FACULTAD # 8

ANÁLISIS, DISEÑO E IMPLEMENTACIÓN DEL SISTEMA DE ESTUDIO DE ADVERSARIO EN EL BOXEO

TRABAJO DE DIPLOMA PARA OPTAR POR EL TÍTULO DE INGENIERO EN CIENCIAS INFORMÁTICAS

AUTORAS:

Elizabeht Arencibia Veloz

Adisbel González Requejo

TUTORAS:

Ing. Aliuska Sánchez Ibarria

Ing. Karenia Donatien Goliath

CONSULTANTE:

Dr.C: Alcides Sagarra Carón

Ciudad de La Habana, junio del 2009

"Año del 50 Aniversario del Triunfo de la Revolución"

DECLARACIÓN DE AUTORÍA

Por este medio se declara como autoras del trabajo de diploma: Análisis, diseño e implementación del Sistema de Estudio de Adversario en el Boxeo a Elizabeht Arencibia Veloz y a Adisbel González Requejo y se le reconocen a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

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

_____	_____
Firma de la autora	Firma de la autora
Elizabeht Arencibia Veloz	Adisbel González Requejo
_____	_____
Firma de la tutora	Firma de la tutora
Ing. Aliuska Sánchez Ibarria	Ing. Karenia Donatien Goliath

DATOS DE CONTACTO

Tutora: Aliuska Sánchez Ibarria

Formación académica: Ingeniero en Ciencias Informáticas (19/JULIO 2007).

Centro laboral: Universidad de las Ciencias Informáticas (UCI)

Profesión: Profesor

Correo electrónico: asanchezi@uci.cu

Tutora: Karenia Donatien Goliath

Formación académica: Ingeniero en Ciencias Informáticas (19/JULIO 2007).

Centro laboral: Universidad de las Ciencias Informáticas (UCI)

Profesión: Profesor

Correo electrónico: kdonatien@uci.cu

Consultante: Alcides Sagarra Carón

Formación académica: Doctor en Ciencias Pedagógicas.

Centro laboral: Escuela Nacional de Boxeo de Cuba.

Profesión: Profesor Titular. Vicepresidente de la Federación Cubana de Boxeo. Jefe de la Comisión Científica para la Asociación Internacional de Boxeo Amateur. Miembro de la Comisión Nacional de Deporte, Cultura y Educación, por la Central de Trabajadores de Cuba.

Correo electrónico: asagarrac@uci.cu, asagarrac@inder.cu

"El secreto de un éxito, es dedicarse por entero a un fin."

José Martí

AGRADECIMIENTOS

A la Revolución, por habernos dado la oportunidad de estudiar y graduarnos en la Universidad de las Ciencias Informáticas.

A nuestros profesores, por habernos educado con tanta dedicación.

A todas las personas que de una forma u otra han contribuido al desarrollo de este trabajo, brindándonos su apoyo incondicional, en especial a Karenia, Aliuska, Edier, Betty, Alison, Mercedes, Leonardo, Misael, Carlos Ismel , Raúl, Yunieski, Dayana, Maikel, Eduardo, Hannybal y Héctor.

A nuestros familiares, por el cariño y la confianza que han depositado en nosotras.

A nuestros amigos, porque siempre han sido especiales con nosotras y han sabido compartir nuestras alegrías y tristezas.

A Rabelo y a Yuandry, por el cariño y el apoyo que nos han dado de manera incondicional.

Elizabeht y Adisbel.

DEDICATORIA

A mi familia por el apoyo y la confianza que me han dado, especialmente a mi papá y a mis abuelos: Isabel y Filo, por haberme educado con tanto cariño y haberme sabido guiar siempre por el camino correcto.

A mi amigo y novio Yuandry, por la ayuda, el apoyo y el cariño que siempre me ha dado, además, por haberme enseñado a ser independiente en los estudios.

A la familia de Yuandry, que es mía también, porque forman una parte importante de mi vida y siempre han sido especiales conmigo.

A Lourdes y a Vladimir, que son como mis padres.

A mi amiga Eliza, porque es una excelente compañera de tesis, en la que siempre he depositado mi entera confianza, pues es una persona muy sacrificada, emprendedora y cumplidora.

A mis amigas y amigos de la universidad, a los que quiero mucho, especialmente a mis amigas del grupo, que son como mis hermanas.

A mis amigas y amigos de Ciego de Ávila, a los que también quiero muchísimo, especialmente a Yaneisy , Dianelis, Dianeisy, Daima, Annie, Yarilys, Lidianny, Yaili, Yasle y Lari.

A todos mis vecinos en Ciego de Ávila, que son muy buenos conmigo.

Adi.

A Gladys Veloz, mi mamá, por haber dedicado su vida a mi educación, por su certera orientación, por su desvelos y eterna vigía para quitar de mi vida todos los obstáculos que se me presentaron en el camino hasta aquí, por ser tan paciente, por tenerme tanta fe, por su amor incondicional, por todos sus sacrificios y esfuerzos por hacer de mi una persona digna y una profesional competente. A ella que es la persona más importante en mi vida.

A mis abuelos maternos Giraldo Veloz y María Serrano por haber ayudado incondicionalmente a mi mamá en mi crianza y educación, por su amor, su confianza, su cariño y por ser tan especiales conmigo.

A la memoria de mi abuela paterna Caridad Anuy por estar al pendiente de mi vida y su cariño, aún cuando vivía tan lejos de mi.

A toda mi familia materna, que aunque es chica nunca ha dejado que me falte nada y me han colmado de cariño.

A mi tía Olga Lidia Arencibia por su preocupación constante por mi bienestar.

A mi primo Yerandy, por estar ahí cada vez que lo necesité estos 5 años, por darme más que un cariño de primos, el de un hermano.

A todos mis compañeros de grupo, por soportarme, por los buenos y malos momentos acumulados todo este tiempo.

A todas mis amistades de la UCI, que la lista de nombres no cabrá en esta página pero si en mi corazón, pero no quiero dejar de mencionar a: Suyen, Dalaity, Marcia, Yaneisy, Beatriz,Odelkys, Addiel y Yudislandry, por estar ahí siempre que los necesité, por apoyarme aún cuando no lo pedía, por tantas cosas compartidas.

A mi compañera de tesis, por ser tan eficiente con su trabajo, por la confianza que depositó en mi, por haber sido además de colega una buena amiga en todo momento.

A Luis Alberto Rabelo mi novio por su paciencia, por su ayuda incondicional, por estar siempre que le necesito, por valorarme tanto como profesional y como mujer.

Elizabeht.

RESUMEN

El Estudio de Adversario es una estrategia, llevada a cabo por varios cuadros pedagógicos (entrenadores), que laboran en la Escuela Cubana de Boxeo. Tiene como propósito la obtención de información técnico-táctica sobre determinados adversarios, para ser utilizada en la elaboración de planes de entrenamiento, que permitan dar una mejor preparación física y psicológica a los boxeadores cubanos, con vista a que obtengan la victoria en competencias. Para solucionar los problemas que presentan los entrenadores, se desarrolló una aplicación de escritorio, capaz de gestionar toda la información que manipulan durante dicho estudio. Por tanto, permite archivar, modificar, consultar e imprimir los datos correspondientes a los boxeadores, las competencias y los combates, además de mostrar una serie de reportes, que son de interés a los usuarios. También cuenta con una interfaz sencilla que brinda un nivel apropiado de seguridad, integridad y disponibilidad de los datos que almacena, teniendo en cuenta su confidencialidad.

ÍNDICE DE CONTENIDO

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

INTRODUCCIÓN

El Instituto Nacional de Deporte, Educación Física y Recreación (INDER), es la organización cubana encargada de dirigir las actividades físicas, deportivas y de recreación en forma masiva, se sustentan en la utilización de recursos humanos altamente calificados y en la aplicación de la ciencia y la innovación tecnológica. Uno de los objetivos fundamentales de esta organización, es lograr la preparación óptima de sus atletas de alto rendimiento, para que obtengan buenos resultados nacional e internacionalmente. Con éste propósito, utilizan la estrategia de Estudio de Adversarios, pues permite trazar planes de entrenamiento adecuados.

Es interés en esta investigación, realizar un análisis del entorno en que se desenvuelven las personas, que se dedican a la práctica pedagógica del Equipo Nacional de Boxeo, específicamente en la dirección del proceso, relacionado de una forma u otra con la realización del Estudio de Adversarios. De esta manera se identifican los problemas que intervienen durante su desarrollo:

- La carencia de un acuerdo teórico, así como una concepción metodológica entre los entrenadores, referente al proceso del estudio con los posibles adversarios.
- Los documentos donde se recoge la información obtenida del Estudio de Adversarios, no son un medio seguro de almacenamiento, ya que son hojas y en la mayoría de los casos se pierden o se deterioran.
- Resulta engorroso archivar la información obtenida del Estudio de Adversarios, debido a que se realiza de forma manual, mediante la recopilación de hojas o cintas de video.
- El video de un determinado combate, generalmente se pierde por deterioro, ya que la tecnología utilizada para su grabación y conservación es obsoleta.
- No es posible obtener estadísticas precisas del comportamiento en combate de determinado boxeador, ya que generalmente no se tiene acceso a los datos históricos del mismo.
- La pérdida de tiempo en que se incurre durante todo el macro ciclo de preparación, en función de localizar la información necesaria para realizar el Estudio de Adversarios.

Al ser planteadas por los clientes las dificultades antes expuestas, surge el siguiente **problema científico**: ¿Cómo facilitar el proceso de gestión de la información que es manipulada por los entrenadores del Equipo Nacional de Boxeo, para realizar el Estudio de Adversarios?

Por tanto el **objeto de estudio de la investigación** está enfocado al proceso de gestión de la información del Estudio de Adversarios.

El **campo de acción** está dirigido al proceso de gestión de la información confidencial que rige el Estudio de Adversarios, realizado por los Entrenadores del Equipo Nacional de Boxeo.

De esta forma el **objetivo general** que orienta esta investigación consiste en: Desarrollar un sistema informatizado, para gestionar el Estudio de Adversarios por parte de los Entrenadores del Equipo Nacional de Boxeo.

A partir de lo cual se desglosan los siguientes **objetivos específicos**:

1. Diseñar teóricamente la investigación.
2. Seleccionar la metodología, el lenguaje y las herramientas para desarrollar el sistema.
3. Modelar el negocio.
4. Modelar el análisis y diseño del sistema.
5. Implementar el sistema, con las características definidas en el análisis y diseño.

En este sentido las **tareas** que permiten dar solución al problema científico son:

1. Realizar el estudio del estado del arte.
2. Realizar la modelación del negocio, mediante la interacción con los clientes.
3. Realizar la captura de los requisitos funcionales y no funcionales.
4. Realizar las actividades correspondientes al análisis y diseño.
5. Realizar la implementación de las funcionalidades que debe tener el software.

Teniendo en cuenta lo antes expuesto, se manifiesta en el siguiente planteamiento la **idea a defender**: Si se desarrolla un software que automatice los procesos correspondientes al Estudio de Adversarios en el Boxeo, se estará dando solución a los problemas que presentan los entrenadores del Equipo Nacional, durante la realización del mismo.

Para obtener toda la información antes expuesta, el **método de investigación científica** que ha sido utilizado es el empírico, mediante la observación científica, pues ha sido evaluado el problema existente en el INDER, específicamente en el Boxeo, como resultado se ha determinado el objeto de investigación, estudiando su curso natural, ya que la observación que es realizada tiene un comportamiento contemplativo. La **técnica** que se ha utilizado con el objetivo de recolectar la información necesaria para la investigación es la entrevista.

CAPÍTULO 1

FUNDAMENTACIÓN TEÓRICA

1.1 Introducción

La necesidad de desarrollar una propuesta de software, que automatice los procesos fundamentales, correspondientes al estudio de adversario en el boxeo, conlleva a la realización de una investigación en esta área del deporte cubano. En este capítulo, se exponen los resultados obtenidos durante la misma, es decir, los conceptos fundamentales asociados al dominio del problema (que permitirán una mayor comprensión del entorno objeto de estudio, por parte de las personas implicadas en el desarrollo de la aplicación). Además, de un análisis de otras soluciones existentes y de las tendencias actuales en metodologías y tecnologías, para el desarrollo de sistemas informáticos, específicamente en aplicaciones de escritorio, realizando una selección de aquellas que serán utilizadas durante el desarrollo del proyecto.

1.2 Conceptos asociados al dominio del problema

INDER: El Instituto Nacional de Deporte, Educación Física y Recreación: es una organización cubana encargada de dirigir las actividades físicas, deportivas y de recreación en forma masiva, se sustentan en la utilización de recursos humanos altamente calificados y en la aplicación de la ciencia y la innovación tecnológica.

Escuela Cubana de Boxeo: Se inscribe en el sistema deportivo cubano, la cual es producto de los esfuerzos del Instituto Nacional de Deporte, Educación Física y Recreación. Comprende un sistema de conocimientos y destrezas tecnológicas, que suelen diferenciarse en cuanto al grado de interdependencia motriz, en función de una orientación hacia rendimiento deportivo. (SAGARRA *et al.* 2007)

Estudio de Adversario: Es una estrategia que utilizan los entrenadores de la Escuela Cubana de Boxeo, para obtener información técnico-táctica sobre los adversarios de interés. Tiene como objetivo lograr una mejor preparación física y psicológica de los boxeadores cubanos con vista a obtener la victoria competitiva.

1.3 Análisis de otras soluciones existentes

Utilius es un sistema de edición de vídeo para todas las aplicaciones en el deporte. Fue creado por *Campus-Computer-Center GmbH* de *Leipzig* asistido por el *IAT Leipzig*. El programa es usado en todo el mundo por centros de entrenamiento olímpico, selecciones nacionales, universidades y equipos de alto nivel para archivar y editar tomas de vídeo análogo o digital. En la actualidad se emplea en el hockey, balonmano, voleibol, fútbol, en tenis y tenis de mesa, en gimnasia en aparatos y gimnasia rítmica, en golf, atletismo y equitación, aunque no se ha constatado su utilización en el boxeo. Existe en alemán, inglés, español, japonés y en chino. Una de sus funcionalidades, es permitir la posibilidad de realizar el Estudio de Adversarios. (ZUAZO 2003)

Entre sus características fundamentales se puede citar que permite:

- Capturar vídeo directo al ordenador.
- Marcar y clasificar fácilmente escenas y acciones.
- Moverse rápidamente dentro del vídeo y de las escenas marcadas.
- Presentar una acción o una selección de acontecimientos a cualquier velocidad, en múltiples ventanas o en pantalla completa, siempre en perfecta calidad.
- Crear un nuevo vídeo de escenas seleccionadas, editarlas fácilmente en una línea de tiempo.
- Dibujar líneas y curvas directamente en la imagen.
- Agregar textos y entre títulos.
- Exportar el vídeo a un nuevo archivo, a CD o DVD
- Administrar ediciones, análisis y vídeos en una única base de datos. (ZUAZO 2003)

En la actualidad, no se conoce la existencia de algún software nacional o internacional, que sea utilizado para realizar el Estudio de Adversarios en el Boxeo. No obstante en el ámbito nacional se ha estado trabajando en este sentido, pues se desarrolló en la Universidad de las Ciencias Informáticas el **Sistema Integral de Baseball (SIB)**, para realizar el Estudio de Adversarios en este deporte. La aplicación tiene como propósito brindar una estadística detallada de cada uno de los peloteros y actualmente se encuentra en etapa de pruebas.

1.4 Metodologías de desarrollo de software

Las metodologías de desarrollo de software surgieron a raíz de la necesidad de controlar, guiar y documentar proyectos cada vez más complejos, impulsadas principalmente por instituciones

económicamente importantes y con requisitos de seguridad y fiabilidad en sus sistemas sumamente estrictos. En la actualidad son muchas las metodologías de desarrollo de software que existen, el uso de una u otra dependen de las características de cada proyecto.

Las metodologías de desarrollo de software pueden ser clasificadas en dos tipos: las tradicionales o robustas y las ágiles. Las metodologías robustas se centran especialmente en el control del proceso, estableciendo rigurosamente las actividades involucradas, los artefactos que se deben producir y las herramientas que se usarán. Por otra parte, las metodologías ágiles son las que se centran en otras dimensiones, como por ejemplo el factor humano o el producto software, dando mayor valor al individuo, a la colaboración con el cliente y al desarrollo incremental del software con iteraciones muy cortas.

El **Rational Unified Process (RUP),** es una propuesta de proceso para el desarrollo de software orientado a objeto, que utiliza UML como Lenguaje de Modelado Unificado para describir un sistema, mejora la productividad del equipo de trabajo y entrega las mejores prácticas del software a todos los miembros del mismo. Es, además, una metodología de desarrollo de software que intenta integrar todos los aspectos a tener en cuenta durante el ciclo de vida del software, con el objetivo de abarcar tanto pequeños como grandes proyectos. En general, es una metodología predictiva y orientada a casos de uso.

RUP tiene una estructura bidimensional, dividiendo el proceso en fases, y éstas en flujos de trabajo, lo cual puede observarse en la siguiente figura, donde se exponen las fases, iteraciones y disciplinas.

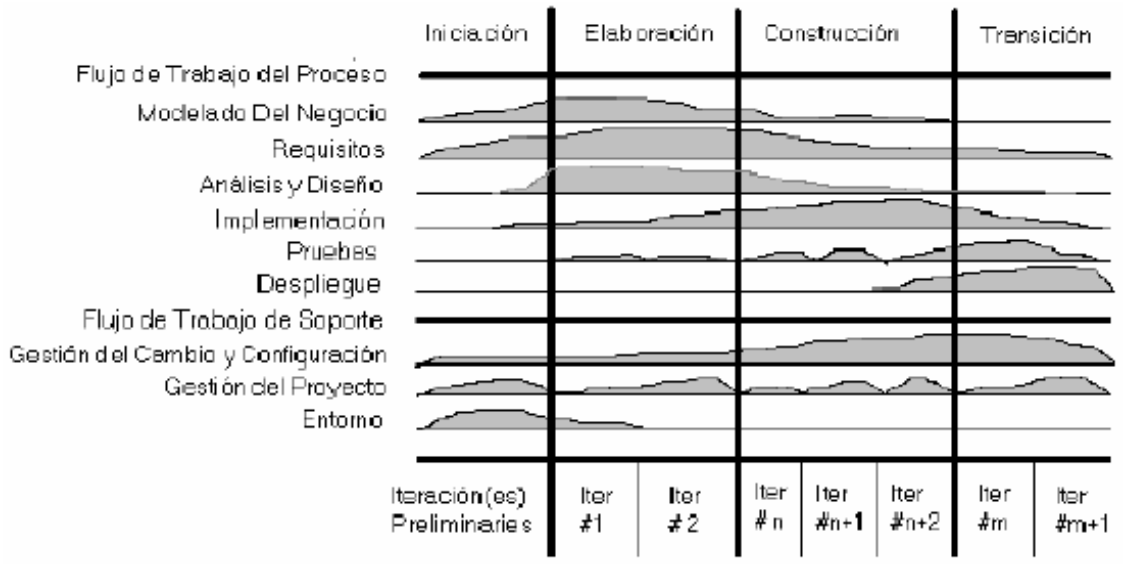


Figura 1. Fases y Flujos de Trabajo de RUP.

RUP divide el desarrollo del software en cuatro fases:

- Inicio o concepción: Determina la visión del proyecto.
- Elaboración: Determina la arquitectura óptima.
- Construcción: Obtiene la capacidad operacional inicial.
- Transición: Obtiene el *release* del proyecto.

Cada una de estas etapas es desarrollada mediante el ciclo de iteraciones, que 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. (RUMBAUGH *et al.* 2000)

El ciclo de vida que se desarrolla por cada iteración, es llevado bajo dos disciplinas o flujos de trabajo:

Disciplina de Desarrollo:

- Ingeniería de Negocios: Entendiendo las necesidades del negocio.
- Requerimientos: Trasladando las necesidades del negocio a un sistema automatizado.
- Análisis y Diseño: Trasladando los requerimientos dentro de la arquitectura de software.
- Implementación: Creando un software que se ajuste a la arquitectura y que tenga el comportamiento deseado.

- Pruebas: Asegurándose que el comportamiento requerido es el correcto y que todo lo solicitado está presente.

Disciplina de Soporte:

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

(RUMBAUGH *et al.* 2000)

Las características distintivas de RUP son las siguientes:

Dirigido por Casos de Uso: Tiene a los casos de uso como el hilo conductor que orienta las actividades de desarrollo. Se centra en la funcionalidad que el sistema debe poseer para satisfacer las necesidades de un usuario (persona, sistema externo, dispositivo) que interactúa con él.

Centrado en la arquitectura: La arquitectura involucra los elementos más significativos del sistema y está influenciada entre otros por plataformas de software, sistemas operativos, manejadores de bases de datos, protocolos; consideraciones de desarrollo como sistemas heredados y requerimientos no funcionales. Una vez definida la arquitectura se puede decir que el sistema tiene "forma".

Iterativo e incremental: Esta característica propone dividir el proceso de desarrollo en partes, cada una de las cuales incluya las fases de: Requerimientos, Análisis, Diseño, Implementación y Pruebas, con el objetivo de acelerar el ritmo de desarrollo para que el producto salga al mercado en el menor tiempo posible y con mayor calidad. (RUMBAUGH *et al.* 2000)

Extreme Programing (XP) pertenece a la 3ra generación de metodologías orientadas a objetos. Es una metodología ágil o adaptativa, que otorga la prioridad a los trabajos que dan un resultado directo y reduce la burocracia alrededor de la programación. Debe ser usado cuando los clientes no tengan una idea clara de los requerimientos que deberá cumplir el software, por lo que a medida que transcurre el tiempo los van cambiando. Tiene como objetivos satisfacer las necesidades del cliente, dándole el software que

necesita, en el momento indicado y potenciar al máximo el trabajo en grupo, donde el cliente, el jefe del proyecto y los desarrolladores son parte del equipo y están involucrados en el desarrollo del software.

XP se basa en cuatro valores o principios fundamentales:

- Comunicación: La comunicación entre los miembros del equipo y los clientes debe ser la máxima posible.
- Simplicidad: Se debe usar la solución más sencilla que pueda funcionar.
- Retroalimentación: Siempre debe ser posible medir el producto que se está desarrollando y conocer qué le falta para satisfacer los requerimientos.
- Coraje: Es necesario armarse de valor para incorporar cambios durante un proyecto. El coraje por sí solo es peligroso, pero sustentado con comunicación, simplicidad y retroalimentación, es una herramienta poderosa.

Selección de la Metodología de desarrollo

Teniendo en cuenta las características fundamentales de XP, se decide que no es la metodología adecuada para desarrollar el software que informatice el Estudio de Adversario en el Boxeo, ya que los clientes tienen bien definidos los requerimientos que deberá cumplir dicho software, además no es posible cumplir con el principio de Comunicación que propone, pues los clientes no tienen la posibilidad de formar parte del equipo de desarrollo.

Rational Unified Process (RUP) es una de las metodologías más destacadas y conocidas. Se considera que es la adecuada para el desarrollo del software que informatice el Estudio de Adversario en el Boxeo, ya que permite obtener resultados más acordes con las previsiones, mediante una mejor planificación, organización y control del trabajo de las personas que intervienen en este proceso. Es orientada a casos de uso, pues establece escenarios y sirve para validar procesos. Mediante el uso de RUP se documentan todas las tareas realizadas en el mismo, aspecto de gran utilidad cuando es necesario que se le hagan modificaciones ha dicho software, pues permite una mayor comprensión del trabajo, por parte de personas ajenas al equipo de desarrollo.

El **Lenguaje Unificado de Modelado (UML),** se usa para especificar, visualizar, construir y documentar artefactos de un sistema de software. Es el sucesor de las notaciones de finales de los años 80 y principios de los 90, unifica las notaciones de Booch, Rumbaught y Jacobson. Captura decisiones y

conocimiento sobre los sistemas que se deben construir. Se usa para entender, diseñar, hojear, configurar, mantener, y controlar la información sobre tales sistemas. Está pensado para usarse con todos los métodos de desarrollo, etapas del ciclo de vida, dominios de aplicación y medios. El lenguaje de modelado pretende unificar la experiencia pasada sobre técnicas de modelado e incorporar las mejores prácticas actuales en un acercamiento estándar. UML incluye conceptos semánticos, notación, y principios generales. Tiene partes estáticas, dinámicas, de entorno y organizativas. Está pensado para ser utilizado en herramientas interactivas de modelado visual que tengan generadores de código, así como generadores de informes. La especificación de UML no define un proceso estándar pero está pensado para ser útil en un proceso de desarrollo iterativo. Pretende dar apoyo a la mayoría de los procesos de desarrollo orientados a objetos. El UML debe adoptar el proceso unificado de desarrollo para modelar las actividades de un proyecto. (RUMBAUGH *et al.* 1999)

1.5 Tendencias y tecnologías actuales

Para dar solución a la problemática anterior, se propone una aplicación de escritorio (*desktop*), teniendo en cuenta los siguientes aspectos que la caracterizan:

- La seguridad: Las aplicaciones *desktop* son más seguras con respecto a las *web*, pues los servidores *web,* al estar conectados a la red, son más propensos a los ataques de virus y *hackers*, cuestiones que pueden comprometer la integridad, disponibilidad y confidencialidad de la información que se gestiona.
- La usabilidad: Las aplicaciones *desktop* tienen mayor usabilidad, pues permiten al usuario trabajar en una computadora, que no se encuentre interconectada con otras, o con determinados dispositivos, si así lo prefieren, evitando de esta manera una inversión en dispositivos de conexión, como por ejemplo, los *módem*, aunque si el usuario prefiere trabajar en red, también brinda facilidades con este propósito.

Las aplicaciones *desktop,* tienen la desventaja de no estar disponibles desde otra estación de trabajo, que no sea aquella en la que se encuentra instalada.

Principales lenguajes de programación

Se define como lenguaje de programación: la técnica estándar de comunicación que permite expresar las instrucciones que han de ser ejecutadas en una computadora. Consiste en un conjunto de reglas sintácticas y semánticas que definen un lenguaje informático.

En la actualidad existen un gran número de lenguajes de programación, que permiten desarrollar una aplicación con las características planteadas anteriormente.

C++ es un lenguaje imperativo orientado a objetos derivado del C. Es en realidad, un súper conjunto de C, que nació para añadirle las cualidades y características de las que carecía. Como su ancestro, sigue muy ligado al hardware subyacente, manteniendo una considerable potencia para programación a bajo nivel, pero se la han añadido elementos, que le permiten también un estilo de programación con alto nivel de abstracción.

Es un lenguaje de nivel intermedio, pudiéndose utilizar tanto para escribir software de bajo nivel, como *drivers* y componentes de sistemas operativos, como para el desarrollo rápido de aplicaciones, según el marco de trabajo con el que se disponga, como VCL de *Borland C++ Builder.*

Una de las características más interesantes del lenguaje, es la sobrecarga de operadores. Lo cual significa que, a los operadores intrínsecos del lenguaje, se les puede redefinir la semántica, por ejemplo: se pueden escribir funciones que en vez de tener un nombre, se asocian a un operador, que debe tener por lo menos un parámetro de tipo clase.

Ventajas

- Es un lenguaje que se adapta a múltiples situaciones.
- Es un lenguaje que prioriza mucho la velocidad de compilación intrínsecamente.
- Soporta la sobrecarga de operadores.

Desventajas

- Es un lenguaje de nivel intermedio, puede escribirse programas de alto nivel con él, pero no es puramente un lenguaje de alto nivel como puede serlo C# o Java.
- Es un lenguaje de programación largo y complejo, por tanto, resulta difícil de aprender.

- El uso de punteros en C++ aporta eficiencia, pero generalmente es fuente de errores de lógica, por tal razón C# y Java (que son lenguajes derivados de él) eliminaron este recurso y solo permiten referencias a objetos.
- No es un lenguaje multiplataforma.

