oracle 10g como sistema gestor.

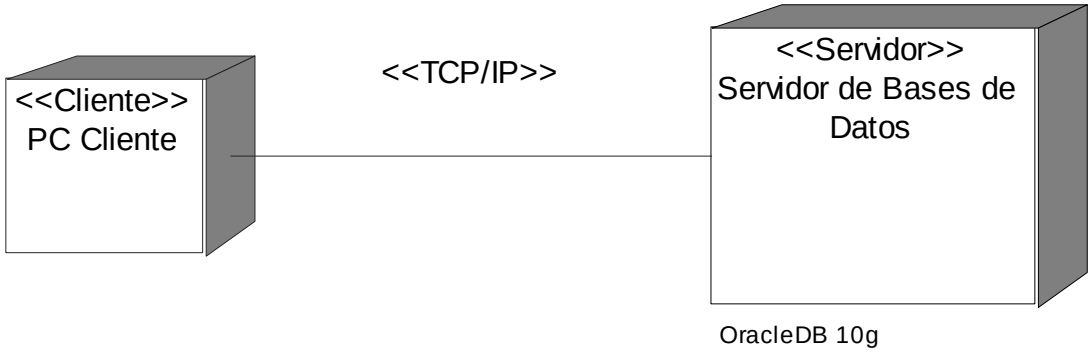

Figura 19 Diagrama de Despliegue

3.7 Conclusiones

En el presente capítulo se obtuvo la solución al software propuesto, describiendo la arquitectura utilizada para el desarrollo del mismo. Se mostraron los resultados de la etapa de diseño, para ello se mostraron los diagramas de clases por cada caso de uso lo cual brindó una mejor comprensión del sistema. Además se realizó el diseño del diagrama de despliegue donde se describen los nodos de procesamiento en tiempo real donde se ejecutará dicho software y los vínculos entre ellos. Todos estos elementos obtenidos son claves para la correcta implementación del sistema propuesto.

Capítulo

4

4 CAPÍTULO 4: IMPLEMENTACIÓN Y PRUEBA

4.1 Introducción

Desde que el implementador empieza a desarrollar una aplicación, se va encontrando a su paso con posibles errores que de ocurrir pudieran afectar el rendimiento de la aplicación, una forma de ver si la aplicación está libre de estos errores es testeando el código o las interfaces. El tratamiento de errores también forma parte del proceso de producción, la buena ejecución del tratamiento de errores conduce a un software que en la fase de pruebas generará menos resultados insatisfactorios.

Este capítulo, tratará fundamentalmente sobre los tipos de pruebas que es necesario realizar para que una aplicación sea probada ya sea en tiempo de compilación (*cuando se está compilando el programa*) o en tiempo de ejecución (*cuando se está ejecutando la aplicación*).

4.2 Implementación

En la implementación se comenzó con el resultado del diseño y se implementó el sistema en términos de componentes, es decir, ficheros de código fuente, scripts, fichero de código binario, ejecutables y similares. El propósito de la implementación es desarrollar la arquitectura y el sistema como un todo (RUMBAUGH, J. 2000)

4.2.1 Modelo de implementación

El Modelo de Implementación es comprendido por un conjunto de componentes y subsistemas que constituyen la composición física de la implementación del sistema. Entre los componentes podemos encontrar datos, archivos, ejecutables, código fuente y los directorios. Fundamentalmente, se describe la relación que existe desde los paquetes y clases del modelo de diseño a subsistemas y componentes físicos.