C# es un lenguaje de programación de uso general y sencillo, que cuenta con seguridad de tipos. Algunas de las características esenciales de este lenguaje de programación son:

- Facilidad de uso: Es fácil de usar, para quien está familiarizado con C++, ya que su estructuración básica es muy similar y el entorno de desarrollo: *Visual Studio .NET* posee un ambiente amigable.
- Programación orientada a objetos: Posee todas las ventajas que proporciona el paradigma de programación, como son la reutilización de código. Soporta la encapsulación, herencia y polimorfismo, además, no soporta la herencia múltiple, corrigiendo de esta manera, los problemas que presenta C++ y la herencia múltiple.
- Administración de memoria: Mediante el empleo de C#, no es necesario crear destructores para liberar el uso de la memoria innecesaria, pues posee destructores que actúan automáticamente.
- Seguridad en el manejo de datos: Se mantiene comprobando, de manera constante, la lógica y la semántica del programa que se está codificando, para verificar que sean correctos, lo que permite evitar errores en tiempo de ejecución.

Ventajas

- En un mismo espacio de nombres se pueden definir varias clases.
- Posee un rango más amplio y definido de tipos de datos que los que se encuentran en C, C++ o Java, aunque los más usados por los programadores, son comunes a los lenguajes mencionados.
- Soporta todas las características de la programación orientada a objetos: herencia, polimorfismo, encapsulación.
- Permite una mayor reutilización de código.
- Es muy factible, para desarrollar aplicaciones *desktop* en poco tiempo.

Desventajas

- El IDE *Visual Studio .NET* es propiedad de *Microsoft.*
- Es necesario tener una de las últimas versiones del *Visual Studio .NET*.
- Para quien no conozca un lenguaje de alto nivel como C++ le resultará engorroso aprender a trabajar con este lenguaje.

Java es un lenguaje de programación, que fue diseñado por la compañía *Sun Microsystems Inc*, con el propósito de lograr su funcionamiento en redes computacionales heterogéneas (redes de computadora formadas por más de un tipo de computadora ya sean PC, *MAC's*, estaciones de trabajo, etc.) y ser independiente de la plataforma en que se ejecute. Por tanto, un programa de Java puede ejecutarse en cualquier máquina o plataforma. (MENDEZ 2008)

Entre las características más sobresalientes de este lenguaje, se puede citar que:

- Es un lenguaje bastante simple, su diseño se basa en el paradigma de programación orientada a objetos.
- Resulta bastante familiar para los programadores que estén acostumbrados a lenguajes como C++ o C#.
- Se encarga automáticamente del manejo de la memoria.
- Fue diseñado con ciertas restricciones, sobre lo que se puede hacer y no sobre los recursos críticos de la computadora, aspecto que evita la posibilidad de codificar virus sobre él.
- Un programa escrito sobre Java es portable, ya que puede ser utilizado por cualquier computadora que tenga implementado su intérprete.
- Puede ejecutar diferentes líneas de código al mismo tiempo.
- Es un lenguaje interpretado, por tanto, puede ejecutarse sobre una máquina virtual.
- Es multiplataforma (debido a la "*Java Virtual Machine*").

Ventajas

- No es necesario volver a escribir el código cuando se desea ejecutar en otra máquina, basta con tener en ella la máquina virtual de Java instalada.
- Como es orientado a objetos, garantiza una mayor reutilización de código y demás ventajas que aporta este paradigma de programación.

- Mediante el uso de Java, se pueden realizar cálculos matemáticos, procesadores de palabras, bases de datos, aplicaciones gráficas, animaciones, sonido, hojas de cálculo, como con cualquier otro lenguaje de programación como C# o C++.
- Permite desarrollar páginas web dinámicas, interactivas, a las que se les pueden incorporar animaciones y toda clase de elementos de multimedia, sin necesidad de invertir en la compra de paquetes de multimedia, que tienen un alto costo.
- El JDK es una herramienta libre de licencias (sin costo).
- Cada 6 meses *Sun Microsystems Inc*, pone en el mercado una nueva versión del JDK.
- Es independiente de la plataforma de desarrollo.
- Tiene las librerías de clases gráficas: *AWT y SWING,* que permiten realizar un diseño más refinado y funcional de las interfaces comunes.
- Permite el acceso a bases de datos fácilmente, con JDBC, independientemente de la plataforma empleada o utilizando *frameworks* como *Hibernate* que están basados en JDBC (*Java Data Base Create*).
- Cuenta con varias herramientas y *frameworks* que aportan soluciones aplicables al desarrollo de varios software a problemas triviales.

Desventajas

- Los programas hechos en Java no poseen gran rapidez.
- Para quienes han programado en lenguajes como C++ o C#, no será muy difícil aprenderlo, lo cual resulta inversamente proporcional para las personas que no han tenido dicha experiencia.
- Es un lenguaje nuevo en el mercado, aún no se conocen bien todas las funcionalidades que posee.

Selección del lenguaje a utilizar

Después de haber realizado una investigación profunda, acerca de los lenguajes más utilizados por la comunidad internacional de programadores, se concluye que: C# y Java son dos lenguajes de alto nivel inspirados en el deseo de eliminar los inconvenientes de C++. Con C# y Java se puede lograr una buena reutilización del código, son dos potentes lenguajes que permiten realizar aplicaciones *desktop*, ambos soportan todas las características y por transitividad las ventajas de la programación orientada a objetos.

Pero en cuanto al entorno de desarrollo integrado, C# cuenta con menos posibilidades de uso que Java, pues para emplearlo, hay que tener una versión del *Visual Studio .NET*, que es un *software* propietario de *Microsoft*. Java, por su parte, cuenta con varios entornos de desarrollo libres como son: Eclipse, *NetBeans, gel, Dr java java IDE, Bluejava IDE, Jipe Java IDE, Jcreator Java IDE (light edition*), algunos de los cuales son software gratuito y otros son de pago.

Durante la investigación, se han identificado algunas ventajas que posee Java sobre C#, las que constituyen facilidades de implementación, para desarrollar el software propuesto.

Ventajas de Java frente a C#

- Tiene muchas librerías disponibles, como java.util.map, con las que no cuenta C#, las que son muy útiles y en su mayoría *open-source*, característica que es muy utilizada por los programadores, para mejorar las tecnologías.
- Es multiplataforma.
- Swing está muy bien diseñada, mediante el uso del patrón de diseño Modelo Vista Controlador (MVC), aspecto en el que aventaja a *Windows Forms.*
- Existen numerosos *frameworks* en Java que facilitan la realización de los reportes, la graficación de estadísticas, el acceso a datos, el trabajo con archivos multimedia, por lo que permiten ahorrar tiempo a los desarrolladores.
- Teniendo en cuenta que sobre Java no se pueden codificar virus, se garantiza mayor seguridad en las aplicaciones desarrolladas mediante el uso de dicho lenguaje, en cambio C# no brinda esta posibilidad.

Por las ventajas antes expuestas y teniendo presente que el IDE de C# es propiedad de la *Microsoft,* se ha determinado usar como lenguaje de programación: Java.

1.6 Entorno de desarrollo Integrado (IDE)

Un IDE es un entorno de programación, que ha sido empaquetado como un programa de aplicación, es decir, consiste en un editor de código, un compilador, un depurador y un constructor de interfaz gráfica GUI. Los IDEs pueden ser aplicaciones por sí solas, o pueden ser parte de aplicaciones existentes.

NetBeans

NetBeans es un proyecto de código abierto, fundado por la empresa *Sun MicroSystems. NetBeans* IDE es un entorno de desarrollo, es decir, una herramienta para que los programadores puedan escribir, compilar, depurar y ejecutar programas. Está escrito en Java, pero puede servir para cualquier otro lenguaje de programación. Existe además, un número importante de módulos para extenderlo. Es un producto libre y gratuito, sin restricciones de uso. (AUTORES 2009a)

Este entorno de desarrollo, ofrece varias ventajas a los programadores tanto de *Java* como de otros lenguajes, ya que es muy fácil de usar, posee un editor de texto que permite el completamiento de código, aspecto importante tanto para principiantes como para expertos en el lenguaje, es muy factible para el desarrollo de aplicaciones *desktop*, pues a diferencia del *Eclipse,* no es necesario instalarle *plug-ins* como el *Visual Editor*, por lo que ofrece muchas ventajas para el diseño visual (librería S*wing* y *awt*).

Antes de la versión 5.5, no era considerada una herramienta tan competente como el Eclipse, (en la comunidad internacional de programadores en Java), pero de ésta versión en lo adelante, ha demostrado ser suficientemente robusta. Con la versión 6.1, es posible crear aplicaciones *desktop*, conectadas a la base de datos (si está previamente creada), además se puede contar con una librería *Swing* mucho más abarcadora que en las versiones anteriores, aspecto importantísimo para el diseño visual de las aplicaciones *desktop*, pues permite desarrollar interfaces que cuentan con mayor cantidad de funcionalidades visuales, por lo que son más atractivas al usuario.

Como un aspecto negativo, se puede citar que: consume gran cantidad de recursos al ejecutarse en la máquina. Teniendo en cuenta el auge que posee en la actualidad el desarrollo del hardware, en función de producir computadoras más rápidas y capaces de atender mayor cantidad de procesos a la vez, se puede concluir que la característica antes expuesta, aunque tiene cierta importancia, no resulta muy relevante. Habiendo analizado las características del *NetBeans,* se determina que brinda mayores ventajas, que desventajas para el desarrollo de una aplicación, por tanto, puede ser empleado de manera eficaz.

Eclipse

Eclipse es originalmente creado por la IBM en noviembre de 2001, como sucesor de la familia de herramientas *VisualAge*. Pertenece a una comunidad de código abierto, sin ánimos de lucro, que se

centra en el desarrollo de una plataforma, formada por herramientas para crear y gestionar *software* en todo el ciclo de vida.

Es un entorno de desarrollo integrado, de código abierto, multiplataforma, que tiene como objetivo desarrollar "Aplicaciones de Cliente Enriquecido", lo que es opuesto a las aplicaciones "Cliente-liviano" basadas en navegadores. Esta plataforma, típicamente ha sido usada para desarrollar entornos de desarrollo integrados (del inglés IDE), como el IDE de Java, llamado *Java Development Toolkit* (JDT) y el compilador (ECJ), que se entrega como parte de Eclipse (y que son usados también para desarrollar el mismo Eclipse).

Es una plataforma universal para integrar herramientas de desarrollo, basadas en arquitecturas abiertas y en *plug-ins*. El uso de los *plug-ins* ayuda a integrar diversos lenguajes sobre un mismo IDE, debido a la no existencia de un IDE universal. Tiene varios aspectos desventajosos, entre ellos se puede citar que: consume muchos recursos de la máquina, es un poco lento con respecto a otros entornos de desarrollo, además, para manejarlo es necesario estudiar y aprender un poco sobre él.

Selección del IDE

Eclipse es el entorno de desarrollo integrado, para el lenguaje Java, más usado en todo el mundo, el 70% de los programadores en Java utilizan ésta herramienta. No obstante, no debe descartarse el *NetBeans*, pues ha tenido un gran avance a partir de la versión 5.5.

El Eclipse es muy usado en la Universidad de las Ciencias Informáticas, fundamentalmente para el desarrollo de aplicaciones *web* sobre *Java*, pues para las aplicaciones *desktop*, el *plug-ins* que utiliza con este fin es el "V*isual Editor*", el cual es complicado para trabajar y no es muy popular entre los programadores, sin embargo, el *NetBeans* es un entorno en el que no es necesario usar los *plug-ins* y que ha demostrado ser muy factible para aplicaciones *desktop.*

En la actualidad, *NetBeans* es uno de los IDE más utilizados en la Universidad de las Ciencias Informáticas, para el desarrollo de aplicaciones *desktop*, teniendo en cuenta la potente librería *Swing* que posee para el diseño visual, además, evita el trabajo con el V*isual Editor (*que utiliza Eclipse con el mismo propósito), pues ofrece un editor bastante cómodo para dicha finalidad. Teniendo en cuenta lo antes

planteado, se determina utilizar el N*etBeans,* para el desarrollo del sistema propuesto, pues en su última versión ofrece la librería *Swing,* que brinda más facilidades para el diseño de una interfaz visual atractiva a los usuarios, mejor elaborada y más funcional, con el propósito de lograr que interactúen con la aplicación de manera ágil y sencilla.

También se hará uso del Eclipse, por las facilidades que aporta en el trabajo con el lenguaje Java, específicamente en el acceso a los datos. Se determinó usar el *framework Hibernate,* para establecer la conexión de la aplicación con el gestor de Bases de Datos, que se realizará mediante la generación del código XML correspondiente a esta funcionalidad, el cual no puede ser generado con el *NetBeans*.

Frameworks soportados por Java

Un *framework* es un conjunto de clases y librerías que cooperan entre sí para lograr un código reutilizable, como una especie de plantilla, que se puede aplicar a cualquier clase de *software* que se desee desarrollar. Un programador adapta un *framework* a una aplicación cuando hereda y crea instancias de las clases del mismo.

Hibernate

Hibernate es un poderoso objeto relacional, que se caracteriza por poseer un alto rendimiento. Permite desarrollar clases persistentes siguiendo la programación orientada a objetos, incluida la asociación, la herencia, el polimorfismo, la composición y las colecciones. A diferencia de muchas otras soluciones de persistencia, no oculta las potencialidades de SQL.

Hibernate es un proyecto *open source* profesional. Este *framework* será utilizado con el propósito de agilizar la gestión de la base de datos, en aras de poder dedicarle más tiempo a la implementación del *software*.

JfreeChart

JfreeChart es una biblioteca gráfica de Java y es 100% libre, permite a los desarrolladores mostrar gráficos de calidad profesional en sus aplicaciones, de manera sencilla

JFreeChart cuenta con un extenso conjunto de características, entre las que a continuación se exponen:

- Cuenta con una API coherente y bien documentada, para el apoyo de una amplia gama de tipos de gráficos.
- Posee un diseño flexible, que es fácil de extender, tanto del lado del servidor, como de las aplicaciones cliente.
- Cuenta con un soporte para muchos tipos de salida, incluyendo los componentes *Swing*, archivos de imagen (incluyendo PNG y JPEG), gráficos vectoriales y los formatos de archivo (incluyendo PDF, EPS y SVG).
- *JFreeChart* es un *framework* de código abierto, que se distribuye bajo los términos de *Lesser General Public Licence (LGPL)*, la licencia de GNU que permite su uso en aplicaciones propietarias.

El uso del *framework,* responde a la necesidad de los desarrolladores de poder graficar gran cantidad de datos estadísticos, que son manejados en el estudio de adversarios, como por ejemplo: graficar la cantidad de golpes, por cada tipo de golpe.

JasperReport

JasperReports es el *framework* de código abierto de Java más reconocido mundialmente, el cual permite generar reportes. Se puede integrar fácilmente en cualquier aplicación Java, para ofrecer sofisticadas funcionalidades de impresión o web de presentación de informes. También, se puede utilizar para crear archivos de salida para su transformación en aplicaciones como *Excel*. Los informes se basan en un número definido, de forma independiente y con formato de secciones. Las secciones pueden contener elementos como el diseño de las líneas, rectángulos, las imágenes (estáticas o dinámicas) y los campos de texto, así como la fuente de datos y los datos computados.

JasperReports ofrece información operativa cuando es necesario: en la pantalla, en la impresora y para otras aplicaciones. Entre los formatos de salida que incluye, están PDF, HTML, XLS, CSV, RTF (*Word*), TXT o archivos XML.

El uso de este *framework,* hace posible la implementación de la funcionalidad crítica del software, que consiste en generar reportes, que deben contener la información solicitada por el usuario, la cual debe ser exportada a PDF.

Selección del Framework a utilizar.

Debido a que no existe un *framework* que realice por sí solo el acceso a datos, la gestión de reportes y la gráfica de estadísticas, se determina la utilización de los tres *framework,* que fueron descritos anteriormente, para poder implementar las funcionalidades referenciadas.

Herramientas derivadas de los *frameworks* seleccionados.

iReport IDE

iReport es una herramienta vinculada al leguaje Java, que permite crear e imprimir informes de forma rápida y sencilla. Además, cuenta con distintos formatos, para generar documentos (XML, PDF, XLS, HTML, CSV, etc.) y varias funciones que pueden ser personalizadas. Es gratuita y multiplataforma. Resulta necesaria su utilización, debido al uso del *framework JasperReport*.

Hibernate Tools

Hibernate Tools es un *plugin* para Eclipse, utilizado en el trabajo con el *framework Hibernate*. Este *plugin* tiene las siguientes características disponibles:

- Editor de mapeos: Es un editor para los archivos de mapeos XML de *Hibernate*, permitiendo auto completamiento y sintaxis resaltada.
- Consola: Permite configurar conexiones a base de datos, provee visualización de clases y sus relaciones.
- Ingeniería inversa: Permite generar las clases del modelo de dominio y los archivos de mapeos de *Hibernate* a partir de una base de datos.
- Asistentes: Presenta asistentes como: el generador de archivos de configuración, la configuración de la consola, entre otros.

1.7 Herramienta CASE de modelado con UML

Varias herramientas han sido creadas con el propósito de desarrollar la ingeniería de software, las que tienen el fin de desarrollar programas utilizando técnicas de diseño y metodologías bien definidas, soportadas por herramientas automatizadas. Constituyen objeto de estudio en este trabajo: el ***Rational Rose Enterprise Edition*** y **el *Visual Paradigm.***

El ***Visual Paradigm para UML*** es una herramienta UML y herramienta CASE fácil de usar, que soporta la última notación UML 2.1, cuenta con las siguientes características: ingeniería inversa, generación de código, importación desde *Rational Rose*, exportación/importación XMI, generador de informes, editor de figuras, integración con MS Visio, plug-in, integración IDE con *Visual Studio, IntelliJ IDEA, Eclipse, NetBeans* y otros. Además incluye el modelado colaborativo con *CVS* y *Subversion* y la interoperabilidad con modelos UML2 a través de XMI. Existen varias ediciones con prestaciones diferentes. Una de las más interesantes puede ser la *Community Edition* pues, aunque está algo limitada, es gratuita si se usa con fines no comerciales. (AUTORES 2008)

El ***Visual Paradigm Community Edition*** **(VP-UML CE),** es una plataforma de modelado de aprendizaje diseñada para el modelado UML, cuenta con una de las mejores tecnologías de modelado visual, por lo que permite la visualización UML en la última notación UML 2.1 en 13 tipos de diagramas. Posee una interfaz de usuario potente y fácil de usar, por lo que brinda a los desarrolladores la posibilidad de crear diagramas con mayor rapidez y sencillez. Permite la gestión de requisitos, el modelado de base de datos, el modelado visual, el diseño de instalaciones, la personalización de estilos y formatos, la interoperabilidad, etc.

Ventajas:

- El software es gratuito.
- Interfaz de usuario intuitiva.
- Actualizaciones automáticas.
- Arquitectura abierta.
- Permite un modelado visual intuitivo.
- UML se prorroga más allá de las últimas capacidades de UML 2.1.

Desventajas:

- Aunque es gratuito, este programa se encuentra bajo licencias que no permiten el estudio y modificación del mismo.
- Sólo permite un máximo de un diagrama por proyecto, de cada tipo de diagrama UML.
- No permite la integración a los IDE. (AUTORES 2009b)

Otra edición interesante es la del ***Visual Paradigm Enterprise Edition*** **(EE_VP_UML),** la cual es una plataforma para el modelado de sistemas, diseñada para ser usada por arquitectos, desarrolladores, diseñadores, analistas de procesos de negocio y modeladores de datos, con el propósito de acelerar todo el modelo de código para el complejo proceso de desplegar las aplicaciones empresariales a través de la tecnología de modelado visual que facilita el modelado de procesos de negocio. Posibilita la visualización de UML en la última notación UML 2.1 en 13 tipos de diagramas. Además, permite a los desarrolladores crear diagramas con mayor rapidez que cualquier herramienta en el mercado, al contar con una interfaz de usuario potente y fácil de usar, posee motores de generación de código de gran alcance, facilita la programación de bases de datos y cuenta con una excelente interoperabilidad, con mayor apoyo en los IDEs.

Características principales de *Visual Paradigm Enterprise Edition*:

- Soporta la notación 2.1 de UML, con todas sus funcionalidades y aporta nuevas capacidades a la misma.
- Permite el modelado de procesos del negocio.
- Permite la gestión de requisitos.
- Código de ingeniería de ida y vuelta.
- Posibilita grandes integraciones con IDEs.
- Permite la colaboración de los equipos (cliente).
- Permite la generación de código e ingeniería inversa.
- Permite la importación y exportación a XML e imagen.

- Posibilita la generación de informes en formatos conocidos como: PDF, MS Word y HTML. (AUTORES 2009c)

El ***Rational Rose Enterprise Edition*** es una herramienta Lower CASE, que permite el diseño detallado del *software* y la generación de código fuente (de programas y bases de datos) e ingeniería inversa (obtención del diseño a partir del código fuente), basado en modelos con soporte UML, promueve el trabajo en equipo y el desarrollo iterativo. Es una forma de ayuda para la comprensión del sistema y de sus distintos componentes. Su característica más significativa consiste en la creación de componentes, que contengan una serie de archivos dentro de los cuales se encuentran las distintas clases pertenecientes a dicho componente. Es un *software* propietario. (AUTORES 2009d)

Selección de la herramienta CASE de modelado con UML

Teniendo en cuenta las características del *Visual Paradigm Community Edition* se concluye que no es la herramienta indicada para el desarrollo del software, pues a pesar de ser un software libre y gratuito que ofrece funcionalidades interesantes, éstas se encuentran limitadas, además no es posible utilizarla con fines comerciales. Tampoco se decide usar el *Rational Rose Enterprise Edition*, porque es un *software* propietario. Por tanto, la herramienta CASE más apropiada para el desarrollo del software que informatice el Estudio de Adversario es el *Visual Paradigm Enterprise Edition*, al ser una herramienta potente, que brinda gran cantidad de funcionalidades de interés para el equipo de desarrollo, entre ellas se puede citar que permite la integración con los IDEs, la generación de código e ingeniería inversa, permite la creación de varios diagramas de cada tipo, etc. Aunque es *software* libre, no es gratuita, pero el centro donde se desarrollará el *software*, paga la licencia de uso de la misma.

1.8 Sistema Gestor de Base de Datos

Para el desarrollo de una aplicación, que debe contar con una base de datos, resulta necesario realizar un análisis sobre los diferentes gestores existentes, teniendo en cuenta sus características principales, valorando sus ventajas y desventajas, para determinar cuál de ellos es más factible, para gestionar la información que es manipulada durante la realización del Estudio de Adversario en el Boxeo. Por tanto, constituyen objeto de estudio en esta investigación, los gestores de bases de datos PostgreSQL y MySQL.

PostgreSQL

PostgreSQL es un sistema gestor de bases de datos, que no pertenece a ninguna compañía, sino que es dirigido por una comunidad de desarrolladores, que se hace llamar ¨*PostgreSQL Global Development Group*¨ y organizaciones comerciales que están a cargo de su desarrollo, cuyo portal es www.postgresql.org.

El *PostgreSQL* es un poderoso sistema manejador de bases de datos, es decir, un sistema diseñado para administrar grandes cantidades de datos, en la actualidad es la base de datos de código abierto (*open source*) más avanzada del mundo, por tanto, es comparado con un elefante para representar su robustez. (AUTORES 2009e)

PostgreSQL tiene más de 15 años de desarrollo activo y se ha ganado la reputación de ser confiable y mantener la integridad de los datos. De hecho hay compañías que aseguran haber tenido ejecutando el *PostgreSQL*, o sea, trabajando, durante varios años y con altas tasas de actividad sin haber experimentado problemas de ningún tipo. El *PostgreSQL* se ejecuta en la mayoría de los Sistemas Operativos más utilizados en el mundo incluyendo, Linux, varias versiones de UNIX y por supuesto Windows. (AUTORES 2009e)

Éste robusto sistema gestor de bases de datos es una solución real, a los complejos problemas del mundo empresarial y a la vez mantiene la eficiencia, al consultar los datos. Ha desarrollado y añadido al *PostgreSQL* las más interesantes y útiles características, que antes sólo podían hallarse en sistemas manejadores de bases de datos comerciales como *Oracle, DB2 o Sybase*, por lo que hace honor a su lema: "El manejador de bases de datos de código abierto más avanzado del mundo".

Debido a sus características técnicas sobresalientes, el *PostgreSQL* se ha ganado la admiración y el respeto de sus usuarios, así como el reconocimiento de la industria (ganador del *Linux New Media Award for Best Database System* y 3 veces ganador del *The Linux Journal Editors' Choice Award for best DBMS*). (AUTORES 2009e)

PostgreSQL es un producto *open source*, sin costos de licencia, que se convierte en una alternativa extremadamente atractiva para las empresas, que buscan un ahorro significativo de costos en activos. (AUTORES 2009e)

Este poderoso software posee numerosas interfaces nativas de lenguajes como son: C / C + +, Java, Net, Perl, Python, Ruby, Tcl, ODBC, entre otros.

PostgreSQL ejecuta procedimientos almacenados en más de una docena de lenguajes de programación, como Java, Perl, Python, Ruby, Tcl, C / C + +, y su propio PL / pgsql, que es similar a la de Oracle PL / SQL.

El código fuente de *PostgreSQL* está disponible bajo los más liberales términos de licencia de código abierto: la licencia BSD: por tanto pueden hacerse todas las modificaciones, mejoras o cambios que se estimen convenientes.

MySQL

MySQL es un sistema gestor de bases de datos, que permite realizar varias consultas y la conexión de varios usuarios de forma concurrente (multihilos y multiusuario). Cuenta con una comunidad de al menos 6 millones de desarrolladores. Desde enero de 2008 una subsidiaria de la Sun produce *MySQL* como *software* libre, en un esquema de licenciamiento dual. Por un lado es distribuido sobre la licencia GNU GPL, la cual restringe ciertos usos para los usuarios de los *software* desarrollados bajo ella, su propósito es declarar que el *software* cubierto por esta licencia es *software* libre y protegerlo de intentos de apropiación que restrinjan esas libertades a los usuarios. Sin embargo, es distribuido bajo cierta dualidad, por la sencilla razón de que: si el desarrollador de un sistema pretende usar *MySQL* para crear un *software* privativo con fines comerciales, debe comprarle a la Sun otra licencia para poder hacerlo.

A diferencia de proyectos como *Apache*, donde el desarrollo del *software* pertenece a una comunidad pública y el *copyright* pertenece al desarrollador individual, *MySQL* es propietario y está patrocinado por una empresa privada que posee el *copyright* de la mayor parte del código.

MySQL posibilita el acceso a bases de datos, escritas en varios lenguajes como son: C, C++, C#, Pascal, Delphi, Eiffel, Smalltalk, Java, Lisp, Perl, PHP, Python, Ruby, Gambas, REALbasic (para el sistema operativo Mac), FreeBASIC, y Tcl, cada uno de éstos utiliza una interfaz específica. Además, es multiplataforma.

Selección del Sistema Gestor de Bases de Datos.

MySQL y PostgreSQL son potentes gestores de bases de datos, ambos pueden ser ejecutados sobre las plataformas más usadas en todo el mundo: Windows, Linux y GNU, aunque *MySQL* tiene la ventaja en este sentido, pues permite su ejecución sobre un mayor número de sistemas operativos. Éstos gestores implementan una gran cantidad de interfaces para propiciar el acceso a datos, de aplicaciones desarrolladas sobre una larga lista de lenguajes de programación, principalmente Java, que ha sido el lenguaje seleccionado para desarrollar el software propuesto, además de que constituyen software libre.