4.2.1.1 Diagramas de Componentes

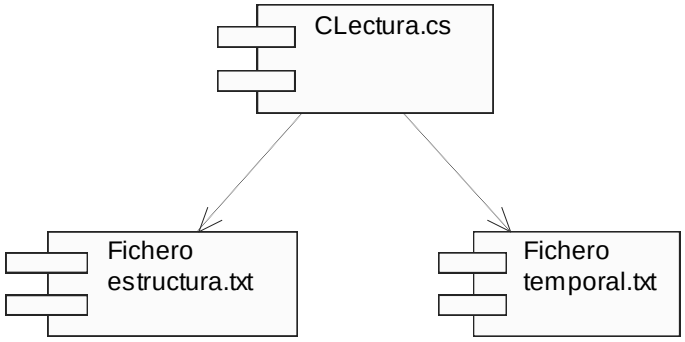

Figura 20 Diagrama de componentes CU Cargar Lectura

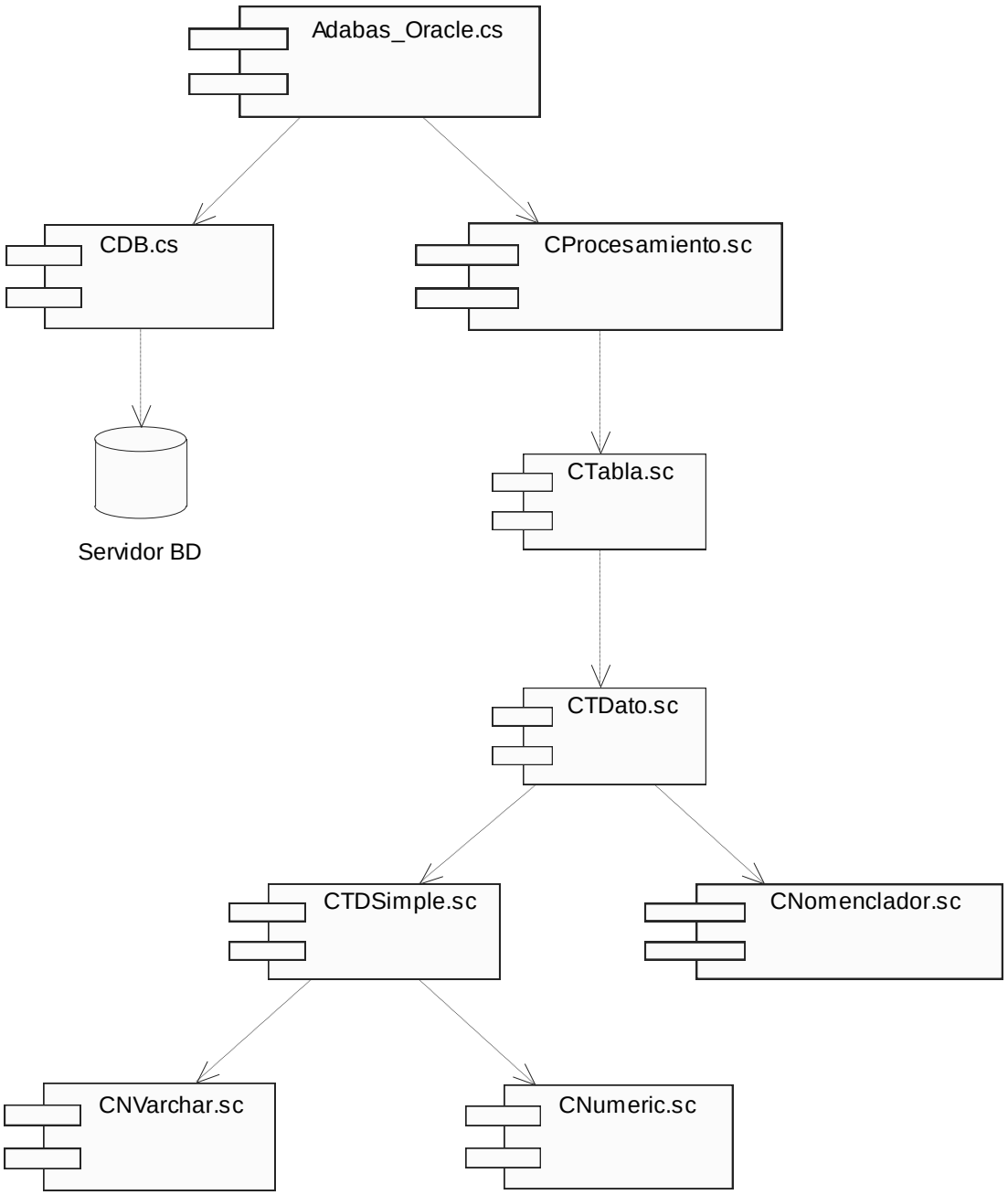

Figura 21 Diagrama de componentes CU Preparar Base de datos

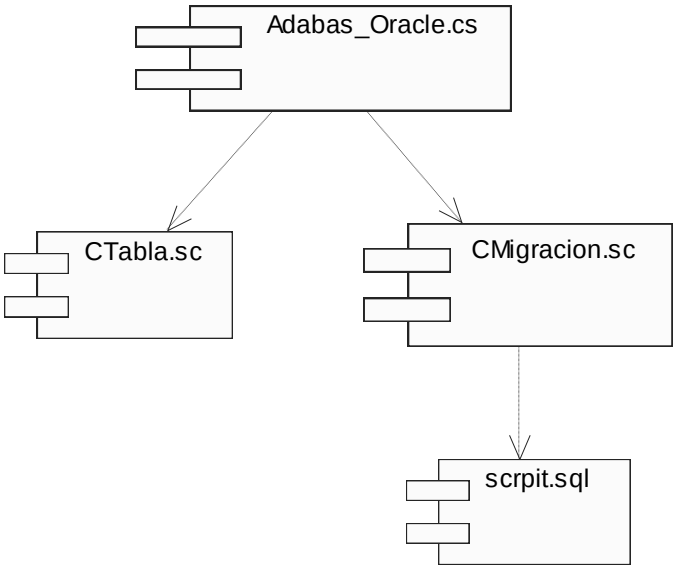

Figura 22 Diagrama de componentes CU Generar Script de Migración

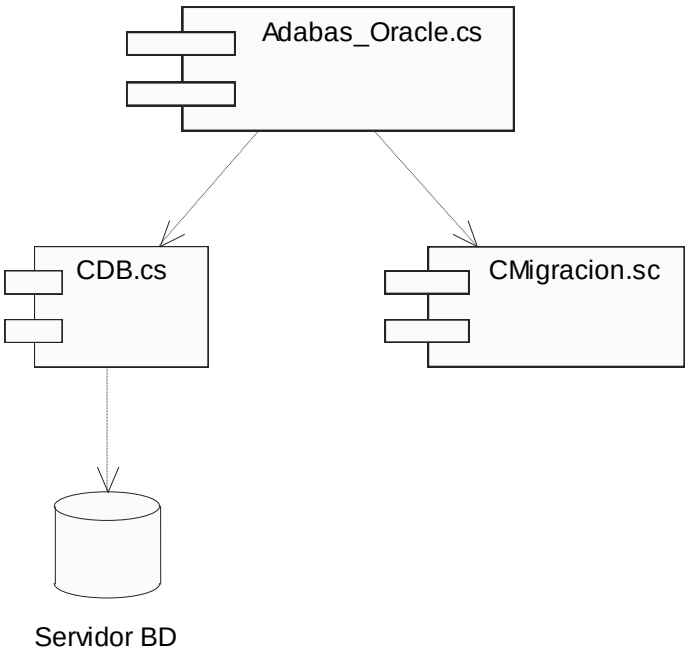

Figura 23 Diagrama de componentes CU Migrar datos automáticamente

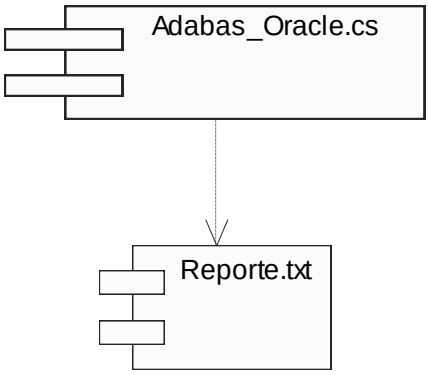

Figura 24 Diagrama de componentes CU Mostrar Reporte

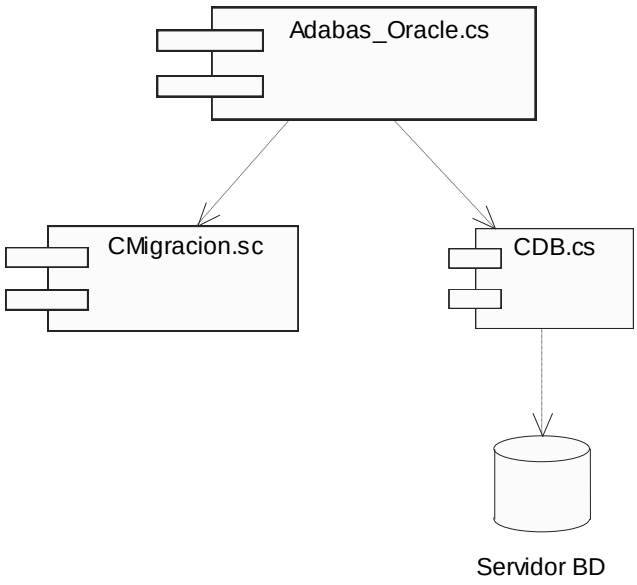

Figura 25 Diagrama de componentes CU Migrar datos manualmente

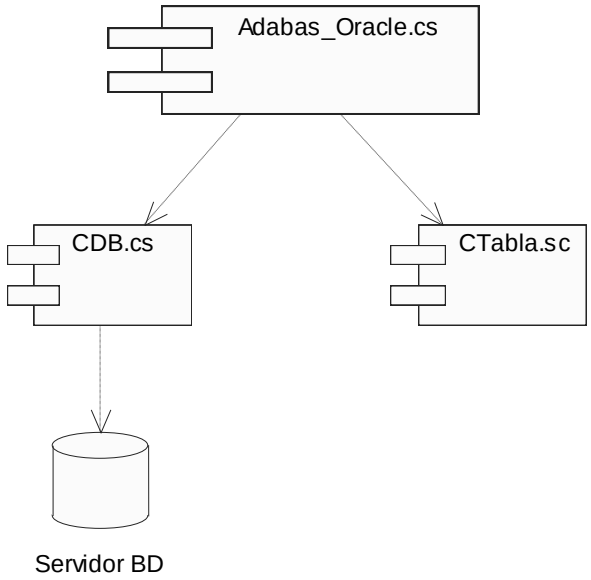

Figura 26 Diagrama de componentes CU Completar Migración

4.3 Prueba

La prueba es un proceso de ejecución de un programa con la intención de descubrir errores. El objetivo de la prueba es diseñar pruebas que saquen a la luz diferentes clases de errores con la menor cantidad de tiempo y espacio. (PRESSMAN 2005)

4.3.1 Tratamiento de errores