Enfatizando en el carácter ¨libre¨ de ambos gestores, como se enunciaba anteriormente en la caracterización de *MySQL*, la dualidad que proporciona la licencia GNU GPL sobre la cual se distribuye, permite determinar que no es factible para el desarrollo de la aplicación que se concibe con fines comerciales, para el Instituto Nacional de Deportes, Educación Física y Recreación (INDER). En este sentido, brinda mayor ventaja la utilización del *PostgreSQL*, pues no es propiedad de ninguna compañía, sino que es producto de una comunidad de desarrolladores. Por tanto, se decide utilizar para el desarrollo del sistema propuesto, el gestor de base de datos PostgreSQL.

1.9 Conclusiones

En este capítulo, se realizó un bosquejo del estado del arte. También se investigó acerca de las herramientas a emplear en el *software* que se propone desarrollar. Teniendo en cuenta las características que deberá tener dicho *software* y las necesidades del equipo de desarrollo, se eligieron las herramientas y la metodología que serán utilizadas.

Teniendo en cuenta el estudio realizado previamente, sobre las tendencias que existen en la actualidad acerca de las tecnologías y herramientas más usadas en el campo de la informática, se toma como decisión desarrollar una aplicación de escritorio, sobre el lenguaje Java, con la integración de los *frameworks,* ganando en velocidad de desarrollo.

Para lograr tal resultado, se propone el uso de las herramientas *NetBeans*, Eclipse y *Visual Paradigm,* por las facilidades que éstas brindan. Todo este proceso, será controlado y orientado por la metodología RUP, que constituye una guía de cómo se debe desarrollar.

CAPÍTULO 2

CARACTERÍSTICAS DEL SISTEMA

2.1 Introducción

El objetivo fundamental de este capítulo, es exponer las características del sistema propuesto, para que cumpla con las expectativas de los clientes y usuarios finales. Por lo que inicialmente, se realiza una descripción detallada de los procesos identificados en el entorno de negocio, correspondiente al Estudio de Adversario en el Boxeo, para lograr una mayor comprensión sobre el tema. Luego, se detalla el software que deberá informatizar de dichos procesos. A continuación, se expone el modelo de dominio, identificando los conceptos fundamentales asociados al mismo, los que son recogidos en el glosario de términos. Además, se enumeran los requisitos funcionales y no funcionales que deberá tener dicho sistema y se define su alcance, mediante la descripción de los casos de uso; con este propósito son previamente identificados los actores, los casos de uso que responden a los requerimientos que deberá cumplir el software y las relaciones existentes entre ellos, a través del artefacto Modelo de Casos de Uso del Sistema.

2.2 Descripción de los procesos del negocio

El proceder del Estudio de Adversario, responde a una dirección indispensable para garantizar los éxitos deportivos en la Escuela Cubana de Boxeo, la misma se inicia con el comienzo de ciclo olímpico. Se debe destacar que en el marco de cada olimpiada, las distintas confederaciones internacionales se benefician del número de comité olímpico que allí asiste, utilizando este contexto para establecer las primeras reuniones oficiales. Así surge el primer encuentro de la Asociación Internacional de Boxeo Amateur (AIBA), momento en que se ofrece el calendario competitivo internacional. Seguidamente los federativos nacionales, dan a conocer las competencias regionales, por invitación o de otro tipo, quedando registrado todas las posibles competencias a las que pidieran asistir los boxeadores de Equipo Nacional de Boxeo.

Con toda esta información, la dirección de alto rendimiento, en correspondencia con la Comisión Nacional de Boxeo, y el Equipo Nacional, establecen las prioridades de participación competitiva, momento donde se inicia el proceso de Estudio de Adversario, atendiendo a las necesidades estratégicas de cada división de peso.

Esta actividad es organizada a través del contacto entre el federativo que representó a Cuba en la olimpiada, quien debió recopilar la mayor cantidad de información o cronogramas competitivos de la AIBA, así como las competencias regionales y nacionales, con el jefe de cátedra y jefe de colectivo técnico de entrenadores del Equipo Nacional de Boxeo.

Por tanto, el Estudio de Adversario se proyecta confidencialmente, entre los cuadros pedagógicos que tienen la mayor responsabilidad con la preparación de los boxeadores del Equipo Nacional de Boxeo y dará solución mediata o inmediata según las proyecciones de los boxeadores y la participación competitiva. Seguidamente los entrenadores junto al jefe de división, trazan el plan de entrenamiento individual para cada boxeador, que puede incluir la realización de una reunión educativa, donde el entrenador establece un diálogo entre el boxeador preseleccionado para participar en una competencia y otro boxeador, que pudiera ser miembro del Equipo Nacional, o no y se haya enfrentado al adversario de mayor relieve para el ciclo olímpico en cuestión. Con todo este proceder pedagógico, se alcanza una auténtica información que se define como elemento distintivo en la preparación teórica que demanda la formación de un deportista de alto rendimiento.

2.3 Descripción del sistema propuesto

Para informatizar los procesos enmarcados en el entorno de negocio, se propone desarrollar un sistema que cuente con dos módulos (*business* y *common*), que permitan una mejor organización de los casos de uso y dos roles (jefe técnico y especialista), que serán los facultados para interactuar con los casos de uso, teniendo en cuenta sus privilegios de acceso a la aplicación.

Se controlará la información almacenada en la base de datos del sistema, para asegurar su integridad y seguridad. Con este propósito, se asignará a cada rol solo los permisos necesarios, para que el usuario pueda realizar su trabajo. Por tanto, los usuarios anónimos no tendrán acceso a ninguna funcionalidad de la aplicación, solo los registrados, que serán los que desempeñen el rol de jefe técnico o especialista.

En el paquete *business* se agrupan los casos de uso correspondientes a las funcionalidades principales que deberá cumplir el software, es decir, gestionar las cuentas de los usuarios, autenticarlos en el sistema, gestionar los datos correspondientes a las competencias, los combates y los boxeadores, incluyendo los aspectos observados durante la ofensiva, además de visualizar los reportes que sean de interés al jefe técnico, el cual desempeña el rol de administrador del sistema y por tanto puede acceder a

todos los casos de uso. No obstante, el especialista desempeña un rol con menor nivel de acceso, al no permitírsele gestionar las cuentas de usuarios, ni ver los reportes.

En el paquete *common* se encuentra el caso de uso que posibilita la búsqueda de información correspondiente a determinada competencia, combate o boxeador. Además de visualizar la información solicitada, permite exportarla a PDF. Este módulo puede ser accedido por la persona que desempeñe el rol de especialista.

En resumen, el sistema permitirá gestionar toda la información que es manipulada durante la realización del Estudio de Adversario en el Boxeo y de esta manera contribuir a que los entrenadores den una mejor preparación física y psicológica a los boxeadores cubanos, con vista a que logren éxitos competitivos.

2.4 Modelado de dominio

Los procesos que se manifiestan durante la realización del Estudio de Adversario en el Boxeo, no se han definido con suficiente claridad, por tanto, no es posible realizar el modelo de negocio correspondiente y de esta manera, resulta de mayor factibilidad realizar un modelo de dominio.

El modelo de dominio permite representar de forma sencilla las clases conceptuales o entidades del mundo real, al capturar las variantes más importantes de objetos que existen o los eventos que suceden en el entorno donde estará el sistema. De manera que ayuda a los usuarios, clientes y desarrolladores a utilizar un lenguaje en común para entender el contexto en el que se emplaza el mismo.

2.4.1 Glosario de términos del dominio

Jefe de Cátedra: Es un cuadro pedagógico que labora en la Escuela Cubana de Boxeo, que dirige el proceso de Estudio de Adversarios, en coordinación con el Jefe de Colectivo Técnico.

Jefe de Colectivo Técnico*:* Es un cuadro pedagógico de la Escuela Cubana de Boxeo, que dirige el proceso de Estudio de Adversarios, en coordinación con el Jefe de Cátedra.

Entrenador: Es un cuadro pedagógico que labora en la Escuela Cubana de Boxeo. Utiliza la información recopilada por los expertos, mediante el estudio de determinados adversarios, para trazar Planes de Entrenamiento adecuados a los boxeadores que entrena.

Calendario Competitivo: Es un documento donde se establecen las prioridades de participación competitiva de los deportistas que conforman el Equipo Nacional de Boxeo.

Adversario objeto de estudio: Es el adversario que tienen mayor ranking internacional, por lo que es titular de olimpiadas, o el que la dirección pedagógica de la Escuela Cubana de Boxeo, junto a los entrenadores, decidan que deben estudiar, dado el caso que haya vencido en competencia a algún deportista de Cuba, en uno o más eventos importantes.

Experto: Es un entrenador designado para viajar al exterior a realizar el Estudio de Adversarios, en un continente en específico. Se selecciona teniendo en cuenta sus características fisiológicas, sus conocimientos sobre el idioma de los países que debe visitar y sus conocimientos técnico-tácticos sobre el boxeo.

Estudio de Adversario: Es una estrategia que utilizan los entrenadores de la Escuela Cubana de Boxeo, para obtener información técnico-táctica sobre los adversarios objeto de estudio. Tiene como propósito lograr una mejor preparación física y psicológica de los boxeadores cubanos, con vista a que obtengan la victoria en competencias.

Video del combate: Es la grabación de un combate, donde participa un adversario que es objeto de estudio.

Documentación: Son los documentos donde se recogen (mediante la observación de los combates), las informaciones técnico-tácticas sobre el desempeño de los adversarios objeto de estudio.

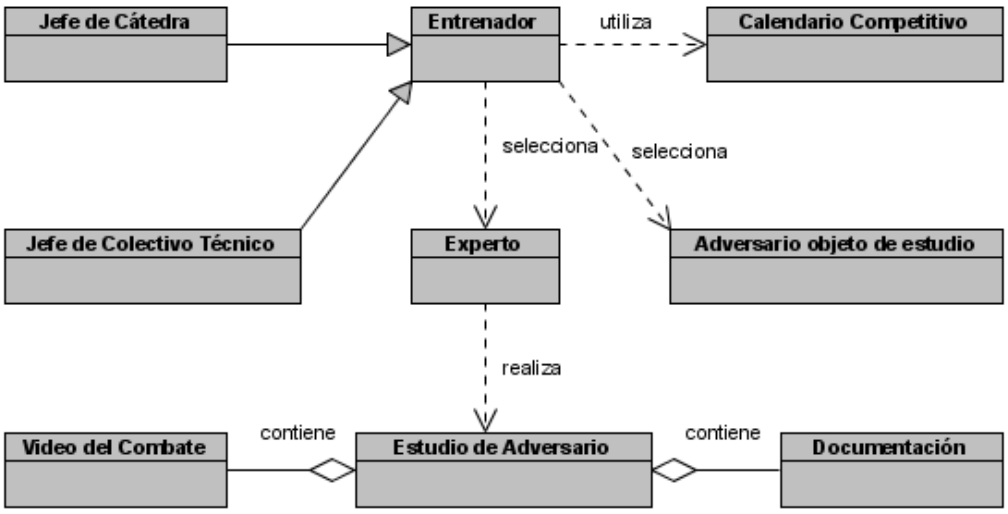


Figura 2. Modelo de dominio.

En las siguientes tablas se exponen las entidades y los trabajadores identificados en el dominio, cuyas descripciones se muestran en el Glosario de Términos abordado anteriormente.

Tabla 1. Trabajadores del dominio.

Jefe de colectivo técnico
Jefe de cátedra
Entrenador
Experto

Tabla 2. Entidades del dominio.

Calendario competitivo
Video del combate
Documentación

2.5 Requerimientos funcionales

Paquete Business

R1. Gestionar Usuario.

R1.1 Crear nuevo usuario. (Solicitar: nombre, apellidos, CI, usuario y contraseña)

R1.2 Modificar datos de un usuario.

R1.3 Eliminar cuenta de usuario.

R1.4 Gestionar privilegios de un usuario.

- Modificar privilegios de un usuario.
- Asignar privilegios de un usuario.

R1.5 Mostrar listado de los usuarios registrados. (nombre, apellidos, nombre de usuario, rol y privilegios)

R2. Autenticar usuario.

R2.1 Permitir al usuario entrar al sistema insertando su usuario y contraseña.

R2.2 Cambiar contraseña al usuario, una vez autenticado.

R2.3 Permitir al usuario acceder solo a los recursos que tiene privilegios.

R3. Gestionar Boxeador.

- Insertar datos del boxeador (nombre, apellidos, fecha de nacimiento, división, estatura, país por el que compite, país natal, mano hábil [izquierdo o derecho], mano adelantada [izquierdo o derecho] y ranking hasta el momento.
- Modificar los siguientes datos del boxeador:
  - Según cambios en la información, permitir modificar división, país por el que compite, ranking hasta el momento.
  - Según errores a la hora de entrar la información permitir modificar todos los datos.
- Exportar a PDF los datos personales del boxeador.

R4. Gestionar Competencia.

- Insertar los siguientes datos de la competencia: fecha de inicio, fecha de fin, tipo de evento, país sede, nombre de la competencia, listado de boxeadores que participan en la competencia, resultados que obtuvieron (oro, plata y bronce) en la competencia (organizado por división).
- Modificar los datos de la competencia.
- Exportar a PDF los datos de la competencia.

R5. Gestionar Combate.

- Insertar datos del combate (fecha, momento (todos contra todos, eliminatoria, cuartos de final, semifinal y final), boxeadores que se van a enfrentar (boxeador 1, boxeador 2), resultados del

combate (ganador y perdedor), forma en la que culmina el combate [KO, RSC, RSCH, puntos por golpes de coincidencia, puntos por votación individual, decisión de los jueces, superioridad técnica, descalificación], condición física en la que terminan los boxeadores [B, R, M]).

- Modificar los datos del combate.
- Exportar a PDF los datos del combate.

R6. Gestionar Ofensiva por asalto.

R6.1 Dado el nombre y apellidos de un boxeador, insertar los siguientes aspectos observados sobre su comportamiento durante los tres asaltos del combate:

- Distancia de combate del asalto (corta, media o larga).
- Cantidad de golpes por distancia de combate.
- Tipo de golpes.
- Tipo de combinaciones que realiza.
- Cantidad de golpes efectivos por tipos de golpes (puntos).
- Cantidad de golpes aislados.
- Cantidad de veces que el boxeador inicia el ataque.
- Cantidad de puntos que marca cuando realiza el ataque.
- Tipo de pelea por asalto (ataque, contraataque al encuentro y contraataque de riposta).
- Cantidad de puntos por tipo de pelea (ataque y contraataque).
- Comportamiento ante boxeadores de diferentes estaturas (B, R, M).
- Condiciones físicas en la que termina el boxeador (B, R, M).
- Tipo de defensa que utiliza.
- Brazo fuerte (izquierdo o derecho).
- Acciones (rápidas o lentas).

R7. Mostrar Reportes.

R9.1 Mostrar los siguientes datos (correspondientes a un boxeador) tomados en un rango de fecha o ante un adversario determinado.

- Distancia de combate preferida.

- Distancia de combate más efectiva.
- Graficar cantidad de golpes por distancia de combate.
- Tipo de golpe más efectivo.
- Graficar cantidad de golpes por tipo de golpe.
- Combinación más efectiva.
- Probabilidad que haga puntos al iniciar el ataque.
- Probabilidad que le hagan puntos en el contraataque.
- Probabilidad que le haga puntos cuando pelea de riposta.
- Asalto en el que golpea más.
- Graficar cantidad de golpes por asalto.
- Promedio de cantidad de golpes efectivo por asalto.
- Condiciones físicas más probables en que termina el boxeador.
- Cantidad de combates efectuados
- Cantidad eventos fundamentales ganados.
- Conocer si es un boxeador lento o rápido.
- Conocer su mano fuerte.
- Comportamiento ante adversarios de mayor o menor estatura.
- Defensa que más realiza, por asalto.
- Defensa que más realiza, durante todo el combate.

R9.2 Exportar a PDF el informe con el reporte ya sea en una temporada o contra un boxeador determinado.

Paquete Common

R8. Realizar Búsqueda.

R8.1 Realizar la búsqueda simple, sobre los datos correspondientes al boxeador objeto de estudio.

R8.1.1 Realizar búsqueda de texto libre, sobre la especificación de los datos esenciales de un boxeador: nombre y apellidos.

Teniendo en cuenta la importancia de los requisitos funcionales antes expuestos, se determinó su prioridad de implementación en el sistema, de esta manera:

El caso de uso crítico es:

- Mostrar Reportes.

Los casos de uso secundarios son:

- Gestionar Usuario.
- Autenticar Usuario.
- Gestionar Boxeador.
- Gestionar Competencia.
- Gestionar Combate.
- Gestionar Ofensiva.

El caso de uso auxiliar es:

- Realizar Búsqueda.

2.6 Requerimientos no funcionales

Usabilidad:

- El sistema podrá ser usado por cualquier usuario que posea conocimientos básicos en el manejo de la computadora y de la aplicación en sentido general.

Confiabilidad:

- La herramienta de implementación a utilizar tiene soporte para recuperación ante fallos y errores.
- La aplicación debe ser capaz de mantener la integridad de los datos.
- Garantía de un tratamiento adecuado de las excepciones y validación de las entradas del usuario.

Rendimiento:

- La aplicación debe contar con un tiempo de respuesta rápido a las peticiones del usuario, para lo cual debe realizar el procesamiento de los datos a gran velocidad.

Soporte:

- Garantía de instalación y prueba del sistema, además de un breve entrenamiento a los futuros usuarios.
- Se requiere un servidor de bases de datos con las siguientes características:
  - Soporte para grandes volúmenes de datos.
  - Gran velocidad de procesamiento, con tiempo de respuesta rápido en accesos concurrentes.

Interfaces de Software:

- Postgre SQL 8.2.1.
- SubVersion 1.4.5.
- Java.
- NetBeans 6.1.
- Visual Paradigm Enterprise Edition 6.1.
- Framework (Hibernate, JasperReport, JFreeChart, JavaMediaFramework).

Interfaces de Hardware:

- Pentium IV.
- 512 Mb de RAM o más.

Requisitos de Licencia:

- Para el proyecto Estudio de Adversarios no se requiere de patentes, ni licencias, puesto que es un software desarrollado sobre la plataforma de software libre.

Seguridad:

- La base de datos debe estar protegida con clave.
- Existencia de distintos roles que establezcan las acciones que pueden realizar los usuarios.

Instalador:

- El instalador debe garantizar de manera transparente la corrida exitosa de todos los componentes del programa.
- El instalador debe flexibilizar el directorio de instalación del producto.

Legales, de Derecho de Autor y otros:

- La aplicación tendrá todas las especificaciones legales posibles, por su importancia en el campo del deporte cubano.

Documentación de usuarios en línea y ayuda del sistema:

- El sistema debe poseer toda la documentación establecida por los estándares internacionales así como documentación de “ayuda”.
- Políticos-culturales: Se debe hacer uso correcto del idioma español en la interfaz pública de la aplicación, con logotipos e imágenes que se encuentren en correspondencia con el centro al que va destinada.

2.7 Modelo de Casos de Uso.

El modelo de casos de uso permite visualizar de una forma sencilla las funcionalidades que deberá tener el sistema propuesto, en él se definen los actores y los casos de uso que corresponden a las mismas.

Tabla 3. Justificación de los actores del sistema.

Actor	Descripción
Especialista	Es el rol encargado de gestionar toda la información correspondiente al boxeador (incluye los aspectos técnico-tácticos del mismo), la competencia y el combate. Puede consultar los reportes que sean de su interés y realizar búsquedas de información.
Jefe Técnico	Es el rol autorizado a administrar el sistema. Puede inicializar todos los casos de uso.

Para que el software cumpla con las funcionalidades previstas y de esta manera satisfaga las expectativas de los clientes y usuarios finales, se realiza una propuesta de 8 casos de uso, que servirán de guía durante el desarrollo del sistema.

2.7.1 Determinación de los casos de uso.

Tabla 4. Descripción del Caso de Uso Gestionar Usuario.

Caso de Uso:	Gestionar Usuario
Propósito:	Almacenar, modificar o eliminar los datos de las cuentas de usuario. Mostrar el listado de los usuarios existentes.
Actores:	Jefe Técnico
Prioridad:	Secundario.
Resumen:	El caso de uso se inicia cuando el actor selecciona la opción que le permite realizar una acción sobre un usuario en el sistema. El actor puede crear, modificar o eliminar los datos correspondientes a las cuentas de usuario. En caso de que seleccione la opción de crear un nuevo usuario, el sistema dará la posibilidad de introducir los siguientes datos: nombre, apellidos, ci, usuario, contraseña y de definir los privilegios para el usuario, que determinan su nivel de acceso y una vez

	introducidos dichos datos, los almacenará. Si el actor elige la opción de modificar los datos de un usuario, el sistema mostrará los datos que pueden ser editables y luego guardará las modificaciones realizadas. Si el actor elige la opción de eliminar una cuenta de usuario, el sistema procederá a realizar esta operación. Si el actor elige la opción de mostrar un listado con los usuarios registrados, el sistema mostrará dicho listado con los siguientes datos: nombre, apellidos, nombre de usuario y CI, dando fin al caso de uso.

Tabla 5. Descripción del Caso de Uso Autenticar Usuario.

Caso de Uso:	Autenticar Usuario
Propósito:	Entrar al sistema dado el nombre de usuario y contraseña, cambiar contraseña una vez autenticado, permitir al usuario acceder solo a los recursos que tiene privilegios.
Actores:	Especialista
Prioridad:	Secundario.
Resumen:	El caso de uso se inicializa cuando el actor introduce en la interfaz de autenticación su usuario y contraseña, el sistema deberá comprobar que son correctos para permitir su acceso a los recursos que están disponibles, teniendo en cuenta los privilegios con los que cuenta. El actor puede cambiar su contraseña, una vez autenticado en el sistema. Si el actor elige esta opción, el sistema deberá solicitar que introduzca su usuario, contraseña anterior y la nueva contraseña, luego deberá comprobar que son válidos los datos introducidos y en este caso guardará las modificaciones, dando fin al caso de uso.

Tabla 6. Descripción del Caso de Uso Gestionar Boxeador.

Caso de Uso:	Gestionar Boxeador
Propósito:	Insertar, modificar y exportar a PDF los datos de determinado boxeador.
Actores:	Especialista
Prioridad:	Secundario.
Resumen:	El caso de uso se inicializa cuando el actor selecciona la opción que le permite realizar una acción sobre un boxeador en el sistema. El actor puede insertar y

	modificar los datos de un boxeador. Si el actor elige la opción de insertar los datos de un boxeador, el sistema dará la posibilidad de almacenar los siguientes datos: nombre, apellidos, edad, división, estatura, país por el que compite, país natal, mano hábil (izquierdo, derecho), mano adelantada (izquierdo, derecho) y ranking hasta el momento. Si el actor elige la opción de modificar los datos del boxeador, el sistema mostrará los que pueden ser editables, teniendo en cuenta si se desean realizar cambios en la información, o si se produjo algún error durante la introducción de los mismos y posteriormente guardará las modificaciones realizadas. El sistema permite exportar a PDF los datos del boxeador, dando fin al caso de uso.

Tabla 7. Descripción del Caso de Uso Gestionar Competencia.

Caso de Uso:	Gestionar Competencia
Propósito:	Insertar, modificar y exportar a PDF los datos de determinada competencia.
Actores:	Especialista
Prioridad:	Secundario.
Resumen:	El caso de uso se inicializa cuando el actor selecciona la opción que le permite realizar una acción sobre una competencia en el sistema, teniendo en cuenta que posee los permisos suficientes para insertar y modificar los datos de la misma. Si el usuario elige la opción de insertar los datos, el sistema dará la posibilidad de almacenar los siguientes datos: fecha de inicio, fecha de fin, país sede, nombre de la competencia, listado de boxeadores que participan en ella, listado de los boxeadores que obtuvieron oro, plata y bronce (organizado por división). En caso de seleccionar la opción de modificar los datos, el sistema mostrará los datos que pueden ser editables y luego guardará las modificaciones realizadas. El sistema permitirá exportar a PDF la información correspondiente a determinada competencia, dando fin al caso de uso.

Tabla 8. Descripción del Caso de Uso Gestionar Combate.

Caso de Uso:	Gestionar Combate
Propósito:	Insertar, modificar y exportar a PDF los datos de un combate.

Actores:	Especialista
Prioridad:	Secundario.
Resumen:	El caso de uso se inicializa cuando el actor selecciona la opción que le permite realizar una acción sobre un combate en el sistema. El actor puede insertar y modificar los datos del combate. Si el actor elige la opción de insertar los datos del combate, el sistema dará la posibilidad de almacenar los siguientes datos: fecha, momento (todos contra todos, eliminatoria, cuartos de final, semifinal y final), boxeadores que se van a enfrentar (boxeador 1, boxeador 2), resultados del combate (ganador y perdedor), forma en la que culmina el combate (KO, RSC, RSCH, puntos por golpes de coincidencia, puntos por votación individual, decisión de los jueces, superioridad técnica, descalificación), condición física en la que terminan los boxeadores (B, R, M). Si el actor elige la opción de modificar los datos del combate, el sistema mostrará los datos que pueden ser editables y luego guardará las modificaciones realizadas. El sistema también permite exportar a PDF los datos del combate, dando fin al caso de uso.

Tabla 9. Descripción del Caso de Uso Gestionar ofensiva por asalto.

Caso de Uso:	Gestionar ofensiva por asalto
Propósito:	Almacenar los aspectos observados sobre el comportamiento de determinado boxeador, durante los tres asaltos de un combate.
Actores:	Especialista
Prioridad:	Secundario.
Resumen:	El caso de uso se inicializa cuando el actor selecciona la opción que le permite almacenar los aspectos que ha observado sobre el comportamiento de un boxeador durante un combate, el sistema dará la posibilidad de almacenar los siguientes datos: distancia de combate del asalto (corta, media o larga), cantidad de golpes por distancia de combate, tipo de golpes, tipo de combinaciones que realiza, cantidad de golpes efectivos por tipos de golpes (puntos), cantidad de golpes aislados, cantidad de veces que el boxeador inicia el ataque, cantidad de puntos que marca cuando realiza el ataque, tipo de pelea por asalto (ataque,

	contraataque al encuentro y contraataque de riposta), cantidad de puntos por tipo de pelea (ataque y contraataque), comportamiento ante boxeadores de diferentes estaturas (B, R, M), condiciones físicas en la que termina el boxeador (B, R, M), tipo de defensa que utiliza, brazo fuerte (izquierdo o derecho), acciones (rápidas o lentas), dando fin al caso de uso.

Tabla 10. Descripción del Caso de Uso Mostrar Reportes.

Caso de Uso:	Mostrar Reportes
Propósito:	Mostrar reportes del comportamiento de un boxeador determinado, en un rango de fecha dado, o ante un adversario determinado.
Actores:	Jefe técnico.
Prioridad:	Crítico.
Resumen:	El caso de uso se inicializa cuando el actor selecciona la opción que le permite visualizar el reporte del comportamiento en combate del boxeador que es objeto de estudio. El actor deberá especificar el nombre y los apellidos de dicho boxeador, para localizar sus datos en el sistema, además deberá establecer el rango de fecha en que desea que sea enmarcada la búsqueda, o especificar el nombre del adversario al que se ha enfrentado, para localizar la información correspondiente a los combates efectuados entre ambos. El sistema mostrará al usuario los siguientes datos: distancia de combate preferida, distancia de combate más efectiva, graficar cantidad de golpes por distancia de combate, tipo de golpe más efectivo, graficar cantidad de golpes por tipo de golpe, combinación más efectiva, probabilidad que haga puntos al iniciar el ataque, probabilidad que le hagan puntos en el contraataque, probabilidad que le haga puntos cuando pelea de riposta, asalto en el que golpea más, graficar cantidad de golpes por asalto, promedio de cantidad de golpes efectivo por asalto, condiciones físicas más probables en que termina el boxeador, cantidad de combates efectuados, cantidad eventos fundamentales ganados, conocer si es un boxeador lento o rápido, conocer su mano fuerte, comportamiento ante adversarios de mayor o menor estatura, defensa que más realiza, por asalto, defensa que más realiza,

	durante todo el combate. El sistema permitirá exportar a PDF cualquiera de los reportes que el usuario desee extraer del atleta.