Para que la aplicación funcionara adecuadamente frente a cualquier situación que se presentase, ya sea por errores inducidos por el usuario, por el propio sistema o debido a factores externos, fue necesario tener un completo control sobre todos los posibles errores a ocurrir en el sistema, de tal forma que si ocurriese alguno se pudiera manejar y tomar las acciones pertinentes para que no comprometan la integridad del sistema ni la integridad de los datos.

La administración de excepciones incluye la detección y generación de excepciones, el diseño de éstas, el flujo de información de las mismas y la publicación de información de las excepciones a los usuarios. Las excepciones proporcionan un flujo de información ascendente. Esto adquiere particular

relevancia cuando se alcanza una interfaz de servicios o de usuario con la que no desea establecer un flujo de la excepción literalmente, sino más bien traducirla a algo que el cliente pueda procesar, y sin exponer información confidencial empresarial o técnica acerca de su aplicación o servicio (*como por ejemplo, una cadena de conexión a una base de datos en caso de un error de conexión*) que se pueda utilizar contra el sistema o la organización. Es habitual derivar a dos ramas principales de excepciones: excepciones empresariales y técnicas. Este diseño facilita la detección y publicación del tipo de excepciones.

4.3.2 Administración de excepciones en la interfaz de usuario

Las excepciones son lanzadas en cascada desde la capa de acceso a datos hacia la capa del negocio y finalmente hacia la capa de presentación, donde se mostrarán finalmente y las visualizará. Estas excepciones ya vendrán transformadas de forma que el usuario pueda entenderlas y le facilite la tarea de ubicar el error sin brindar información que ponga en peligro nuestra aplicación.

4.3.3 Modelo de Prueba

El desarrollo del software implica una serie de actividades de producción en las que las posibilidades de que aparezca la falibilidad humana son comunes. Debido a la imposibilidad humana de trabajar y comunicarse de forma perfecta, el desarrollo del software ha de ir acompañado de una actividad que garantice la calidad. La prueba de software es un elemento crítico para la garantía de la calidad del software y representa una revisión final de las especificaciones del diseño y de la codificación.