Tabla 11. Descripción del Caso de Uso Realizar Búsqueda.

Caso de Uso:	Realizar Búsqueda.
Propósito:	Realizar la búsqueda simple, sobre los datos del boxeador objeto de estudio.
Actores:	Especialista.
Prioridad:	Auxiliar.
Resumen:	El caso de uso se inicializa cuando el actor selecciona la opción que permite realizar una búsqueda simple, para localizar la información correspondiente a determinado boxeador, por lo que, el sistema solicitará la introducción de los datos esenciales del mismo (nombre y apellidos) y mostrará los resultados encontrados, dando fin al caso de uso.

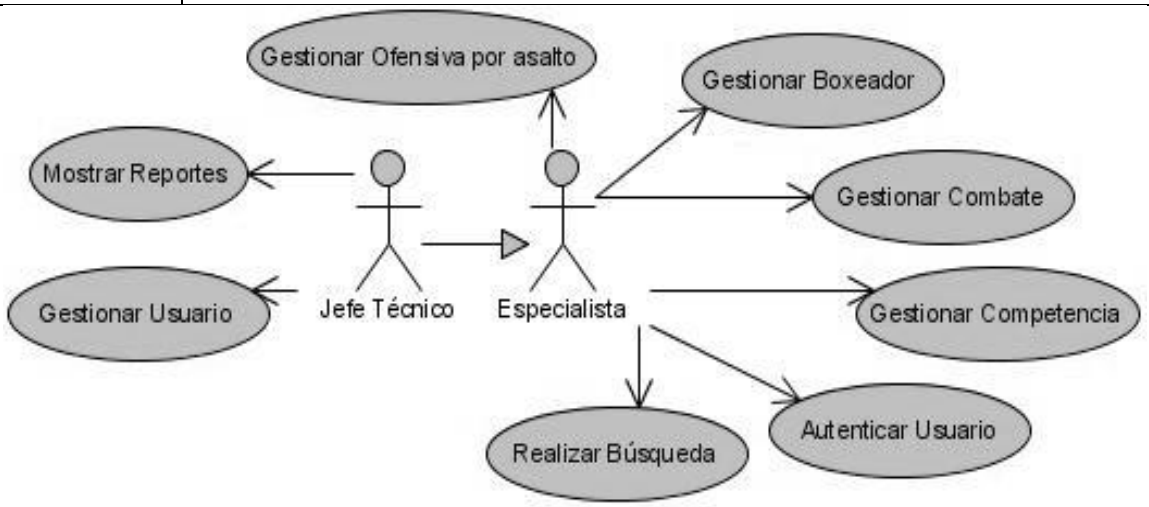


Figura 3. Modelo de casos de uso del sistema.

2.8 Conclusiones

En este capítulo se realizó un análisis de los procesos de negocio vigentes en el Estudio de Adversarios, que es llevado a cabo en el boxeo y se obtuvo como resultado una propuesta de desarrollo de software para su automatización. Con este propósito, se identificaron los requisitos funcionales y no funcionales que debe cumplir el sistema, los que se visualizaron a través del Modelo de Casos de Uso, donde se representaron los actores y casos de uso que engloban las funcionalidades deseadas por el cliente.

CAPÍTULO 3

ANÁLISIS Y DISEÑO DEL SISTEMA

3.1 Introducción

En este capítulo se realiza el análisis y diseño del sistema propuesto, para lo cual inicialmente se exponen los conceptos fundamentales asociados a este flujo de trabajo que propone la metodología *Rational Unified Process (RUP),* además se visualizan los diagramas de clases del análisis y del diseño, con sus respectivas clases, los atributos que las caracterizan y las relaciones existentes entre ellas.

También se propone un diseño para la base de datos, que permitirá almacenar la información correspondiente al Estudio de Adversarios en el Boxeo. A través del modelo lógico de datos (diagrama de clases persistentes) y el modelo físico de datos (modelo de datos).

3.2 ¿Qué es el análisis y el diseño?

El análisis y diseño es uno de los flujos de trabajo que propone la metodología RUP, cuyo objetivo fundamental es traducir los requisitos, a una especificación que describe cómo implementar el sistema propuesto.

El análisis permite obtener una visión de qué es lo que debe hacer el software, para cumplir con las necesidades y expectativas de los clientes y usuarios finales, a través de los requisitos funcionales del mismo. El diseño permite refinar el análisis, ocupándose de describir cómo el sistema debe cumplir con los requisitos funcionales establecidos, además de tener en cuenta los requisitos no funcionales, permitiendo de esta manera una implementación más eficiente del sistema.

El análisis y diseño permite refinar la arquitectura en cada una de las iteraciones del proceso de desarrollo, hasta que la misma sea lo suficientemente robusta para dar paso a la fase de implementación. Además posibilita el diseño de la base de datos mediante el modelo de datos correspondiente. El resultado más importante de esta fase es la obtención del Modelo de Diseño.

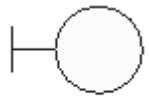
3.3 Modelo de Análisis

El modelo de análisis permite obtener una mejor visión de la propuesta de desarrollo del software. Tiene como punto de partida el modelo de casos de uso del sistema, de donde se extraen las clases que

conforman el diagrama de clases del análisis, el cual se descompone para agrupar las clases en paquetes. Por tanto, facilita la realización del diseño e implementación, sirviendo de soporte para su planificación y división en pequeños módulos.

Para representar las clases en el análisis se utilizan los siguientes estereotipos:

Clase interfaz: Modela la interfaz del sistema, es decir, su interacción con el actor a través de ventanas, formularios, o la comunicación con otros sistemas o dispositivos.



Clase control: Permite coordinar el trabajo de las clases, encapsulando el comportamiento de los casos de uso y realizando en varias ocasiones funciones complejas.

Clase entidad: Modelan el comportamiento asociado a la información y los datos del sistema que tienen larga vida, por lo que son persistentes.

3.3.1 Diagramas de clases del análisis

A continuación se muestran los diagramas de clases del análisis, agrupados por casos de uso, para lograr un mayor entendimiento de la lógica de negocio.

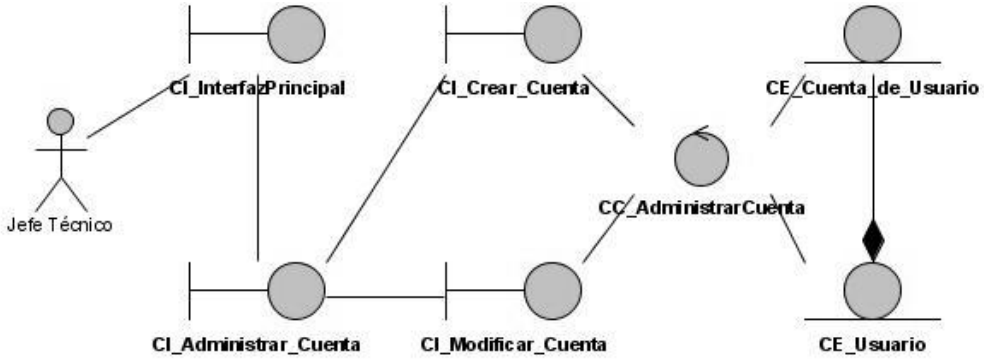


Figura 4. Diagrama de clases del análisis de Gestionar usuario.

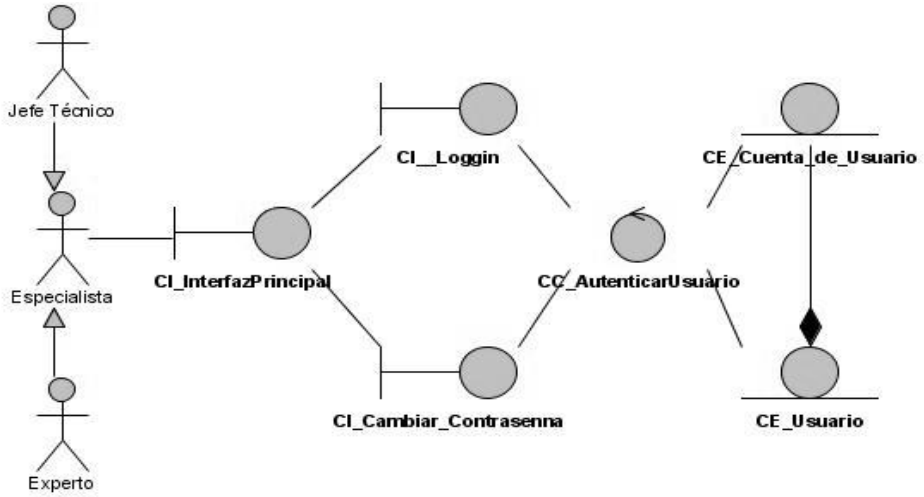


Figura 5. Diagrama de clases del análisis de Autenticar usuario.

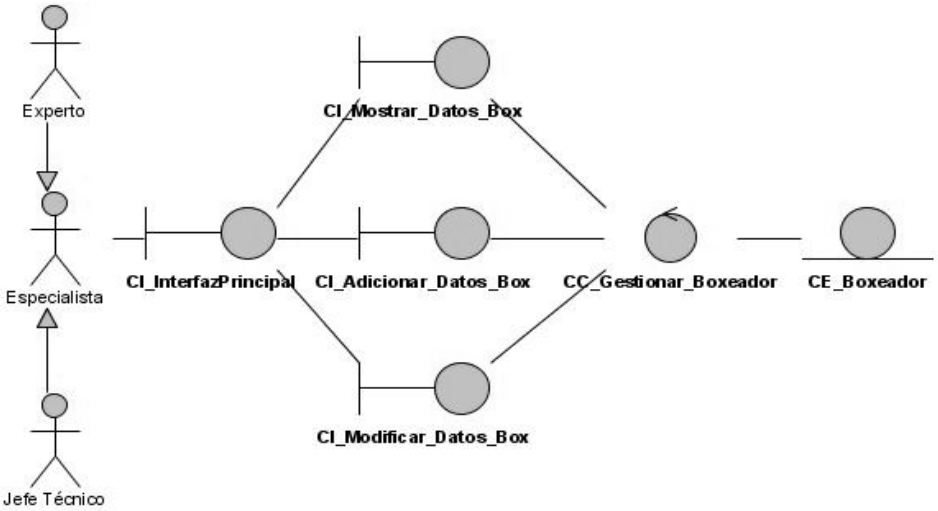


Figura 6. Diagrama de clases del análisis de Gestionar boxeador.

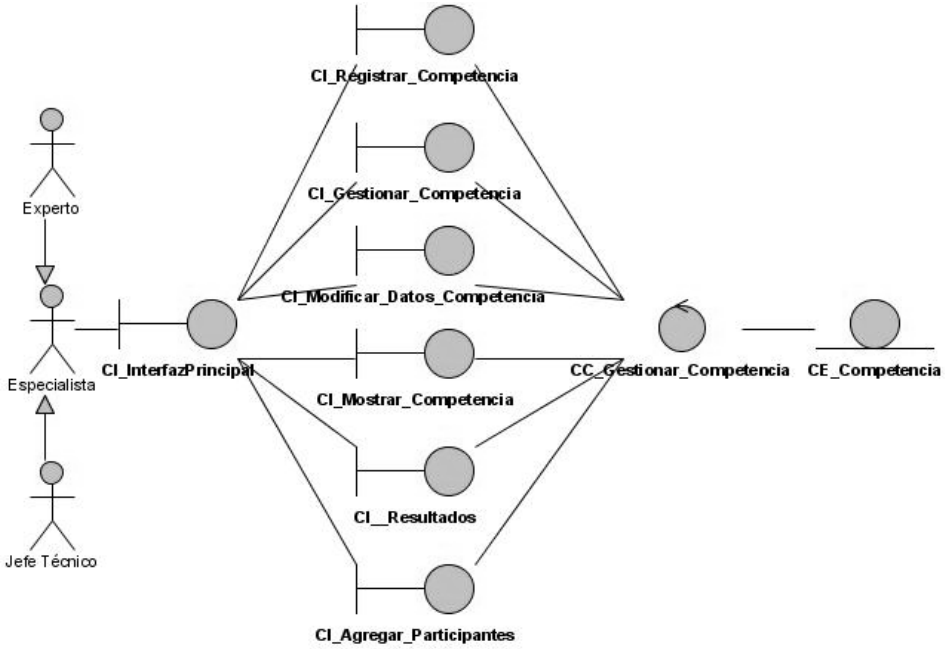


Figura 7. Diagrama de clases del análisis de Gestionar competencia.

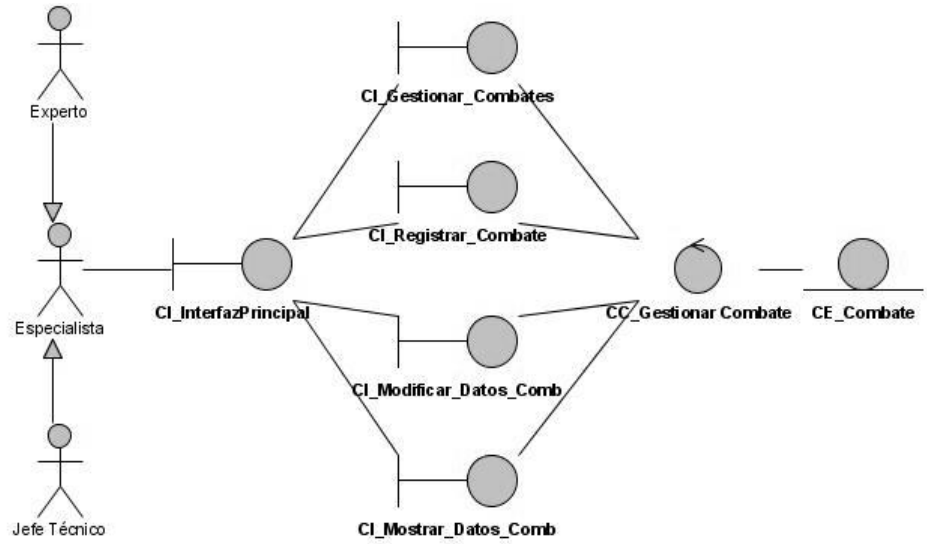


Figura 8. Diagrama de clases del análisis de Gestionar combate.

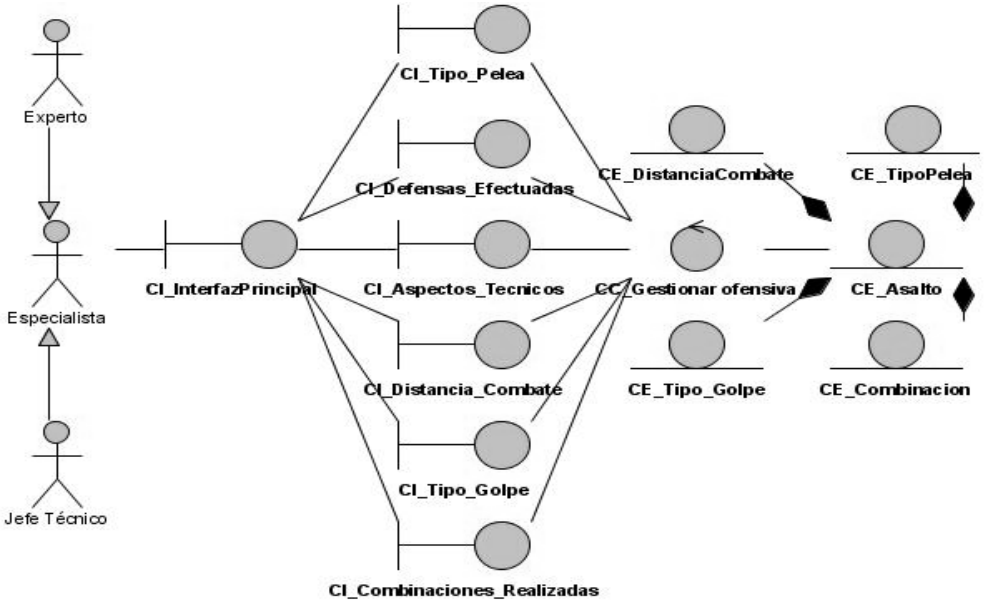


Figura 9. Diagrama de clases del análisis de Gestionar ofensiva por asalto.

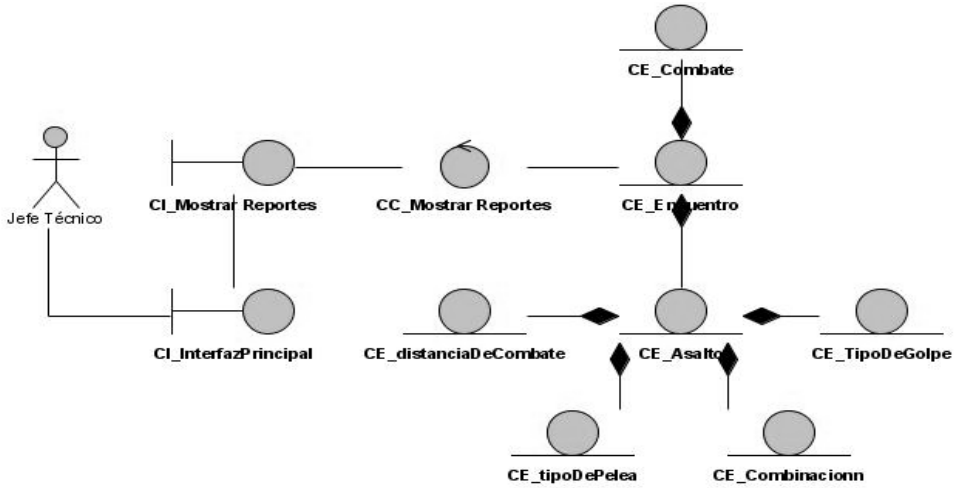


Figura 10. Diagrama de clases del análisis de Mostrar reportes.

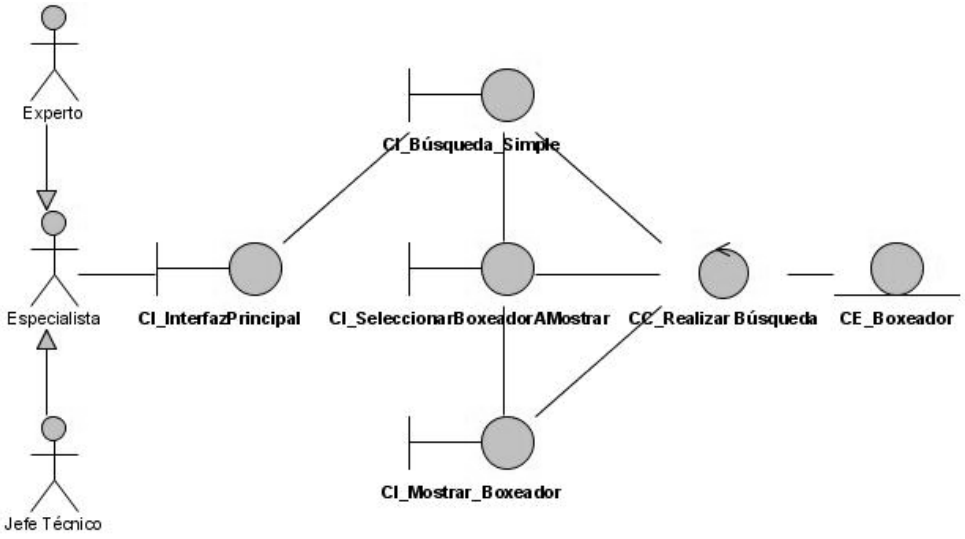


Figura 11. Diagrama de clases del análisis de Realizar búsqueda.

3.4 Modelo de Diseño

El modelo de diseño describe la realización física de los casos de uso, haciendo énfasis en cómo los requisitos funcionales y no funcionales junto a otras restricciones relacionadas con la programación, conforman el sistema que se propone desarrollar. Tiene como propósito fundamental, lograr que los desarrolladores comprendan los aspectos relacionados con los requisitos no funcionales y las restricciones de implementación, para que puedan realizar un trabajo eficiente. Además, permite la descomposición de las tareas de implementación en partes más manejables, que puedan ser llevadas a cabo por varios equipos de desarrollo.

El modelo de diseño es muy semejante al modelo de implementación, debido a que es importante mantenerlo durante todo el ciclo de desarrollo del software.

3.4.1 Principios de diseño

Los principios de diseño constituyen una guía a los desarrolladores de software, les permiten conformar un diseño del sistema fácil de entender y acceder por los clientes y usuarios finales. A continuación se citan los Principios de Diseño Universal o Diseño para Todos (BETTYE ROSE CONNELL 1997):

1er Principio: Uso equiparable.

El diseño debe ser atractivo, útil y vendible a personas con diversas capacidades, las características de privacidad, garantía y seguridad deben estar igualmente disponibles para todos los usuarios. (BETTYE ROSE CONNELL 1997)

2do Principio: Uso flexible.

El diseño se debe acomodar a un rango de preferencias y habilidades individuales, para que pueda ser accedido y usado tanto con la mano derecha como con la izquierda, también debe facilitar al usuario la exactitud y precisión, al adaptarse a su paso o ritmo de navegación en la aplicación. (BETTYE ROSE CONNELL 1997)

3er Principio: Simple e intuitivo.

El diseño debe ser fácil de entender, acorde a la experiencia, conocimientos, habilidades lingüísticas o grado de concentración actual del usuario, por lo que se debe tener en cuenta varios aspectos importantes, entre ellos: la eliminación de la complejidad innecesaria, la consistencia con las expectativas e intuición del usuario, la organización de la información en correspondencia con su importancia, además se deben proporcionar avisos eficaces y métodos de respuesta necesarios, durante y tras la finalización de cada tarea. (BETTYE ROSE CONNELL 1997)

4to Principio: Información perceptible.

El diseño debe comunicar de manera eficaz la información necesaria para el usuario, atendiendo a las condiciones ambientales o a las capacidades sensoriales del mismo. Es decir, debe usar diferentes modos, para presentar de manera redundante, la información esencial (tanto gráfica, como verbalmente) y proporcionar el contraste suficiente entre los datos fundamentales gestionados y sus alrededores, además de ampliar su legibilidad. (BETTYE ROSE CONNELL 1997)

5to Principio: Con tolerancia al error.

El diseño debe minimizar los riesgos y las consecuencias adversas de acciones involuntarias o accidentales, o sea, debe disponer de los elementos necesarios para minimizar los riesgos y errores: elementos más usados, más accesibles; y los elementos peligrosos eliminados, aislados o tapados, también ha de proporcionar advertencias sobre peligros y errores, debe poseer características seguras de interrupción y desalentar acciones inconscientes en tareas que requieren vigilancia. (BETTYE ROSE CONNELL 1997)

6to Principio: Que exija poco esfuerzo físico.

El diseño debe ser usado eficaz y confortablemente, con un mínimo de fatiga, por lo que ha de permitir que el usuario mantenga una posición corporal neutra, que utilice de manera razonable las fuerzas necesarias para operar, que minimice las acciones repetitivas y el esfuerzo físico continuado. (BETTYE ROSE CONNELL 1997)

7mo Principio: Tamaño y espacio para el acceso y uso.

Debe proporcionar un tamaño y espacio apropiados para el acceso, alcance, manipulación y uso, atendiendo al tamaño del cuerpo, la postura o la movilidad del usuario, es decir, debe facilitar una línea de visión clara hacia los elementos importantes y permitir el alcance de cualquier componente de manera confortable, tanto para un usuario sentado, como de pie, también debe proporcionar el espacio necesario para el uso de ayudas técnicas o de asistencia personal. (BETTYE ROSE CONNELL 1997)

3.4.2 Patrones de diseño

Los patrones de diseño constituyen la compilación de la experiencia de muchas personas durante el desarrollo de diversos software. Son soluciones simples y elegantes a problemas específicos y comunes del diseño orientado a objetos, las cuales se ha demostrado que funcionan. Permiten que el diseño de determinado sistema sea flexible, modular y reutilizable, facilitando notablemente el trabajo posterior.

Fundamentación del uso de patrones:

Para realizar un diseño eficiente del sistema propuesto, se hará uso de los siguientes patrones:

- Modelo Vista Controlador (MVC):

  Es un patrón de arquitectura de software que utiliza tres componentes distintos (modelo, vista y controlador) para separar los datos de la aplicación, de la interfaz de usuario y la lógica de control. El modelo está compuesto por las entidades, que son las clases contenedoras de toda la información que es manipulada en el sistema y por lo general es persistente. La vista la conforman las interfaces, que son las que permiten la interacción entre el usuario y el sistema. Como su nombre lo indica, el controlador representa a las clases de control, que son encargadas de responder a las acciones del usuario e invocar cambios en el modelo y en ocasiones también en la interfaz. (ANÓNIMO 2009).

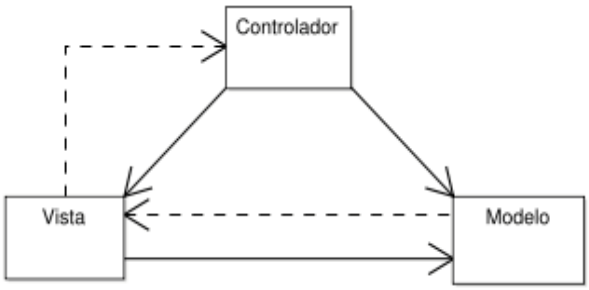


Figura 12. Patrón de arquitectura de software Modelo Vista Controlador.

- *Data Access Object* (DAO, Objeto de Acceso a Datos):

  Es un patrón que cuenta con diversas fuentes de datos (bases de datos, archivos, servicios externos, etc.), encapsula y oculta la forma de acceder a la fuente de datos, es decir, se encarga de que el software cliente se centre en los datos que necesita, sin tener en cuenta cómo se realiza el acceso a los datos o cuál es la fuente de almacenamiento. (LAGO 2007)

  De esta manera, cuando la capa de lógica de negocio necesite interactuar con la base de datos, ésta utilizará los métodos ofrecidos por DAO. Generalmente la clase de operaciones que ofrece la capa de datos, se le conoce como CRUD (*Create, Read, Update* y *Delete*). (JUAREZ 2008)

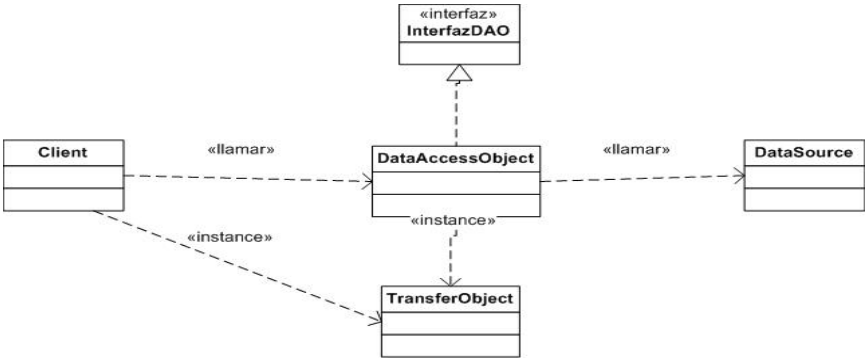


Figura 13. Patrón de diseño Data Access Object (DAO).

- Facade:

  Es un patrón estructural, que brinda una interfaz en común para un conjunto de interfaces de un subsistema. Al proveer una interfaz de alto nivel, pero sencilla, permite que el subsistema sea más fácil de usar. Implementa métodos convenientes para tareas comunes, reduce la dependencia de código externo en los trabajos internos de una biblioteca, permitiendo así más flexibilidad en el desarrollo de sistemas. (GRACIA 2005)

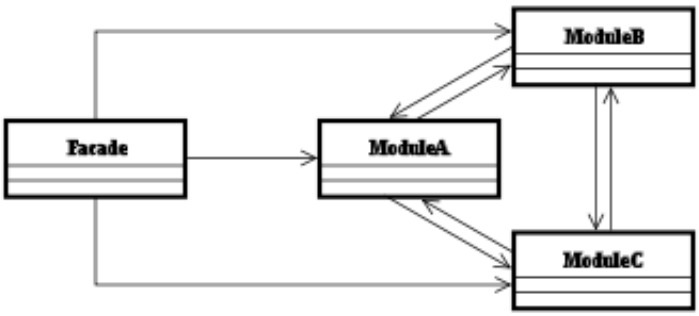


Figura 14. Patrón de diseño Facade.

- BeanFactory:

  Es una implementación del patrón Factory, pero a diferencia de las demás implementaciones de este patrón, que muchas veces solo producen un tipo de objeto, BeanFactory es de propósito general, ya que puede crear muchos tipos diferentes de Beans.

  Beans: Componentes de software que tienen la particularidad de ser reutilizables. Un Bean puede representar desde un botón, un *grid* de resultados, un panel contenedor o un simple campo de texto, hasta otras soluciones mucho más complejas como conexiones a bases de datos, etc.

  Factory: Es un patrón de diseño que posibilita instanciar objetos en tiempo de ejecución, por lo que permite tener el control de la creación de objetos.(GRACIA 2005)

3.4.3 Arquitectura de software

Las capas dentro de una arquitectura son un conjunto de servicios especializados, que pueden ser accesibles por múltiples clientes y que deben ser fácilmente reutilizables. Una capa puede contener muchos componentes, un mismo componente puede ubicarse en varias capas, de acuerdo a su naturaleza y a las consideraciones explícitas de la arquitectura. (ZAPATA 2009)

Cada componente de un sistema puede verse como un paquete o módulo que está compuesto por elementos que pueden ser clases y/o recursos complementarios, como archivos de configuración, imágenes, entre otros, por lo que es un elemento de software que encapsula una serie de funcionalidades. (ZAPATA 2009)

Los componentes desarrollados satisfacen una necesidad asociada a una o varias partes de una aplicación y son separados de acuerdo a su uso o por una agrupación lógica, que determina la relación entre ellos. (ZAPATA 2009)

El paradigma básico de la separación por capas establece al menos 3 partes distintas dentro de una aplicación:

- La presentación (o interfaz de usuario): Se refiere al mecanismo de interacción del usuario con el sistema.
- La lógica de negocio: Se refiere al conjunto de reglas que determinan específicamente como funciona un sistema, según su naturaleza y bajo qué parámetros y condiciones de acuerdo a las necesidades y condiciones de los usuarios.
- El acceso y almacenamiento de datos: El acceso a datos refiere al medio a través del cual se puede acceder y manipular los datos persistentes de un sistema. El almacenamiento de datos refiere a la forma en que se encuentran guardados, por ejemplo en archivos o bases de datos. (ZAPATA 2009)

Por tanto, una aplicación de 3 capas es aquella donde la interfaz, la lógica de negocio, el acceso a datos y los datos se encuentran separados. (ZAPATA 2009)

Figura 15. Arquitectura en 3 capas.

Teniendo en cuenta lo antes planteado, la arquitectura propuesta para el desarrollo de la propuesta de software se basa en el estilo n capas, implementando 3 capas lógicas:

- La presentación: Contiene todas las clases interfaz.

- La lógica de negocio: Contiene todas las clases controladoras, cuya implementación se basa en la utilización de los patrones de diseño: Manager y Facade. Además, utiliza los framework JasperReport y JFreeChart, para generar los reportes.
- El acceso y almacenamiento de datos: Implementa por sí solo el patrón arquitectónico Modelo Vista Controlador, debido al uso del framework de acceso a datos Hibernate. Por esta misma razón, implementa el patrón de diseño DAO y el BeanFactory. Además, contiene las clases del dominio.

3.4.4 Descripción de las clases del diseño

- **Clases entidad**

Las clases entidad contienen la información persistente que debe ser almacenada en la aplicación, por lo que están compuestas por un conjunto de atributos, además definen una serie de métodos, que son descritos a continuación:

- El constructor: permite crear objetos de la clase e inicializar los atributos que tiene definidos como parámetros.
- Métodos get: permiten retornar la información que contiene el atributo que les corresponde.
- Métodos set: permiten modificar la información que contiene el atributo que les corresponde.

Las clases entidad que se utilizan para desarrollar la propuesta de software son:

Tabla 12. Clases entidad.

Clase	Descripción
CE_asalto	Contiene la información correspondiente a cada uno de los asaltos del combate.
CE_tipo_de_pelea	Contiene la cantidad de puntos que marca el boxeador, por cada tipo de pelea que utiliza.
CE_tipo_de_golpe	Contiene la cantidad de puntos que marca el boxeador, por cada tipo de golpe que utiliza.
CE_distancia_combate	Contiene la cantidad de golpes que

	lanza determinado boxeador en la distancia corta, media y larga.
CE_boxeador	Contiene los datos personales que caracterizan determinado boxeador.
CE_competencia	Contiene los datos fundamentales de una competencia, es decir, su nombre, la fecha de inicio y la de fin, el país sede, el nombre de la competencia, el listado de los boxeadores que participan en la misma, el listado de los boxeadores que obtuvieron oro, plata y bronce en ella (organizado por división).
CE_nomPais	Contiene los nombres de los países del mundo y su identificador.
CE_combinacion	Contiene la cantidad de puntos que marca, por cada combinación que utiliza el boxeador.
CE_cuenta	Contiene los datos correspondientes a una cuenta de usuario: nombre de usuario y contraseña.
CE_usuario	Contiene los datos fundamentales del usuario, como son: el carné de identidad, el nombre, los apellidos, el rol que desempeña en la aplicación y su cuenta de usuario.
CE_encuentro	Contiene el resultado de un combate, es decir, boxeador ganador.
CE_evento	Contiene los resultados obtenidos

	en determinada competencia.
CE_combate	Contiene los datos que son recopilados durante el combate objeto de estudio.
CE_defensa	Contiene el identificador de la defensa realizada por determinado boxeador.
CE_nomTipoDeEventos	Contiene los nombres de los tipos de eventos en que participan los boxeadores, así como el identificador de cada cual.
CE_nomAcciones	Contiene las acciones que puede realizar el boxeador objeto de estudio, así como su identificador.
CE_nomDefensa	Contiene los tipos de defensa que puede realizar un boxeador, así como el identificador de cada una de ellas.
CE_nomDivision	Contiene el identificador y el nombre de cada división existente.
CE_nomGolpes	Contiene los nombres de los golpes que pueden ser lanzados por el boxeador objeto de estudio y incluyendo su identificador.

- **Clases de acceso a datos**

Son clases que no tienen atributos definidos, solo métodos, los que contienen las consultas que deben realizarse a la base de datos y de esta manera posibilitan acceder a ella. Con éste fin, son empleadas las clases DAO, mediante el uso del patrón de diseño *Data Access Object* (DAO) descrito anteriormente.

Cada clase DAO se encuentra estrechamente relacionada con una clase que la implementa, cuyo nombre está compuesto por el de su clase DAO análoga, seguida por las letras Impl.
Las clases DAO definidas en la propuesta de software son las siguientes:

- CE_UsuarioDAO
- CE_CuentaDAO
- CE_BoxeadorDAO
- CE_AsaltoDAO
- CE_CombateDAO
- CE_EncuentroDAO
- CE_TipoGolpeDAO
- CE_CombinacionDAO
- CE_DistanciaCombateDAO
- CE_TipoPeleaDAO
- CE_CompetenciaDAO
- CE_EventoDAO
- CE_NomPaisDAO
- CE_BaseDAO
- CE_DefensaDAO
- CE_NomTipodeEventosDAO
- CE_NomaccionesDAO
- CE_NomdefensaDAO
- CE_NomdivisionDAO
- CE_NomgolpesDAO

A continuación se describen los métodos que contiene la clase CE_BaseDAO, de la cual heredan el resto de las clases DAO definidas. También se especifica la clase CE_CuentaDAO y la CE_UsuarioDAO, que además de heredar los métodos correspondientes a la clase CE_BaseDAO, definen un conjunto de métodos que les otorgan mayor funcionalidad.

Tabla 13. CE_BaseDAO

Métodos:	**Descripción:**
findById(ID id, boolean lock)	Permite realizar una búsqueda de información, teniendo en cuenta el identificador pasado por parámetro.
persist(T entity)	Permite adicionar un objeto a la base de datos.
delete(T entity)	Permite eliminar un objeto, de la base de datos.
findAll()	Permite realizar una búsqueda de información.
findByExample(T exampleInstance,	Permite realizar una búsqueda, basada

String... excludeProperty)	en un ejemplo.

Tabla 14. Clase CE_CuentaDAO

Métodos:	**Descripción:**
verificarCuenta(String usuario, String contrasenna)	Permite verificar que el nombre de usuario y la contraseña introducida por el usuario son correctos.

Tabla 15. Clase CE_UsuarioDAO.

Métodos:	**Descripción:**
encontrarCuenta(String user)	Realiza una búsqueda de una cuenta cuyo nombre de usuario coincida con el que se le pasa por parámetro.
verificarNohayusuarioIgual (String usuario)	Permite verificar que no existe en la base de datos otra cuenta con el mismo nombre de usuario.
esUsuario(Usuario usuario)	Permite comprobar si una cuenta de usuario existe en la base de datos.
verificarUsuarioalMod(Usuario usuario)	Permite verificar que los datos de la cuenta de usuario son válidos, para luego modificar su contraseña.
verificarPrivilegiosusuario(int ci)	Verifica los privilegios que tiene definidos el usuario, teniendo en cuenta su rol en el sistema.

- **Clases controladoras**

Las clases controladoras, son las encargadas de realizar las principales funcionalidades en el sistema, por lo que mediarán entre las clases interfaz y las de acceso a datos, a través de los métodos que tienen definidos, que les permitirán realizar un conjunto de operaciones. Por tanto, las clases fachada (*facade*) y las *manager* son empleadas como clases controladoras, para desarrollar el software propuesto.

Las clases controladoras utilizadas se exponen a continuación:

- CC_UtilVisual
- CC_AutenticarUsuarioFacade

- CC_CommonFacade
- CC_CommonManager
- CC_AutenticarUsuarioManager
- CC_GestionarBoxeadorFacade
- CC_GestionarBoxeadorManager
- CC_GestionarCombateFacade
- CC_GestionarCombateManager
- CC_GestionarCompetenciaFacade
- CC_GestionarCompetenciaManager
- CC_GestionarOfensivaFacade
- CC_GestionarOfensivaManager
- CC_AdministrarCuentaFacade
- CC_AdministrarCuentaManager
- CC_MostrarReporteFacade
- CC_MostrarReporteManager
- CC_RealizarBusquedaFacade
- CC_RealizarBusquedaManager

A continuación se describen los métodos definidos en algunas clases controladoras:

Tabla 16. Clase controladora para trabajar en el visual.

Nombre de la clase:	**Descripción:**
CC_UtilVisual	Define un conjunto de métodos, que son comunes en las formas del visual. Contiene soluciones triviales, para trabajar con las clases de interfaz.
Métodos:	**Descripción:**
UtilVisual()	Constructor de la clase.
llenarComponente (JComponent componente, List lista)	Se utiliza para llenar componentes, dado el tipo de componente y la lista de elementos que va a contener.
llenarComboBox (JComboBox comboBox, List lista)	Permite llenar un ComboBox, dada una lista de elementos y un objeto de tipo ComboBox.
llenarJList (JList listBox, List lista)	Permite llenar un JList, dado un objeto de tipo JList y una lista de elementos.
soloNumeros()	Permite validar que solo se introduzcan números en la interfaz.
soloLetras()	Permite validar que solo se introduzcan

	letras en la interfaz.
llenarJTable (JTable tabla, List lista)	Permite llenar una tabla, dada una lista de elementos y un objeto de tipo JTable.
obtenerFecha (String fecha, String formato)	Permite convertir la fecha introducida, al formato Date: dd/mm/aaaa
obtenerDivision(Nomdivision division)	Dado un objeto Nomdivision retorna un string con la división a la que pertenece, que finalmente se muestra en el JComboBox.
ObtenerFecha(Date fecha)	Permite convertir un objeto de tipo Date a un string.
calcularEdad(Date fechaNacimiento)	Calcula la edad, dada una fecha de nacimiento.
obtenerPosicionPaisComboBox(JComboBox combo, int idPais)	Permite visualizar el país natal y el país por el cual compite determinado boxeador, en la interfaz destinada a modificar los datos del boxeador.
obtenerPosicionDivisionCombobox(JComboBox combo, int iddiv)	Permite obtener la división en la que compite determinado boxeador.
limpiarTablas(JTable tabla)	Permite limpiar las filas de una tabla.
obtenerDivision(int division)	Retorna la división en la que compite determinado boxeador.
comparaFechas()	Permite validar que la fecha de fin sea mayor que la de inicio de una competencia determinada.
mostrarDatosBoxeador()	Permite mostrar los datos de un boxeador.
mostrarDatosUsuario(JTable tabla, Home parent)	Permite mostrar los datos de determinado usuario.
retornaObjetoTabla(JTable tabla)	Retorna el objeto que es insertado en una

	tabla de la interfaz visual.

Tabla 17. Clase controladora para autenticar el usuario en el sistema.

Nombre:	**Descripción:**
CC_AutenticarUsuarioFacade	Controla el acceso de los usuarios al sistema, según el rol que desempeñen, además les permite cambiar su contraseña, una vez autenticados en la aplicación.
Métodos:	**Descripción:**
esUsuario(String usuario, String password)	Autentica al usuario en el sistema, dándole acceso solo a los recursos que tiene permitidos.
CambiarContrasenna(String user, String contranueva)	Permite al usuario cambiar su contraseña.
buscarUsuarioPorCuenta(Cuenta cuenta)	Permite buscar los datos de un usuario, dada una cuenta de usuario.
verificarContrasenna(String user,String contra)	Permite verificar que el nombre de usuario y la contraseña introducida son válidos.

Tabla 18. Clase controladora de fachada común.

Nombre:	**Descripción:**
CC_CommonFacade	Es una clase donde se definen un conjunto de métodos comunes, para un conjunto de clases controladoras que heredan de ella.
Métodos:	**Descripción:**
obtenerListadoPaises()	Permite listar los países, para que el usuario pueda seleccionar uno de ellos.
adicionarPais(Nompais pais)	Permite adicionar los datos correspondientes a un país.
obtenerPaisPorID(int idPais)	Permite conocer dado un país su

	identificador (id).
participantesCompetencia(Competencia comp)	Retorna un listado con los participantes en determinado evento o competencia.
obtenerListadoCompetencias()	Permite obtener un listado de las competencias.
obtenerBoxeadoresPorCompetenciaDivision(int idcompetencia, int division)	Retorna un listado con todos los boxeadores de una división dada, que participan en una competencia dada.
obtenerDivisiones()	Retorna un listado con los nombres de las divisiones.
obtenerEventosCompetitivos()	Retorna un listado con los nombres de los eventos competitivos.
obtenerAcciones()	Retorna un listado con las acciones realizadas por determinado boxeador.
obtenerDivisionPorID(int idnomdivision)	Retorna el nombre de una división, dado su identificador.
obtenerListaDefensas()	Retorna un listado con los tipos de defensa que existen.
competenciasHaParticipado(int idboxeador)	Permite obtener un listado de las competencias en que ha participado un boxeador determinado.
respondenAlNombre(String nombre)	Retorna los datos de los boxeadores cuyo nombre coincida con el especificado por parámetro.
respondenAlNombreyApellido(String nombre, String apellido)	Retorna los datos de los boxeadores cuyo nombre y primer apellido coincidan con los especificados por parámetro.
respondenAlNombreyApellidos(String nombre, String papellido, String	Retorna los datos de los boxeadores cuyo nombre y apellidos coincidan con los

sapellido)	especificados por parámetro.
respondenAlPapellidoSeg(String papellido, String apellido)	Retorna los datos de los boxeadores cuyos apellidos coincidan con los especificados por parámetro.
respondenAlPapellido(String papellido)	Retorna los datos de los boxeadores cuyo primer apellido coincida con el especificado por parámetro.
respondenAlSapellido(String sapellido)	Retorna los datos de los boxeadores cuyo segundo apellido coincida con el especificado por parámetro.

Tabla 19. Clase controladora para gestionar datos del boxeador.

Nombre:	**Descripción:**
CC_GestionarBoxeadorFacade	Modela una interfaz que controla la gestión de la información (archivo, modificación y visualización) correspondiente a los boxeadores objeto de estudio.
Métodos:	**Descripción:**
GestionarBoxeador()	Constructor de la clase.
adicionarBoxeador()	Permite adicionar los datos personales del boxeador.
obtenerBoxeadores()	Permite obtener un listado de todos los boxeadores.
obtenerBoxeadorPorId(int idBoxeador)	Permite obtener los datos de un boxeador, dado su identificador.
upDateBoxeador(Boxeador boxeador, Boxeador acambiar)	Permite actualizar los datos de un boxeador.
buscarBoxeadorPorPaisAndDivision (int idPais, int division)	Permite listar los boxeadores que participan en una división dada, por un

	país dado.
existeYaBoxeador(Boxeador boxeador)	Retorna el objeto boxeador, si éste ya existe en la base de datos.
obtenerCompetenciasPorBoxeador(int idBoxeador)	Retorna un listado con los nombres de las competencias en que ha participado el boxeador, cuyo identificador es pasado por parámetro.

Tabla 20. Clase controladora para administrar los datos del boxeador.

Nombre:	**Descripción:**
CC_GestionarBoxeadorManager	Administra la gestión de la información (archivo, modificación y visualización) correspondiente a los boxeadores objeto de estudio.
Métodos:	**Descripción:**
adicionarBoxeador(Boxeador box)	Permite adicionar los datos personales del boxeador.
modificarDatosBoxeador(Boxeador acambiar, Boxeador cambiado)	Permite modificar los datos de determinado boxeador.
boxeadorYaExiste(Boxeador box)	Retorna los datos del boxeador, si existe en la base de datos.
existeelBox(Boxeador box)	Retorna verdadero, en caso de que el boxeador pasado por parámetros ya exista en la base de datos, de lo contrario retorna falso.
buscarBoxeadorPorPaisAndDivision(int idPais, int division)	Retorna un listado con los boxeadores que pertenezcan al país y a la división pasados por parámetros.
combatesParticipa(int idboxeador, int idcompetencia)	Retorna un listado de los combates en los que participa determinado boxeador.

distanciaMasUsadaFrenteA(int idboxeador_a_est, int idboxeador_frente_a)	Retorna la distancia que más utiliza el boxeador objeto de estudio, frente a determinado adversario.
distanciaMasEfectiva(int idboxeador_a_est, int idboxeador_frente_a)	Retorna la distancia más efectiva que utiliza el boxeador objeto de estudio, frente a determinado adversario.
golpeMasEfectivo(int idboxeador_a_est, int idboxeador_frente_a)	Retorna el golpe más efectivo que utiliza el boxeador objeto de estudio, frente a determinado adversario.
probabilidadhagaPuntosalIniciarAtaque(int idboxeador_a_est, int idboxeador_frente_a)	Retorna la probabilidad de que el boxeador objeto de estudio haga puntos al iniciar el ataque, frente a determinado adversario.
probabilidadLehaganPuntosenelContraataque(int idboxeador_a_est, int idboxeador_frente_a)	Retorna la probabilidad de que le hagan puntos al boxeador objeto de estudio, frente a determinado adversario.
probabilidadLehaganPuntosenlaRiposta(int idboxeador_a_est, int idboxeador_frente_a)	Retorna la probabilidad de que le hagan puntos al boxeador objeto de estudio, cuando pelea de riposta frente a determinado adversario.
asaltoEnqueGolpeaMas(int idboxeador_a_est, int idboxeador_frente_a)	Retorna el asalto en que generalmente el boxeador objeto de estudio golpea más, frente a determinado adversario.
promedioGolpesEfectivosPrimerAsalto(int idboxeador_a_est, int idboxeador_frente_a)	Retorna el promedio de golpes efectivos, que realiza el boxeador objeto de estudio, frente al adversario especificado, en el primer asalto.
promedioGolpesEfectivosSegundoAsalto(int idboxeador_a_est, int idboxeador_frente_a)	Retorna el promedio de golpes efectivos, que realiza el boxeador objeto de estudio, frente al adversario especificado, en el

	segundo asalto.
promedioGolpesEfectivosTercerAsalto(int idboxeador_a_est, int idboxeador_frente_a)	Retorna el promedio de golpes efectivos, que realiza el boxeador objeto de estudio, frente al adversario especificado, en el tercer asalto.
condicionesfisicamasprobablealterminarelPrimerAsalto(int idboxeador_a_est, int idboxeador_frente_a)	Retorna una evaluación de la condición física en que termina el primer asalto el boxeador objeto de estudio, frente a determinado adversario.
condicionesfisicamasprobablealterminarelSegundoAsalto(int idboxeador_a_est, int idboxeador_frente_a)	Retorna una evaluación de la condición física en que termina el segundo asalto el boxeador objeto de estudio, frente a determinado adversario.
condicionesfisicamasprobablealterminarelTercerAsalto(int idboxeador_a_est, int idboxeador_frente_a)	Retorna una evaluación de la condición física en que termina el tercer asalto el boxeador objeto de estudio, frente a determinado adversario.
cantidadCombatesEfectuados(int idboxeador_a_est, int idboxeador_frente_a)	Retorna la cantidad de combates efectuados entre el boxeador objeto de estudio y determinado adversario.
accionMasProbable(int idboxeador_a_est, int idboxeador_frente_a)	Retorna la acción más probable que puede realizar el boxeador objeto de estudio, frente a determinado adversario.
brazoFuerte(int idboxeador_a_est, int idboxeador_frente_a)	Retorna el brazo fuerte del boxeador objeto de estudio, durante el enfrentamiento en combate a otro boxeador.
defensaQueMasRealizaenelPrimerAsalto(int idboxeador_a_est, int idboxeador_frente_a)	Retorna el tipo de defensa empleada por el boxeador objeto de estudio, frente a determinado adversario, durante el primer

	asalto del combate.
defensaQueMasRealizaenelSegundoAsalto(int idboxeador_a_est, int idboxeador_frente_a)	Retorna el tipo de defensa empleada por el boxeador objeto de estudio, frente a determinado adversario, durante el segundo asalto del combate.
defensaQueMasRealizaenelTercerAsalto(int idboxeador_a_est, int idboxeador_frente_a)	Retorna el tipo de defensa empleada por el boxeador objeto de estudio, frente a determinado adversario, durante el tercer asalto del combate.
obtenerBoxeadores()	Retorna un listado de boxeadores.
obtenerBoxeadorPorID(int idBoxeador)	Retorna los datos del boxeador, cuyo identificador coincida con el especificado por parámetros.

Tabla 21. Clase controladora para gestionar los datos del combate.

Nombre:	**Descripción:**
CC_GestionarCombateFacade	Modela una interfaz que controla la gestión de la información (archivo, modificación y visualización) correspondiente a los combates objeto de estudio.
Métodos:	**Descripción:**
GestionarCombate()	Constructor de la clase.
AdicionarCombate()	Permite adicionar los datos de un combate objeto de estudio.
ModificarDatosCombate()	Permite modificar los datos de un combate objeto de estudio.
ExportarDatosCombate()	Permite exportar a PDF los datos del combate objeto de estudio.

Tabla 22. Clase controladora para gestionar los datos de la competencia.

Nombre:	
CC_GestionarCompetenciaFacade	Modela una interfaz que controla la gestión de la información (archivo, modificación y visualización) correspondiente a las competencias objeto de estudio.
Métodos:	**Descripción:**
GestionarCompetencia()	Constructor de la clase.
AdicionarCompetencia()	Permite adicionar los datos de la competencia objeto de estudio.
ModificarCompetencia()	Permite modificar los datos de la competencia objeto de estudio.
ExportarDatosCompetencia()	Permite exportar a PDF los datos de la competencia objeto de estudio.

Tabla 23. Clase controladora para gestionar la ofensiva por asalto.

Nombre:	**Descripción:**
CC_GestionarOfensivaFacade	Modela una interfaz que controla la gestión de la información (archivo, modificación y visualización) correspondiente a los aspectos técnico-tácticos que caracterizan al boxeador objeto de estudio.
Métodos:	**Descripción:**
GestionarOfensiva()	Constructor de la clase.
AdicionarDatosOfensiva()	Permite adicionar los datos técnico-tácticos que caracterizan al boxeador objeto de estudio.

Tabla 24. Clase controladora para administrar las cuentas de usuario.

Nombre:	**Descripción:**
CC_AdministrarCuentaFacade	Modela una interfaz que controla la gestión de la información (archivo, modificación, visualización y eliminación) correspondiente a las cuentas de usuario. Además, permite mostrar un listado con los datos de los usuarios del sistema.
Métodos:	**Descripción:**
crearCuentaUsuario(Usuario: usuario)	Permite adicionar los datos correspondientes a un usuario.
usersRegistrados()	Permite visualizar un listado con los usuarios del sistema.
eliminarUsuario()	Permite eliminar los datos correspondientes a un usuario.
modificarUsuario()	Permite modificar los datos correspondientes a un usuario.

Tabla 25. Clase controladora para mostrar reportes.

Nombre:	**Descripción:**
CC_MostrarReporteFacade	Modela una interfaz que controla el proceso de generación de los reportes que permiten caracterizar los boxeadores objeto de estudio.
Métodos:	**Descripción:**
MostarReporte()	Permite mostrar un reporte con la caracterización del boxeador objeto de estudio.
ImprimirReporte()	Permite imprimir un reporte con la caracterización del boxeador objeto de estudio.

- **Clases de interfaz de usuario.**

Las clases de interfaz de usuario contienen los formularios que permiten introducir información en la aplicación, por tanto, median entre el usuario y la clase controladora, para realizar las operaciones solicitadas por el usuario. Heredan de la clase CC_UtilVisual, en la que se encuentran definidos un conjunto de métodos convenientes para interfaces comunes.

Las clases de interfaz de usuario que se definen en la propuesta de software son las siguientes:

- CI_CambiarContrasenna
- CI_Loggin
- CI_ElejirBoxeador
- CI_DatosPersonales
- CI_MostrarDatosBoxeador
- CI_RegistrarCombate
- CI_MostrarDatosCombate
- CI_GestionarCombate
- CI_Resultados
- CI_RegistrarCompetencia
- CI_MostrarCompetencia
- CI_GestionarCompetencia
- CI_AspectosTecnicos
- CI_CombinacionesRealizadas
- CI_DefensasEfectuadas
- CI_DistanciaCombate
- CI_TipoPelea
- CI_TipoDeGolpes
- CI_AdministrarCuentasUsuario
- CI_CrearCuenta
- CI_ModificarDatosUsuario
- CI_Simple
- CI_Avanzada
- CI_BoxeadorOfensiva
- CI_Home
- CI_Loggin
- CI_NuevoPais

A continuación se definen los atributos y métodos que contienen las clases de interfaz de la propuesta de software:

Tabla 26. Clase interfaz para adicionar datos personales del boxeador.

Nombre:	**Descripción:**

CI_DatosPersonales	Contiene un formulario que permite adicionar en la base de datos la información personal, correspondiente al boxeador objeto de estudio.
Atributos:	**Tipo de dato:**
Cancelar	JButton
Division	JComboBox
Estatura	JTextField
FNacimiento	JTextField
Guardar	JButton
ManoAdelantada	JComboBox
ManoHabil	JComboBox
Nombre	JTextField
PApellido	JTextField
PCompite	JComboBox
PNatal	JComboBox
Ranking	JSpinner
SApellido	JTextField
jLabel1	JLabel
jLabel10	JLabel
jLabel11	JLabel
jLabel14	JLabel
jLabel15	JLabel
jLabel12	JLabel
jLabel2	JLabel
jLabel3	JLabel
jLabel4	JLabel
jLabel6	JLabel
jLabel8	JLabel

jLabel9	JLabel
jPanel1	JPanel
Métodos:	**Descripción:**
DatosPersonales(java.awt.Frame parent, boolean modal)	Constructor de la clase.
llenarComponentes()	Este método utiliza el método llenarComponente() de la clase UtilVisual.
setCompetencia()	Permite eliminar los datos de determinada competencia.
resetearCampos()	Permite limpiar los componentes de la forma, para insertar nuevos datos.