Una de las últimas fases del ciclo de vida antes de entregar un programa para su explotación, es el Flujo de Trabajo de Pruebas, aunque estas se realicen durante todo el desarrollo del software. Cuando ya se cuenta con un producto con capacidad operativa, que haga cumplir los requerimientos funcionales es necesario pasar al proceso de ejercitar el mismo con la intención específica de encontrar errores previos a la entrega al usuario final. Las pruebas son ejecutadas bajo unas condiciones o requerimientos específicos, los resultados son observados y registrados, y se hace una evaluación y rectificación de algunos aspectos de sistema o componente. Tienen varios niveles, Nivel de Unidad, Nivel de Integración, Nivel de Sistema y Nivel de Aceptación.

4.3.4 Descripción de las pruebas

Los “test de unidades” son pruebas llevadas a cabo por los implementadores sobre las unidades mínimas desarrolladas por ellos, estas unidades pueden ser clases, métodos, propiedades, componentes, etcétera. Estas unidades se prueban separadas unas de otras y básicamente se hacen durante la implementación del software. Para realizar estas pruebas es necesario establecer una serie de reglas que sirven como objetivos de estas pruebas:

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

El objetivo de diseñar pruebas se basa en que sistemáticamente le muestren al implementador diferentes tipos de errores, haciéndolo con la menor cantidad de tiempo y esfuerzo.
Los “test de unidades” son orientados casi siempre a las pruebas de “caja blanca” aunque para realizar uno de estos test es necesario probar el flujo de datos desde la interfaz del componente. Si los datos no entran correctamente, todas las demás pruebas no tienen sentido. Estas pruebas son aplicadas a componentes representados en el modelo de implementación para verificar que los flujos de control y de datos están cubiertos, y que estos funcionen como se espera.

Las “pruebas de caja blanca” es prácticamente el minucioso examen de los detalles procedimentales. Se comprueban los caminos lógicos del software proponiendo casos de pruebas que examinan que estén correctos todas las condiciones y/o bucles para determinar si el estado real coincide con el esperado o afirmado. Estos casos de pruebas generan los caminos lógicos o posibles que debe recorrer el flujo de la ejecución, para afirmar que se está ejecutando correctamente el procedimiento.

Una de las técnicas para ejecutar las pruebas de caja blanca es la del “Camino básico”, el resultado de esta técnica es una medición cuantitativa de la complejidad lógica de un programa. El valor calculado como complejidad ciclomática, define el número de caminos independientes del conjunto básico de un programa y da un límite superior para el número de pruebas que se deben realizar para asegurar que se ejecute cada sentencia al menos una vez. Otra forma de probar el código es mediante las pruebas de “caja negra”. Estas pruebas permiten obtener un conjunto de condiciones de entrada que ejerciten

completamente todos los requisitos funcionales de un programa. En ellas se ignora la estructura de control, concentrándose en los requisitos funcionales del sistema y ejercitándolos.

Para fijar estas pautas de diseño de pruebas, se apoya en las siguientes dos definiciones de un caso de prueba bien elegido:

- El que reduce el número de otros casos necesarios para que la prueba sea razonable. Esto implica que el caso ejecute el máximo número de posibilidades de entradas diferentes para así reducir el total de casos.
- Cubre un conjunto extenso de otros casos posibles, es decir, indica algo acerca de la ausencia o la presencia de defectos en el conjunto específico de entradas que prueba, así como de otros conjuntos similares.

Una de las técnicas más usadas dentro de las pruebas de caja negra es la “Partición de Equivalencia” esta técnica divide el campo de entrada en clases de datos que tienden a ejercitar determinadas funciones del software. Es también una de las más efectivas, pues permite examinar los valores válidos y no válidos de las entradas existentes en el software.

Mediante esta técnica se descubren de forma inmediata una serie de errores que de otro modo, requerirían la ejecución de muchos casos antes de detectar el error genérico, además se dirige a la definición de casos de pruebas que descubran clases de errores, reduciendo así el número de clases de prueba que hay que desarrollar.

Para definir las clases de equivalencia es necesario tener en cuenta un conjunto de reglas:

- Si una condición de entrada especifica un rango, entonces se confeccionan una clase de equivalencia válida y 2 no válidas.
- Si una condición de entrada especifica la cantidad de valores, identificar una clase de equivalencia válida y dos no válidas.
- Si una condición de entrada especifica un conjunto de valores de entrada y existen razones para creer que el programa trata en forma diferente a cada uno de ellos, identificar una clase válida para cada uno de ellos y una clase no válida.
- Si una condición de entrada especifica una situación de tipo “debe ser”, identificar una clase válida y una no válida.
- Si existe una razón para creer que el programa no trata de forma idéntica ciertos elementos pertenecientes a una clase, dividirla en clases de equivalencia menores.

Luego de tener las clases válidas y no válidas definidas, se procede a definir los casos de pruebas, pero para ello antes se debe haber asignado un identificador único a cada clase de equivalencia. Luego entonces se pueden definir los casos teniendo en cuenta lo siguiente:

- Escribir un nuevo caso que cubra tantas clases de equivalencias válidas no cubiertas como sea posible hasta que todas las clases de equivalencias hayan sido cubiertas por casos de prueba.
- Escribir un nuevo caso de prueba que cubra una y sólo una clase de equivalencia no válida hasta que todas las clases de equivalencias no válidas hayan sido cubiertas por casos de pruebas.

Con la aplicación de esa técnica se obtiene un conjunto de pruebas que reduce el número de casos de pruebas y dice algo sobre la presencia o ausencia de errores. A menudo se plantea que las pruebas a los software nunca terminan, simplemente se transfiere del desarrollador al cliente.
Cada vez que el cliente usa el programa está llevando a cabo una prueba. Aplicando el diseño de casos de pruebas al software en cuestión se puede conseguir una prueba más completa y descubrir y corregir el mayor número de errores antes de que comiencen las “pruebas del cliente”.

4.4 Conclusiones

El proceso de creación del presente trabajo ha atravesado diferentes etapas dando cumplimiento a los objetivos del mismo. Este proceso constó de una etapa inicial de investigación, en la cual se realizó una búsqueda de información acerca de los sistemas de migración existentes en el mundo, así como sus características, además de una breve investigación de las bases tecnológicas para resolver el problema planteado. Posteriormente se pasó a una etapa de análisis y diseño en la cual se obtuvieron todos los artefactos necesarios para llegar a la solución esperada. Ya en la próxima fase se comenzó con la implementación de todos los módulos, verificando el cumplimiento de todos los requisitos funcionales y no funcionales de la aplicación a realizar. Una vez terminada la implementación se dio inicio a la realización de las pruebas en vista de verificar la calidad y cumplimiento de cada una de las funcionalidades.
Con la culminación y puesta en funcionamiento se espera obtener resultados satisfactorios en la migración a realizar en el CICPC.

CONCLUSIONES GENERALES

Con esta investigación, guiados por el proceso de desarrollo de software RUP, se realizó un software de migración de estructura y datos desde ADABAS a Oracle con fines de automatizar un proceso fundamental en el actual proyecto con el CICPC, contribuirá al traspaso de datos de forma segura y con buenos resultados de migración.

Se concluye que:

- Se procesó toda la información relacionada con las migraciones de datos, integridad de los mismos y características del software de este tipo existente en el mundo.
- Se realizó el análisis, diseño e implementación del software de migración de ADABAS a Oracle, el cual fue probado por el grupo de calidad del proyecto CICPC de la facultad, quedando satisfechos con el resultado. La aplicación será utilizada para el futuro despliegue del proyecto.
- Se creó un documento donde se recogió todo el proceso investigativo del desarrollo del software, la modelación de los artefactos obtenidos durante su ciclo de vida, así como la propuesta de solución y los resultados respecto a esta.
- El software desarrollado le brinda al operador la oportunidad de interactuar con un ambiente informatizado que garantiza integridad y seguridad, empleando las nuevas Tecnologías de la Información y las Comunicaciones y evitando así todo un tedioso trabajo manual.

Con la puesta en funcionamiento del software se logrará una migración de datos de grandes magnitudes desde la BD existente actualmente en el CICPC, elemento que aumentará el nivel de respuesta y acción, dando lugar así a la prestación de un mejor servicio.