Tabla 27. Clase interfaz para modificar los datos del boxeador.

Nombre:	**Descripción:**
CI_ElejirBoxeador	Contiene un formulario que permite modificar en la base de datos la información personal, correspondiente al boxeador objeto de estudio.
Atributos:	**Tipo de dato:**
Division	JComboBox
Estatura	JTextField
FNacimiento	JTextField
ManoAdelantada	JComboBox
ManoHabil	JComboBox
Nombre	JTextField
PApellido	JTextField
PCompite	JComboBox
PNatal	JComboBox
Ranking	JSpinner
SApellido	JTextField

buttonGroup1	ButtonGroup
jButton1	jButton
jButton2	jButton
jLabel1	JLabel
jLabel10	JLabel
jLabel11	JLabel
jLabel2	JLabel
jLabel3	JLabel
jLabel4	JLabel
jLabel5	JLabel
jLabel6	JLabel
jLabel7	JLabel
jLabel8	JLabel
jLabel9	JLabel
jPanel1	JPanel
jPanel2	JPanel
jPanel3	JPanel
jPanel4	JPanel
jPanel5	JPanel
jRadioButton1	jRadioButton
jRadioButton2	jRadioButton
Métodos:	**Descripción:**
ElejirBoxeador(java.awt.Frame parent, boolean modal)	Constructor de la clase.
getModificar()	Devuelve un objeto de tipo boxeador.
setModificar()	Cambia el atributo modificar de tipo boxeador, por otro.
llenarCampos()	Permite visualizar los datos del boxeador, en cada uno de los componentes de la interfaz

	visual.
deshabilitarComponenetes()	Deshabilita determinados componentes en la interfaz.
habilitarComponentes()	Habilita determinados componentes en la interfaz.
cambioInformacion()	Permite habilitar los componentes que según el criterio sobre el cual se va a modificar, contienen dichos campos.
seleccionarModificar()	Permite seleccionar el criterio sobre el cual se va a realizar la modificación.
loadForm()	Muestra datos en la interfaz visual, es decir, los listados de países y las divisiones existentes.
getDatospersonales()	Retorna los datos personales de determinado boxeador.
setDatospersonales(DatosPersonales datospersonales)	Modifica los datos personales de determinado boxeador.

Tabla 28. Clase interfaz para mostrar los datos personales del boxeador.

Nombre:	**Descripción:**
CI_MostrarDatosBoxeador	Permite mostrar la información personal, correspondiente al boxeador objeto de estudio.
Atributos:	**Tipo de dato:**
BFuerte	jLabel
Bpdf	JButton
Cerrar	JButton
Division	jLabel
Edad	jLabel
Estatura	jLabel

MAdelantada	jLabel
MHabil	jLabel
NombreApellidos	jLabel
PCompite	jLabel
PNatal	jLabel
Ranking	jLabel
jLabel1	jLabel
jLabel10	jLabel
jLabel2	jLabel
jLabel3	jLabel
jLabel4	jLabel
jLabel5	jLabel
jLabel7	jLabel
jLabel8	jLabel
jLabel9	jLabel
jPanel1	jPanel
Métodos:	**Descripción:**
MostrarDatosBoxeador(java.awt.Frame parent, boolean modal)	Constructor de la clase.
llenarCampos()	Permite llenar cada componente de la interfaz, con los datos del boxeador, que el usuario desea ver.
setBoxeador(Boxeador boxeador)	Permite modificar los datos de un boxeador.

Tabla 29. Clase interfaz para adicionar los datos de un combate.

Nombre:	**Descripción:**
CI_RegistrarCombate	Contiene un formulario que permite adicionar en la base de datos la información correspondiente a los combates objeto de estudio.

Atributos:	**Tipo de dato:**
CCompetencia	JComboBox
CDA	JComboBox
CDB	JComboBox
CDivision	JComboBox
CFecha	JTextField
CHFin	JTextField
CHinicio	JTextField
FCombate	JComboBox
buttonGroup1	ButtonGroup
jButton1	jButton
jButton2	jButton
jComboBox1	jComboBox
jComboBox2	jComboBox
jLabel1	jLabel
jLabel12	jLabel
jLabel13	jLabel
jLabel2	jLabel
jLabel3	jLabel
jLabel4	jLabel
jLabel5	jLabel
jLabel6	jLabel
jPanel1	jPanel
jPanel2	jPanel
jPanel3	jPanel
jRadioButton1	jRadioButton
jRadioButton2	jRadioButton
Métodos:	**Descripción:**
limpiarCampos()	Permite limpiar los componentes de la

	forma, para insertar nuevos datos.
llenarJCombo()	Permite llenar los ComboBox con las listas de los participantes en una competencia determinada.
llenarCCompetencia()	Permite llenar el ComboBox con la lista de competencias registradas en la base de datos.
onloadForm()	Utiliza los métodos llenarJCombo() y llenarCCompetencia(), con el fin de que, cuando se cree la forma visual, la misma tenga éstos elementos en sus componentes.

Tabla 30. Clase interfaz para mostrar los datos de un combate.

Nombre:	**Descripción:**
CI_MostrarDatosCombate	Permite mostrar la información correspondiente a los combates objeto de estudio.
Atributos:	**Tipo de dato:**
Combates	JComboBox
jButton1	jButton
jButton2	jButton
jLabel1	jLabel
jLabel2	jLabel
jLabel3	jLabel
jLabel4	jLabel
jLabel5	jLabel
jLabel6	jLabel
jLabel7	jLabel
jLabel8	jLabel

jLabel9	jLabel
jPanel1	jPanel
jPanel2	jPanel
jSeparator1	jSeparator
jSeparator2	jSeparator
Métodos:	**Descripción:**
MostrarDatosCombate()	Constructor de la clase.

Tabla 31. Clase interfaz para adicionar los datos de una competencia.

Nombre:	**Descripción:**
CI_RegistrarCompetencia	Contiene un formulario que permite adicionar en la base de datos la información correspondiente a las competencias objeto de estudio.
Atributos:	**Tipo de dato:**
Cerrar	JButton
Competencias	JComboBox
jLabel1	jLabel
jLabel2	jLabel
jLabel3	jLabel
jLabel4	jLabel
jPanel1	jPanel
jPanel2	jPanel
jPanel3	jPanel
jPanel4	jPanel
jScrollPane1	jScrollPane
jScrollPane2	jScrollPane
jScrollPane3	jScrollPane
jSeparator1	jSeparator
jTable1	jTable

jTable2	jTable
jTable3	jTable
Métodos:	**Descripción:**
estaExcluido(Boxeador boxeador)	Verifica si un elemento de tipo Boxeador, se encuentra en la lista excluido.
loadForm()	Utiliza el método llenarComponente() de la clase UtilVisual, con el fin de que cuando se cree la forma, tenga dichos elementos en los componentes.
limpiarCampos()	Permite limpiar los componentes de la forma, dejándolos listos para insertar nuevos datos.
listadoDeParticipantes()	Retorna una lista con los participantes de la competencia que se está registrando.
llenarJTableBAdicionados(Boxeador boxeador)	Permite llenar una tabla con los datos de un boxeador determinado.
llenarJTableBoxeadoresEncontrados()	Permite llenar una tabla con los datos de un boxeador, obtenidos a través de una consulta a la base de datos, según determinado criterio de búsqueda.

Tabla 32. Clase interfaz para mostrar los datos de una competencia.

Nombre:	**Descripción:**
CI_MostrarCompetencia	Permite mostrar la información correspondiente a las competencias objeto de estudio.
Atributos:	**Tipo de dato:**
UConfirmContra	JPasswordField
UContra	JPasswordField
UNCIdentidad	JTextField

UNombre	JTextField
UPApellido	JTextField
USApellido	JTextField
UTUsuario	JComboBox
UUsuario	JTextField
jButton5	jButton
jButton6	jButton
jLabel49	jLabel
jLabel50	jLabel
jLabel51	jLabel
jLabel52	jLabel
jLabel53	jLabel
jLabel54	jLabel
jLabel55	jLabel
jLabel56	jLabel
jPanel1	jPanel
jPanel2	jPanel
Métodos:	**Descripción:**
crearCuenta()	Permite crear una cuenta de usuario, con los datos correspondientes.
limpiarCampos()	Borra los campos de la interfaz visual.

Tabla 33. Clase interfaz para administrar cuentas de usuario.

Nombre:	**Descripción:**
CI_AdministrarCuentasUsuario	Permite gestionar la información correspondiente a las cuentas de usuario del sistema, es decir, posibilita el archivo, modificación, visualización y eliminación de dichos datos. Además, muestra un listado de los usuarios registrados en la aplicación.

Atributos:	**Tipo de dato:**
BEliminar	JButton
BModificar	JButton
BNUsuario	JButton
jPanel1	jPanel
jScrollPane1	JScrollPane
jTable1	JTable
Métodos:	**Descripción:**
AdministrarCuentasUsuario(java.awt.Frame parent, boolean modal)	Permite crear una cuenta de usuario, con los datos correspondientes.
getJTable1()	Retorna el valor del jTable1, es decir, la información que contiene.
llenarJTable()	Muestra los datos de los usuarios registrados en el sistema.
limpiarCampos()	Borra los campos de la interfaz visual.

Tabla 34. Clase interfaz para crear cuentas de usuario.

Nombre:	**Descripción:**
CI_CrearCuenta	Permite crear cuentas de usuario en el sistema, mediante un formulario.
Atributos:	**Tipo de dato:**
UConfirmContra	JPasswordField
UContra	JPasswordField
UNCIdentidad	JTextField
UNombre	JTextField
UPApellido	JTextField
USApellido	JTextField
UTUsuario	JComboBox
UUsuario	JTextField
jButton5	JButton

jButton6	JButton
jLabel49	JLabel
jLabel50	JLabel
jLabel51	JLabel
jLabel52	JLabel
jLabel53	JLabel
jLabel54	JLabel
jLabel55	JLabel
jLabel56	JLabel
jPanel1	JPanel
jPanel2	JPanel
Métodos:	**Descripción:**
CrearCuenta(java.awt.Frame parent, boolean modal)	Permite crear una cuenta de usuario, con los datos correspondientes.
Actualiza()	Permite actualizar los datos en la interfaz visual.
limpiarCampos()	Borra los campos de la interfaz visual.

Tabla 35. Clase interfaz para modificar los datos de un usuario.

Nombre:	**Descripción:**
CI_ModificarDatosUsuario	Permite modificar los datos de un usuario en el sistema.
Atributos:	**Tipo de dato:**
BAceptar	JButton
BCancelar	JButton
UConfirmContra	JPasswordField
UContra	JPasswordField
UNCIdentidad	JTextField
UNombre	JTextField
UPApellido	JTextField

USApellido	JTextField
UTUsuario	JComboBox
UUsuario	JTextField
jLabel49	JLabel
jLabel50	JLabel
jLabel51	JLabel
jLabel52	JLabel
jLabel53	JLabel
jLabel54	JLabel
jLabel55	JLabel
jLabel56	JLabel
jPanel1	JPanel
jPanel2	JPanel
Métodos:	**Descripción:**
MosdificarDatosUsuario(java.awt.Frame parent, boolean modal)	Constructor de la clase.
setUser(Usuario user)	Sustituye el usuario, por el que es pasado por parámetro.
getUser()	Devuelve los datos de determinado usuario.
LlenarCampos()	Permite mostrar datos en la interfaz visual.

Tabla 36. Clase interfaz para que el usuario modifique su contraseña.

Nombre:	**Descripción:**
CI_CambiarContrasenna	Permite al usuario cambiar su contraseña, en el momento que lo desee.
Atributos:	**Tipo de dato:**
BAceptar	JButton
BCancelar	JButton
CAnterior	JPasswordField
NContra	JPasswordField

User	JLabel
jLabel1	JLabel
jLabel3	JLabel
jLabel4	JLabel
jLabel5	JLabel
jPanel1	JPanel
Métodos:	**Descripción:**
CambiarContrasenna(java.awt.Frame parent, boolean modal)	Constructor de la clase.
setUsuario(Usuario usuario)	Permite sustituir los datos del usuario, por los que son pasados por parámetro.
getUsuario()	Retorna un objeto usuario, es decir, el usuario con todos sus datos.
getUser()	Retorna el nombre de determinado usuario.
limpiarCampos()	Borra los campos de la interfaz visual.

Tabla 37. Clase interfaz para que el usuario se autentique en el sistema.

Nombre:	**Descripción:**
CI_Loggin	Permite al usuario autenticarse en el sistema y acceder solo a los recursos que le son permitidos, según los privilegios que tiene definidos.
Atributos:	**Tipo de dato:**
BAceptar	JButton
BCancelar	JButton
Contra	JPasswordField
Usuario	JTextField
jLabel1	JLabel
jLabel2	JLabel
jPanel1	JPanel

Métodos:	**Descripción:**
Loggin(java.awt.Frame parent, boolean modal)	Constructor de la clase.

Tabla 38. Clase interfaz para guardar los aspectos técnico-tácticos del boxeador.

Nombre:	**Descripción:**
CI_AspectosTecnicos	Controla la gestión de los aspectos que caracterizan en combate al boxeador objeto de estudio.
Atributos:	**Tipo de dato:**
Combates	JComboBox
Competencias	JComboBox
DCombate	JComboBox
DCombate1	JComboBox
DCombate3	JComboBox
TGAislados1	JTextField
TGAislados2	JTextField
TGAislados3	JTextField
TGAislados4	JTextField
TGAislados5	JTextField
jButton1	jButton
jButton2	jButton
jButton3	jButton
jButton4	jButton
jButton5	jButton
jButton6	jButton
jButton7	jButton
jButton8	jButton
jButton 9	jButton
jComboBox1	jComboBox

jComboBox3	jComboBox
jComboBox 6	jComboBox
jLabel1	jLabel
jLabel10	jLabel
jLabel11	jLabel
jLabel12	jLabel
jLabel13	jLabel
jLabel14	jLabel
jLabel15	jLabel
jLabel16	jLabel
jLabel17	jLabel
jLabel2	jLabel
jLabel3	jLabel
jLabel4	jLabel
jLabel5	jLabel
jLabel6	jLabel
jLabel7	jLabel
jLabel8	jLabel
jLabel9	jLabel
jPanel1	jPanel
jPanel2	jPanel
jPanel3	jPanel
jPanel4	jPanel
jPanel5	jPanel
jPanel6	jPanel
jPanel7	jPanel
jScrollPane1	jScrollPane
jScrollPane2	jScrollPane
jScrollPane3	jScrollPane

jScrollPane4	jScrollPane
jTable1	jTable
jTable2	jTable
jTable3	jTable
jTable4	jTable
jTextField1	jTextField
jTextField2	jTextField
jTextField3	jTextField
jTextField4	jTextField
Métodos:	**Descripción:**
setBoxeador()	Permite cambiar un objeto de tipo boxeador por otro.

3.4.5 Diagramas de clases del diseño

A continuación se muestran los diagramas de clases del diseño, que constituyen una representación más concreta de la lógica del negocio. Teniendo en cuenta que se desea desarrollar una aplicación de escritorio, se modelan de forma estática las clases identificadas y sus relaciones.

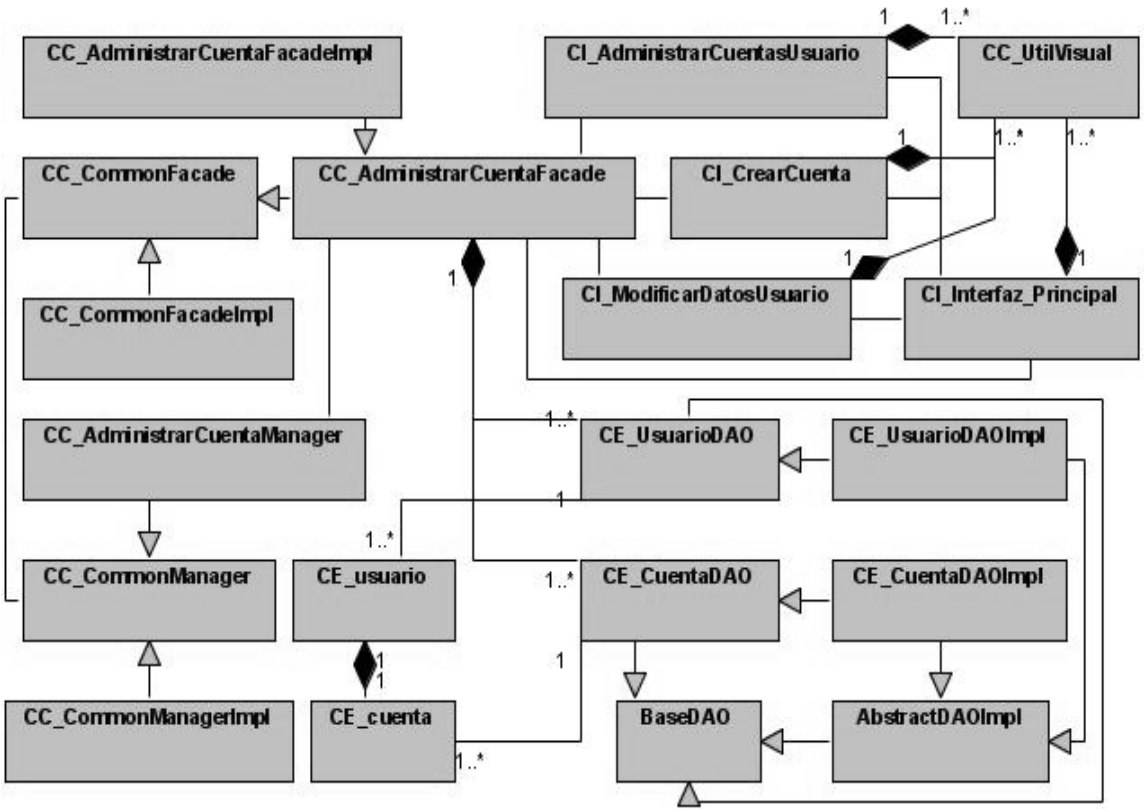


Figura 16. Diagrama de clases del diseño del caso de uso Gestionar usuario.

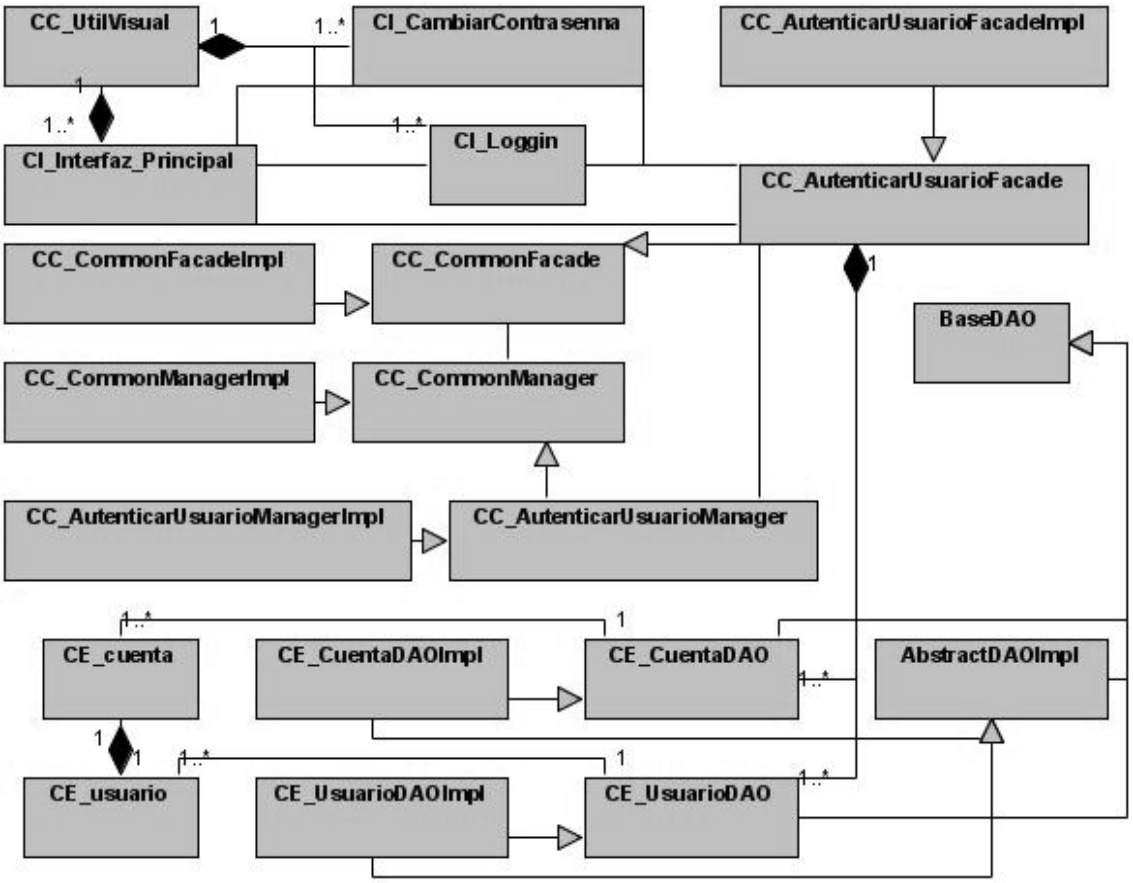


Figura 17. Diagrama de clases del diseño del caso de uso Autenticar usuario.

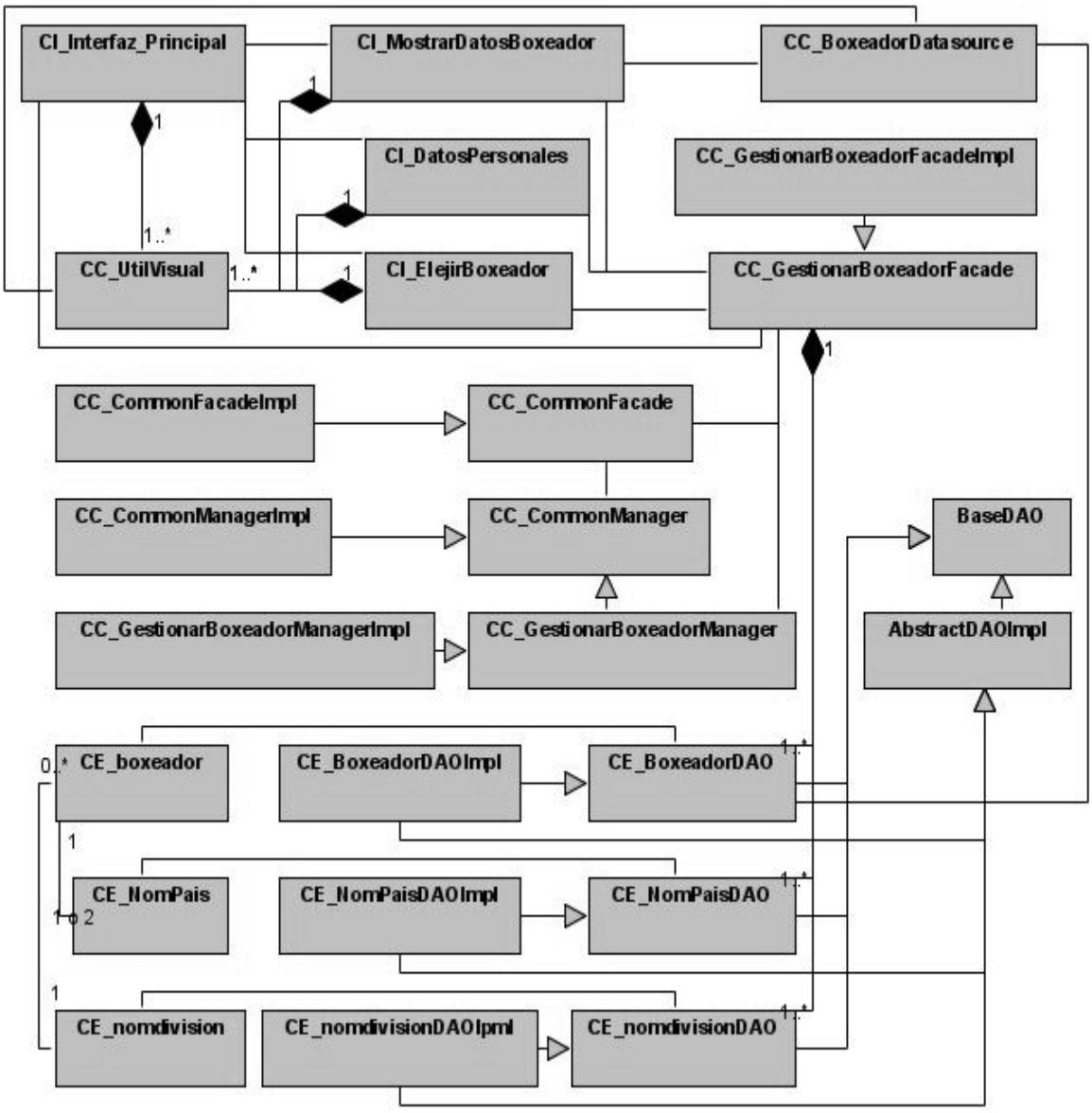


Figura 18. Diagrama de clases del diseño del caso de uso Gestionar boxeador.

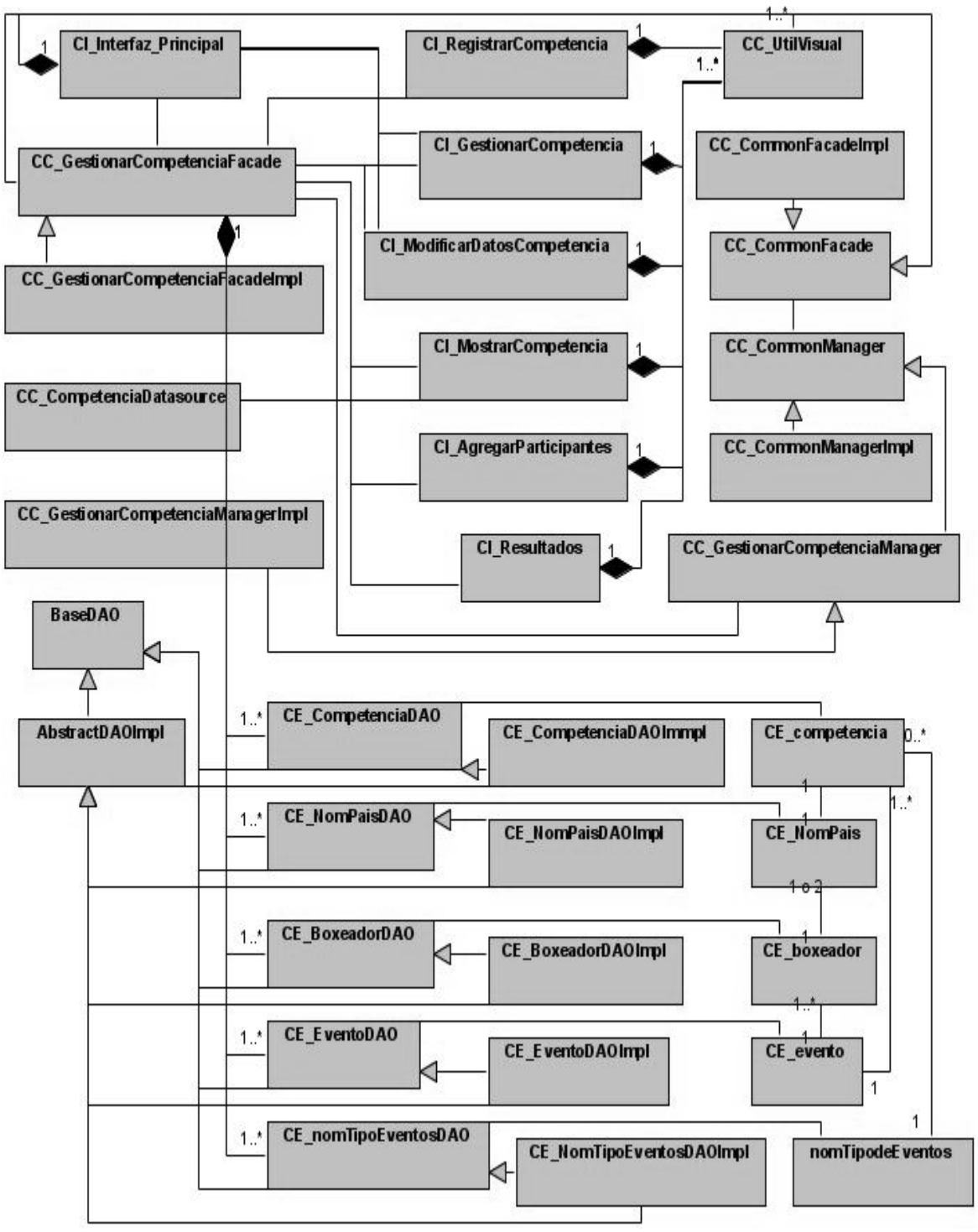


Figura 19. Diagrama de clases del diseño del caso de uso Gestionar competencia.

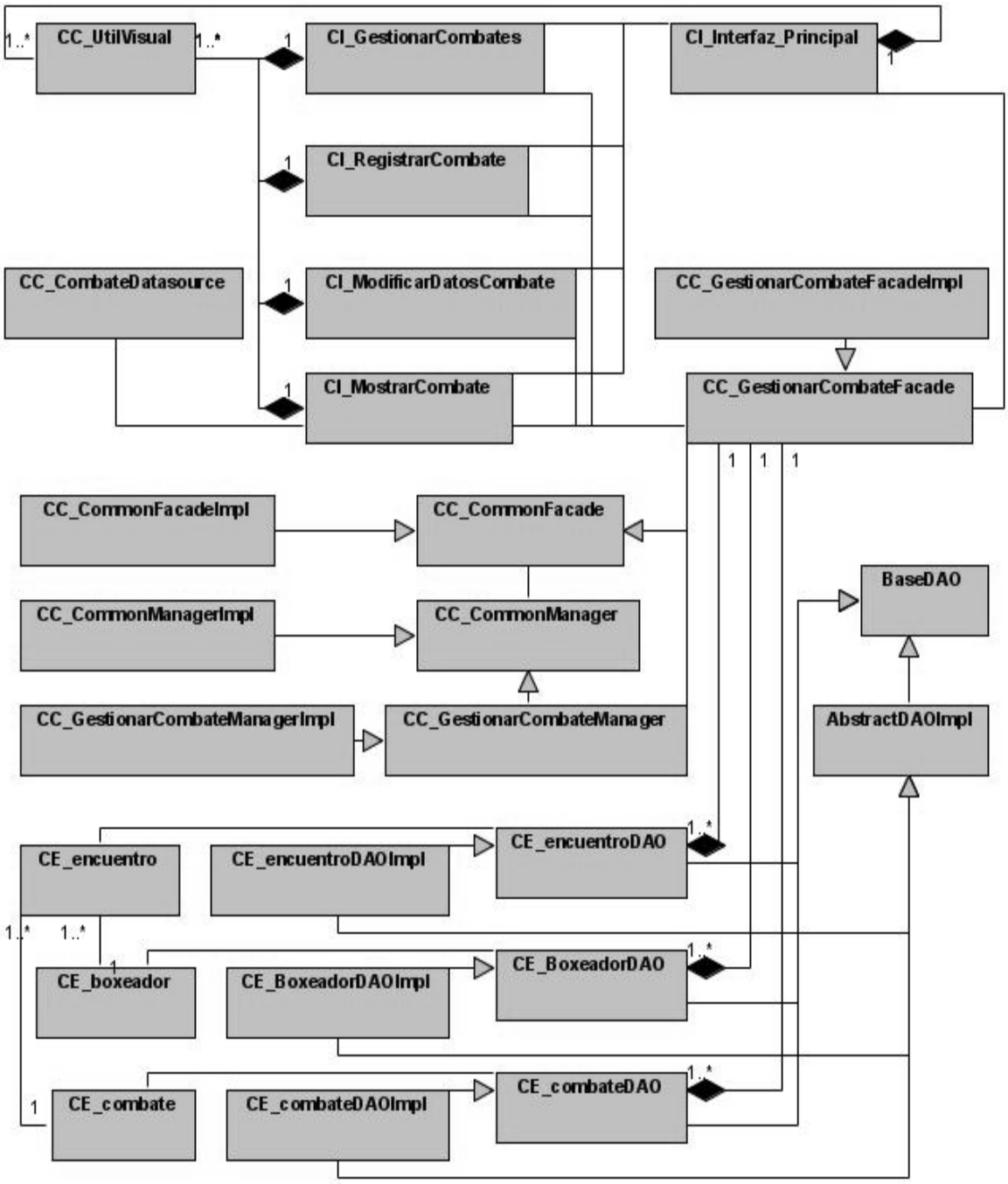


Figura 20. Diagrama de clases del diseño del caso de uso Gestionar combate.

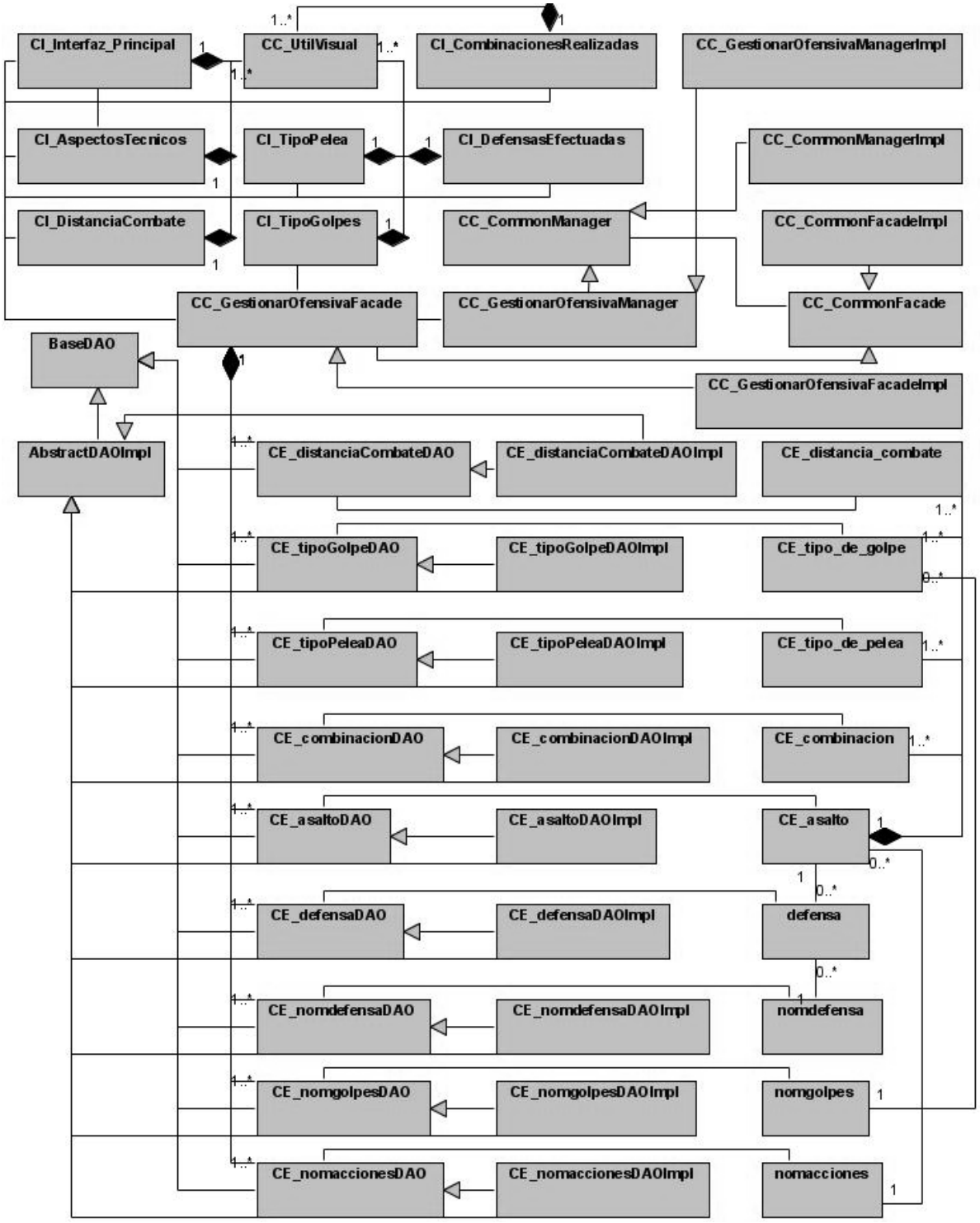


Figura 21. Diagrama de clases del diseño del caso de uso Gestionar ofensiva por asalto.

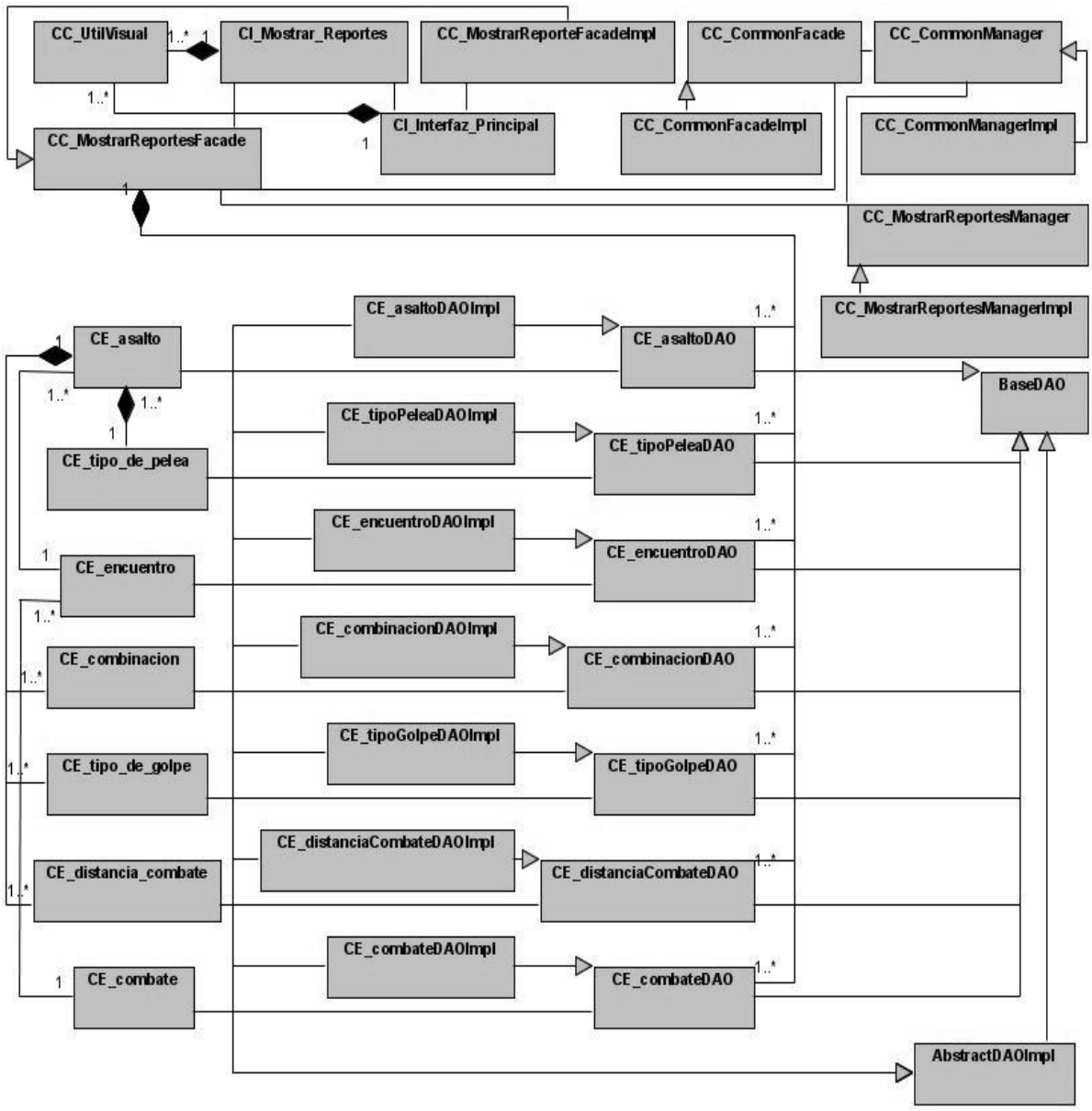


Figura 22. Diagrama de clases del diseño del caso de uso Mostrar reportes.

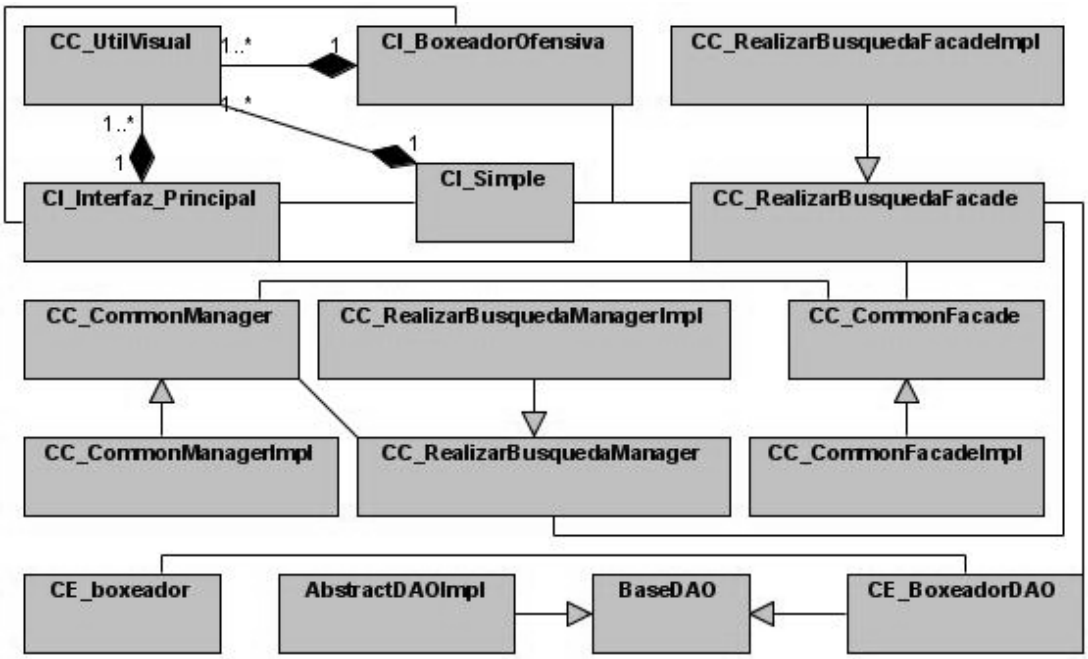


Figura 23. Diagrama de clases del diseño del caso de uso Realizar búsqueda.

3.5 Diseño de la base de datos

Para el desarrollo de la propuesta de software, es muy importante realizar un buen diseño de la base de datos que permitirá almacenar de forma segura y eficiente la información que es manipulada en el Estudio de Adversarios que se lleva a cabo en el boxeo.

3.5.1 Modelo lógico de datos.

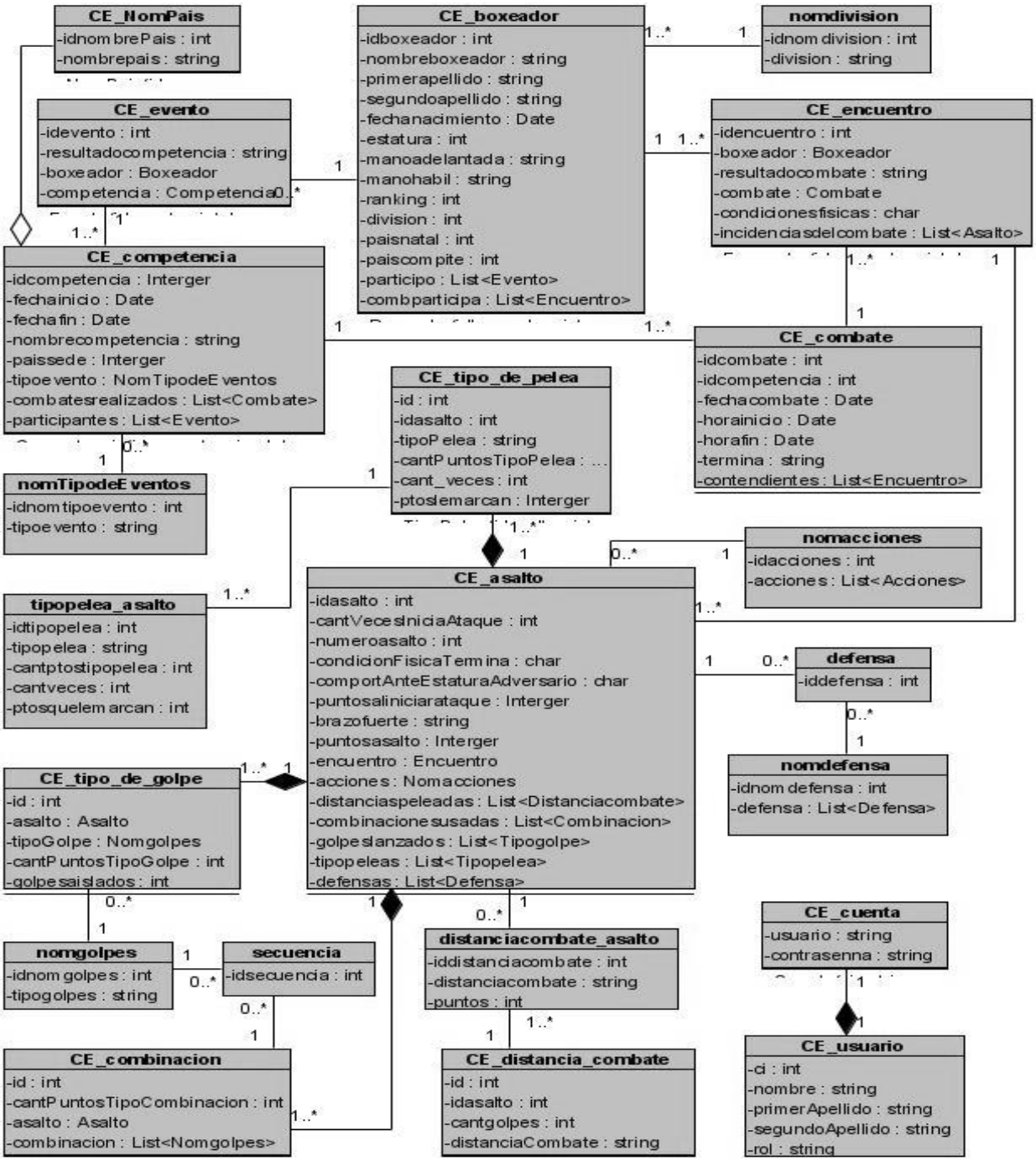


Figura 24. Diagrama de clases persistentes.

3.5.2 Modelo físico de datos.

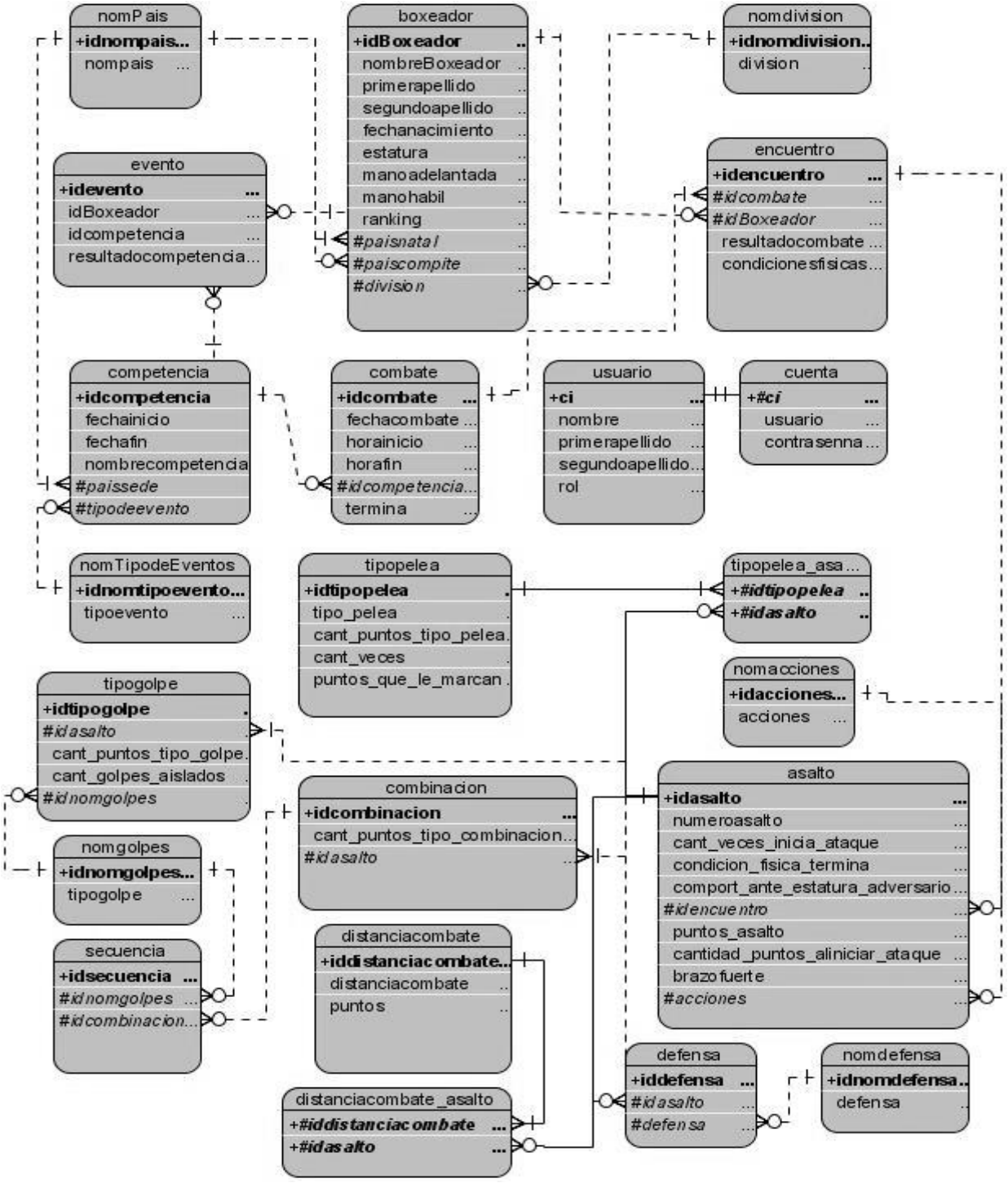


Figura 25. Modelo de datos.

3.6 Conclusiones

En este capítulo se expusieron los conceptos fundamentales asociados al flujo de trabajo de análisis y diseño. También se realizaron los diagramas de clases correspondientes al análisis y al diseño del sistema propuesto, además de conformar el diseño de la base de datos donde se almacenará la información que es manipulada durante el Estudio de Adversarios en el Boxeo. Por tanto, se ha creado la entrada apropiada y punto de partida para las actividades correspondientes a la implementación del software.

CAPÍTULO 4

IMPLEMENTACIÓN DEL SISTEMA

4.1 Introducción

Como resultado de las actividades realizadas durante el diseño de la propuesta de software, se obtiene el modelo de diseño, el cual constituye la entrada fundamental y el punto de partida para las actividades de implementación.

El flujo de trabajo de implementación que propone la metodología RUP, define un conjunto de actividades, cuyo propósito es conformar la organización del sistema, en términos de subsistemas de implementación, que deben ser organizados en capas. Además, implementa los elementos del diseño, en términos de elementos de implementación, que pueden ser ficheros, fuentes, binarios, ejecutables y otros. También permite probar como unidades, los componentes desarrollados independientemente e integrar en un sistema ejecutable, los resultados producidos por desarrolladores independientes o equipos.

En resumen, los diagramas de despliegue y componentes conforman el modelo de implementación, al describir y organizar los componentes a construir, teniendo en cuenta la dependencia existente entre los nodos físicos, en los que funcionará la aplicación.

4.2 Implementación

4.2.1 Diagrama de despliegue.

Los diagramas de despliegue muestran las relaciones físicas de los distintos nodos que componen un sistema y el reparto de los componentes sobre dichos nodos. La vista de despliegue, representa la disposición de las instancias de componentes de ejecución en instancias de nodos conectados por enlaces de comunicación. Un nodo es un recurso de ejecución tal como un computador, un dispositivo o memoria.

Por tanto, un diagrama de despliegue es un grafo de nodos unidos por conexiones de comunicación. Un nodo puede contener instancias de componentes *software*, objetos, procesos (caso particular de un

objeto). En general un nodo será una unidad de computación de algún tipo, desde un sensor a un *mainframe*. Las instancias de componentes *software* pueden estar unidas por relaciones de dependencia, posiblemente a interfaces (ya que un componente puede tener más de una interfaz). (MICHAEL and SUSANA 2001; VILAS 2001)

A continuación se muestra el diagrama de despliegue de la aplicación:

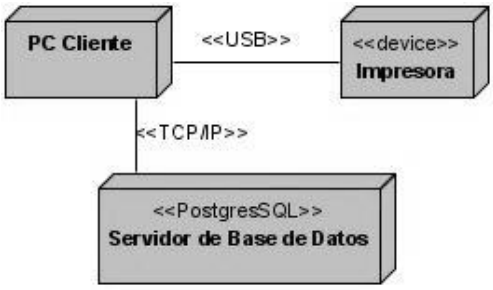


Figura 26. Diagrama de despliegue.