RECOMENDACIONES

Ya terminado este trabajo teniendo en cuenta que se han cumplido todos los objetivos planteados, se recomienda:

- Continuar trabajando en la mejora de los resultados de la aplicación, para de esta forma perfeccionar aún más los servicios de migración de datos que presta la misma.
- Perfeccionar los métodos implementados para ganar en rapidez, seguridad y fiabilidad.
- Integrar los datos obtenidos del cliente con la BD en desarrollo, para así ganar en organización, velocidad de procesamiento y garantizar su correcta manipulación.

REFERENCIAS BIBLIOGRÁFICAS

1) **ARCHER, T. 2001.** *C# A Fondo.* s.l. : McGRAW-HILLlINTERAMERICANA DE ESPmA, S. A. U, 2001.
2) **BARRIENTOS ENRÍQUEZ, A. M. 2005.** *El proceso Unificado de Modelado (RUP).* 2005.
3) **CORPORATION, Case Studio.** [Online] http://www.freedownloadscenter.com/es/Programacion/Base_de_Datos_y_Redes/CASE_Studio_2.html ; http://www.casestudio.com/ .
4) **CORPORATION, Microsoft. 2008.** Introducción al Visual Studio 2007. [Online] 2008. http://msdn.microsoft.com/en-us/library/fx6bk1f4(VS.80).aspx .
5) **CORPORATION, Oracle. 2008.** Oracle. [Online] 2008. http://www.oracle.com/global/es/index.html .
6) **CORPORATION, Rational Rose.** Rational. [Online] http://www.rational.com.ar/herramientas/roseenterprise.html .
7) **CORPORATION, Sharp Develop.** [Online] http://www.bloginformatico.com/sharpdevelop-fiel-entorno-de-desarrollo-de-programacion.php ; http://www.icsharpcode.net/OpenSource/SD/ .
8) **CORPORATION, Visual Paradigm.** Visual Paradigm. [Online] http://www.visual-paradigm.com/product/vpuml/ .
9) **Developer, PL/SQL.** [Online] http://www.software-shop.com/in.php?mod=ver_producto&prdID=159 .
10) **ER/Studio.** [Online] http://www.financialtech-mag.com/000_estructura/index.php?id=24&idb=57&ntt=3777&sec=12&vn=1 .
11) **ERROR500.** Sistema Gestor de Base de Datos. [Online] Garbage Collector. http://www.error500.net/garbagecollector/bases_de_datos/sistema_gestor_de_base_de_dato.php .
12) **HERNANDEZ ORALLO, E. 2001.** El lenguaje Unificado de Modelado (UML). [Online] 2001. http://www.disca.upv.es/enheror/pdf/ActaUML.PDF .
13) **JACOBSON, I. B and GRADY, RUMBAUGH, JAMES. 2000.** El proceso unificado de desarrollo de software. [Online] 2000. http://bibliodoc.uci.cu/pdf/reg00060.pdf .
14) **Java.** [Online] http://www.manual-java.com/manualjava/caracteristicas-java.html .
15) **LARMAN, C. 1999.** *UML y Patrones Introducción al análisis y diseño orientado a objetos.* Mex : Prentice Hall, 1999.
16) **MARQUÉS, ANDRÉS. 2001.** Apuntes de Ficheros y Bases de Datos. [Online] Universitat Jaume I, 2001. http://www3.uji.es/~mmarques/f47/apun/node40.html .
17) **MARTEENS, IAN. 2006.** *La cara oculta del Delphi.* s.l. : Alfaomega, 2006. ISBN 840-607-38-7364.
18) **MOLPECEES, A. 2003.** Procesos de desarrollo: RUP, XP y FDD. [Online] 2003. http://www.javahispano.org/articles.article.action?id=76 .

19) **NaturalAdabas. 2006.** [Online] 2006. http://cerocoma.blogspot.com/2006/05/naturaladabas.html .

20) **PÉREZ, C.** *Oracle PL-SQL.* ISBN 978-970-15-1374-3.

21) **PRESSMAN, R. S. 2005.** *Ingeniería del Software, Un enfoque práctico.* s.l. : Mc Graw Hill, 2005.

22) **ROSSI, G. A. M. 2004.** UML: El lenguaje estándar para el modelado del software. [Online] 2004. http://www.ati.es/novatica/2004/168/168-4.pdf .

23) **RUMBAUGH, J. 2000.** *El lenguaje Unificado de modelado. Manual de Referencia.* s.l. : Adison Wesley, 2000. ISBN 84-7829-7037-7820.

24) **SECO, J. 2001.** El lenguaje de programación C#. [Online] 2001. http://www.josanguapo.com/ .

25) **SILBERSCHATZ, A. KORTH, H. F. 2006.** *Fundamentos de Bases de Datos.* s.l. : McGraw Hill, 2006.

26) **STROUPSTRUP, B. 1998.** *El lenguaje de programación C++.* s.l. : Addison Wesley, 1998. ISBN 84-7829-019-2.

27) **WIKIMEDIA, F. 2007.** Caso de Uso. [Online] 2007. http://es.wikipedia.org/wiki/Caso_de_uso .

28) **WIKIPEDIA, F. 2008.** ADABAS. [Online] 2008. http://es.wikipedia.org/wiki/Adabas .

GLOSARIO DE TÉRMINOS

ADABAS (Adaptable Database System): sistema de gestión de BD propietario creado en 1969.

Automatización: Realización de una combinación específica de acciones por una máquina, sin la ayuda de personas

BD: Base de Datos.

CASE: Ingeniería del software asistida por computadora.

CICPC: Cuerpo de Investigaciones Científicas, Penales y Criminalísticas

C++: Lenguaje híbrido, que se puede compilar y resulta más sencillo de aprender para los programadores que ya conocen C. Las principales características son abstracción (*encapsulación*), el soporte para programación orientada a objetos (*polimorfismo*) y el soporte de plantillas o programación genérica (*templates*). Es un lenguaje que abarca tres paradigmas de la programación: La programación estructurada, la programación genérica y la programación orientada a objetos.

C#: Lenguaje de programación orientado a objetos, evolución del lenguaje C++, desarrollado por Microsoft.

Data warehouse: Almacén de datos orientado a un determinado ámbito (*empresa, organización, etcétera.*), integrado, no volátil y variable en el tiempo, que ayuda a la toma de decisiones en la entidad en la que se utiliza. Expediente completo de una organización, más allá de la información transaccional y operacional, almacenado en una base de datos diseñada para favorecer el análisis y la divulgación eficiente de datos.

Embebido: El concepto de embebido (*embedded*) significa que es algo que está integrado dentro de un conjunto y sirve para controlar diversos periféricos.

FDD (Desarrollo Guiado por Funcionalidad): metodología ágil de desarrollo de software creada para proyectos con tiempos de desarrollo relativamente cortos.

Java: Lenguaje de programación diseñado para Internet, utilizado para crear aplicaciones completas o pequeñas aplicaciones (*applets*) para ser insertados en una página Web.

Migración de datos: Conversión y traspaso de datos desde un sistema gestor de base de datos a otro.

Multiplataforma: Poder funcionar o mantener una interoperabilidad de forma similar en diferentes sistemas operativos o plataformas.

Oracle: Es un sistema de administración de base de datos, es una potente herramienta cliente/servidor para la gestión de bases de datos.

Rational Rose: Herramienta CASE desarrollada por los creadores de UML (Booch, Rumbaugh y Jacobson), que cubre todo el ciclo de vida de un proyecto.

Software: Conjunto de instrucciones escritas en un determinado lenguaje, que dirigen a un ordenador para la ejecución de una serie de operaciones, con el objetivo de resolver un problema que se ha definido previamente.

Scripts: Los scripts comparten las mismas características que los esquemas como estructuras de procesamiento, siendo la organización secuencial o temporal de su estructura el elemento que los define como un tipo específico de esquema.

SGBD (Sistema Gestor de base de datos): es un tipo de software muy específico o conjuntos de ellos que permite manejar de manera clara, sencilla y ordenada altos volúmenes de datos.

SQL (Structured Query Language): Lenguaje declarativo de acceso a bases de datos relacionales que permiten especificar diversos tipos de operaciones sobre las mismas.

XP (Programación Extrema): metodología destacada de los procesos ágiles de desarrollo de software.