4.2.2 Diagrama de componentes.

Los diagramas de componentes modelan la vista estática de un sistema y muestran la organización y las dependencias lógicas entre un conjunto de componentes *software*, los que pueden ser de código fuente, ejecutables, binarios, librerías, entre otros. Por tanto, se representan mediante un grafo, que muestra un conjunto de elementos del modelo, tales como: componentes, subsistemas de implementación y sus relaciones (de compilación o de ejecución). (VILAS 2001)

A continuación se muestran los diagramas de componentes conformados, para desarrollar la propuesta de *software*:

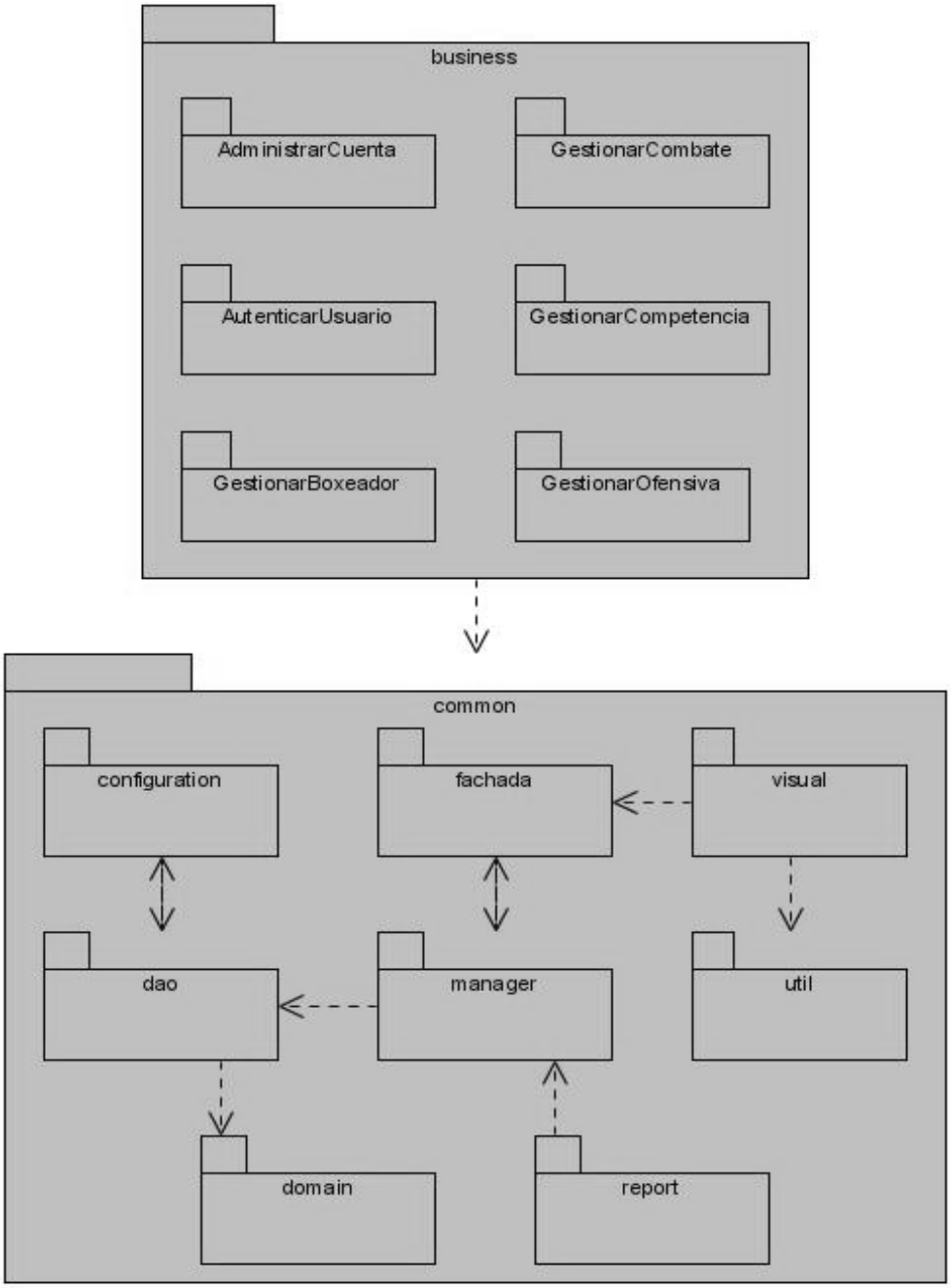


Figura 27. Diagrama de paquetes del sistema.

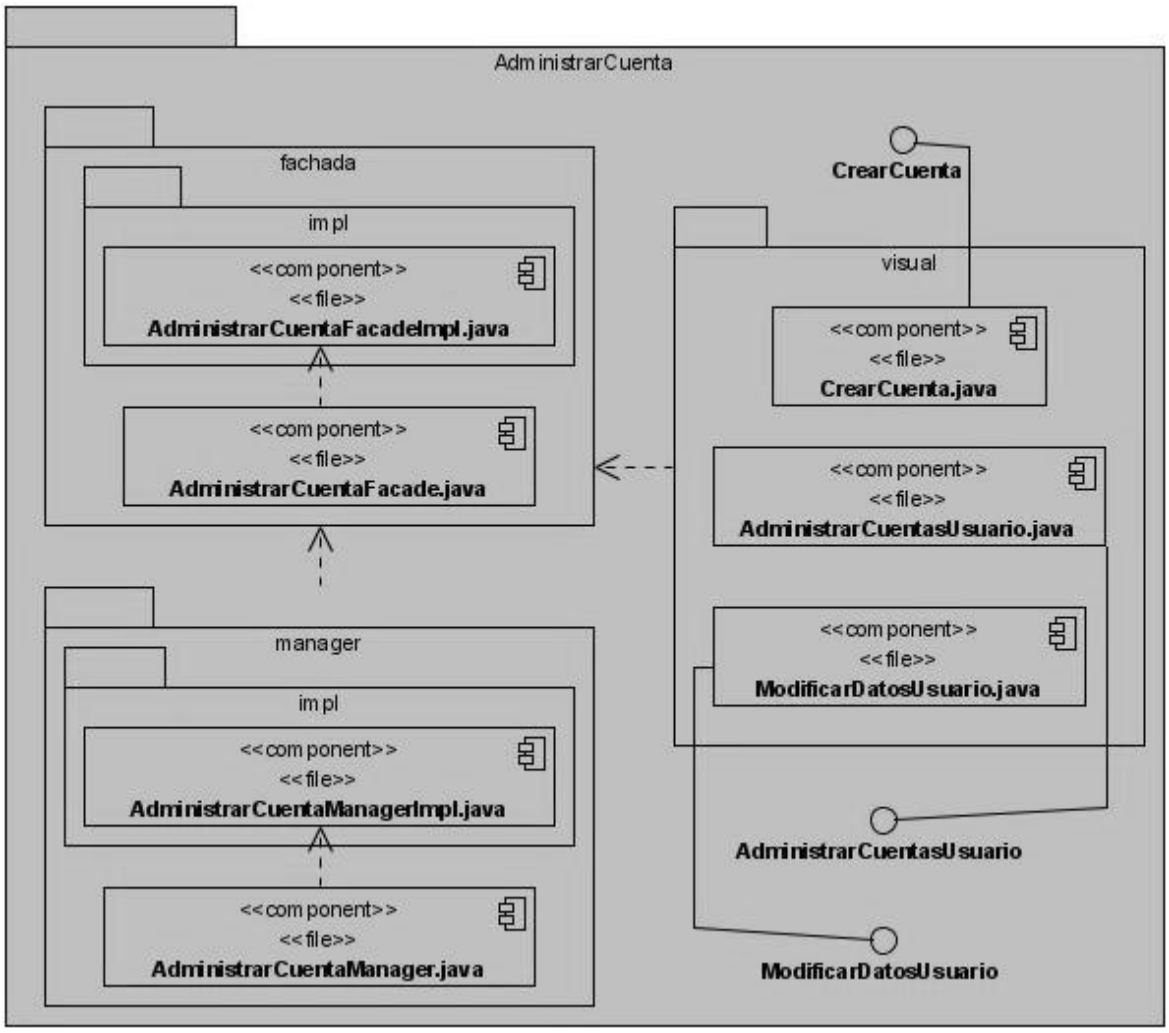


Figura 28. Diagrama de componentes del paquete Administrar Cuenta.

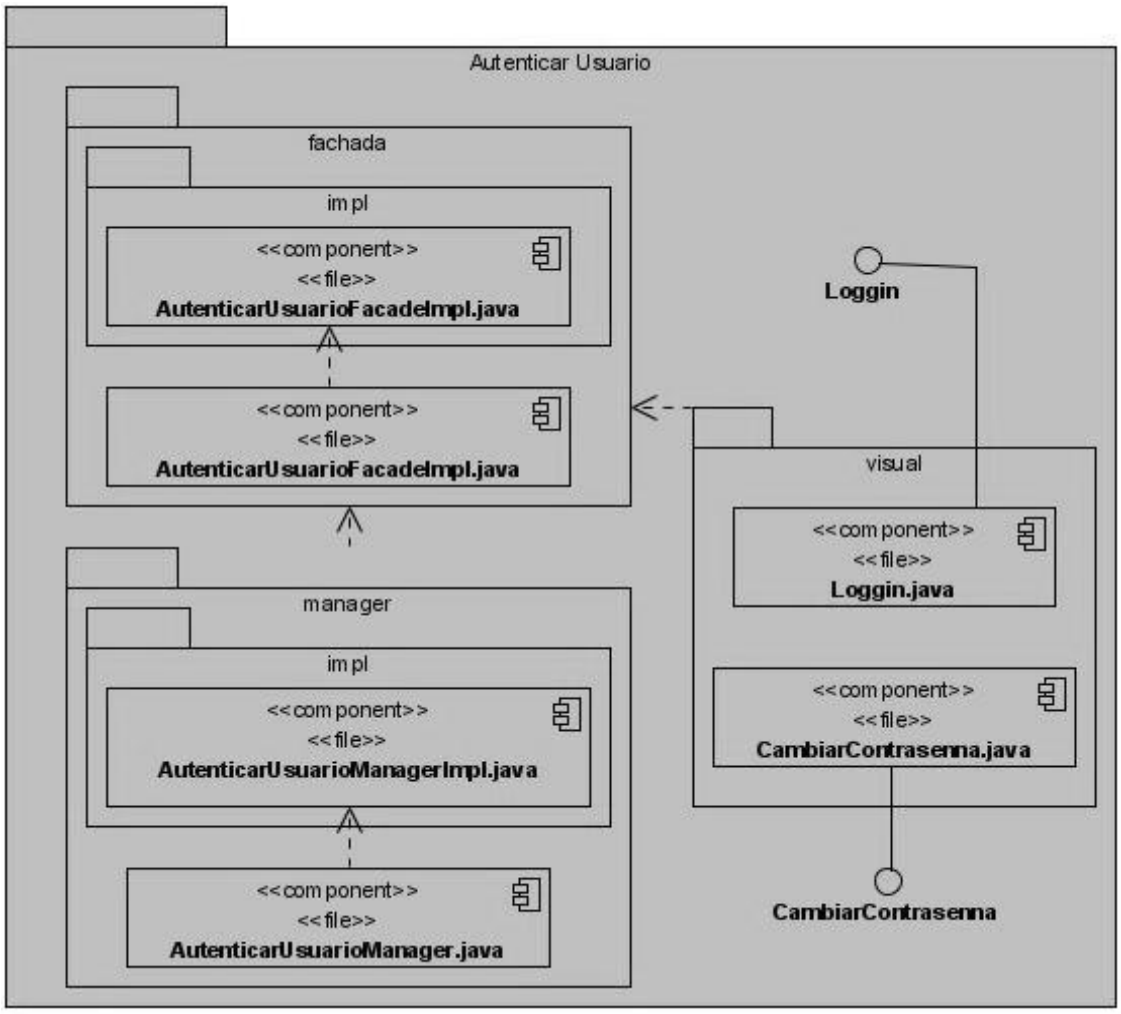


Figura 29. Diagrama de componentes del paquete Autenticar Usuario.

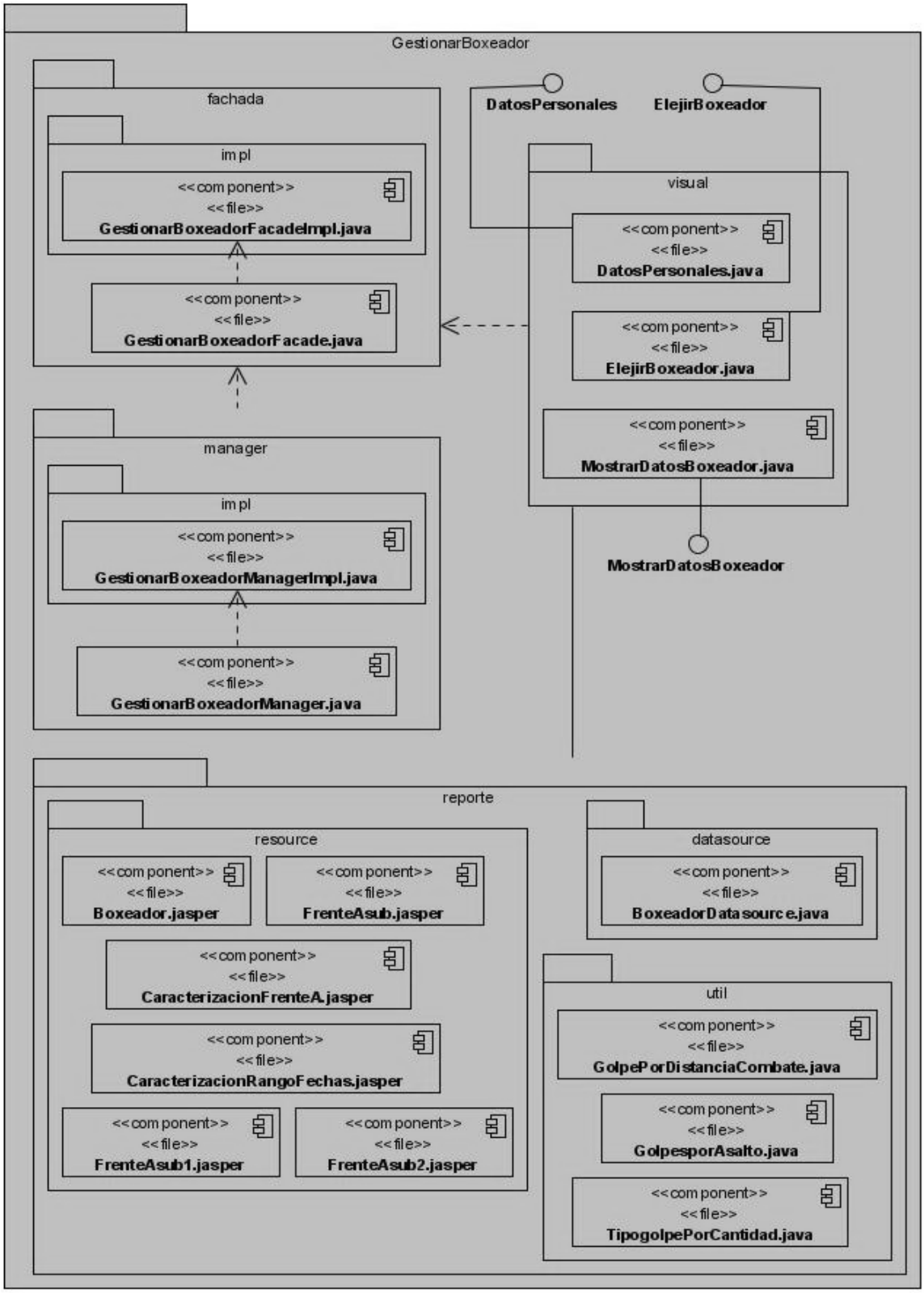

GestionarBoxeador
fachada
impl
<<component>>
<<file>>
GestionarBoxeadorFacadeImpl.java
<<component>>
<<file>>
GestionarBoxeadorFacade.java
DatosPersonales
ElejirBoxeador
visual
<<component>>
<<file>>
DatosPersonales.java
<<component>>
<<file>>
ElejirBoxeador.java
<<component>>
<<file>>
MostrarDatosBoxeador.java
MostrarDatosBoxeador
manager
impl
<<component>>
<<file>>
GestionarBoxeadorManagerImpl.java
<<component>>
<<file>>
GestionarBoxeadorManager.java
reporte
resource
<<component>>
<<file>>
Boxeador.jasper
<<component>>
<<file>>
FrenteAsub.jasper
<<component>>
<<file>>
CaracterizacionFrenteA.jasper
<<component>>
<<file>>
CaracterizacionRangoFechas.jasper
<<component>>
<<file>>
FrenteAsub1.jasper
<<component>>
<<file>>
FrenteAsub2.jasper
datasource
<<component>>
<<file>>
BoxeadorDatasource.java
util
<<component>>
<<file>>
GolpePorDistanciaCombate.java
<<component>>
<<file>>
GolpesporAsalto.java
<<component>>
<<file>>
TipogolpePorCantidad.java

Figura 30. Diagrama de componentes del paquete Gestionar Boxeador.

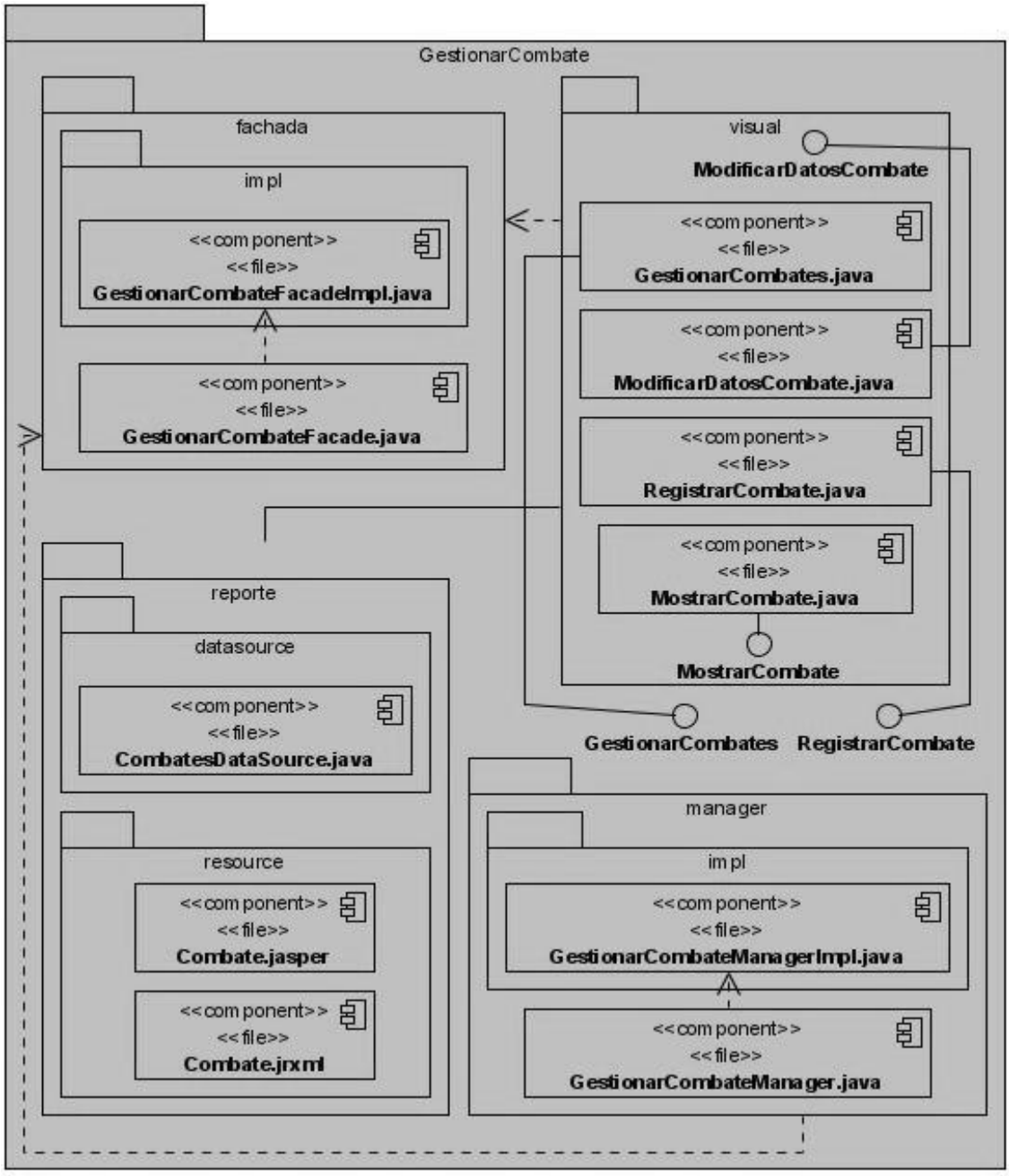


Figura 31. Diagrama de componentes del paquete Gestionar Combate.

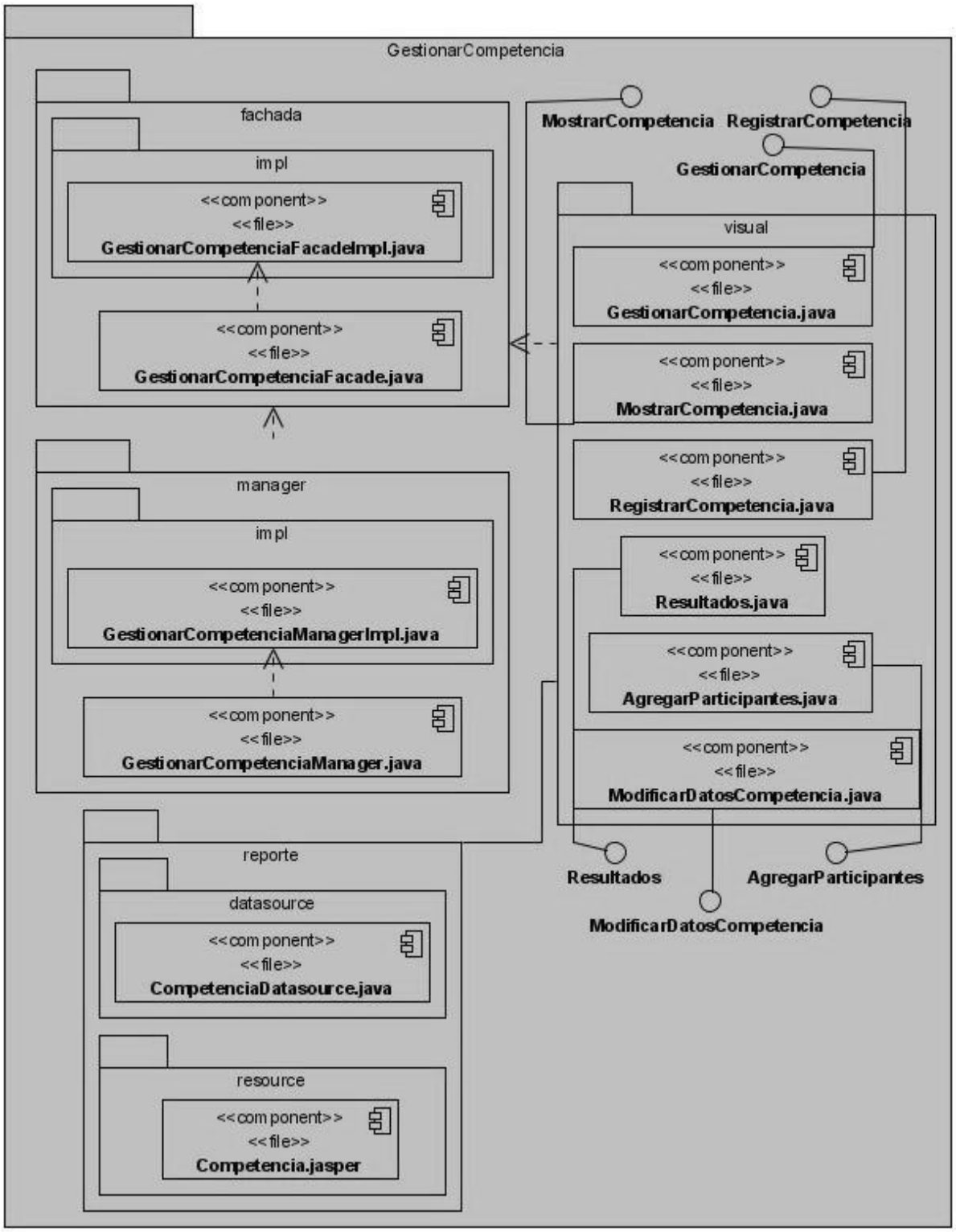

GestionarCompetencia
fachada
impl
<<component>>
<<file>>
GestionarCompetenciaFacadeImpl.java
<<component>>
<<file>>
GestionarCompetenciaFacade.java
manager
impl
<<component>>
<<file>>
GestionarCompetenciaManagerImpl.java
<<component>>
<<file>>
GestionarCompetenciaManager.java
reporte
datasource
<<component>>
<<file>>
CompetenciaDatasource.java
resource
<<component>>
<<file>>
Competencia.jasper
MostrarCompetencia
RegistrarCompetencia
GestionarCompetencia
visual
<<component>>
<<file>>
GestionarCompetencia.java
<<component>>
<<file>>
MostrarCompetencia.java
<<component>>
<<file>>
RegistrarCompetencia.java
<<component>>
<<file>>
Resultados.java
<<component>>
<<file>>
AgregarParticipantes.java
<<component>>
<<file>>
ModificarDatosCompetencia.java
Resultados
AgregarParticipantes
ModificarDatosCompetencia

Figura 32. Diagrama de componentes del paquete Gestionar Competencia.

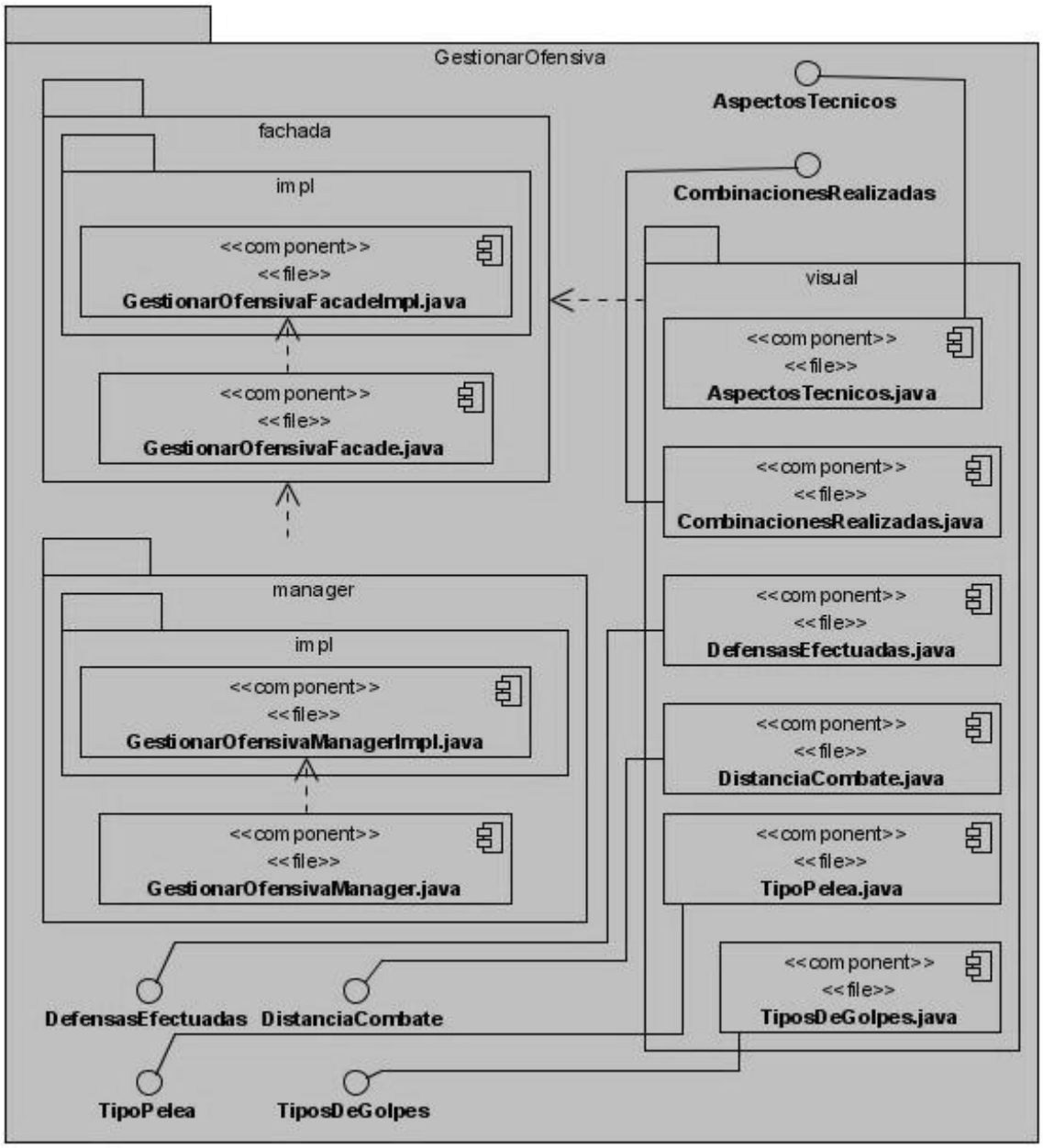


Figura 33. Diagrama de componentes del paquete Gestionar Ofensiva.

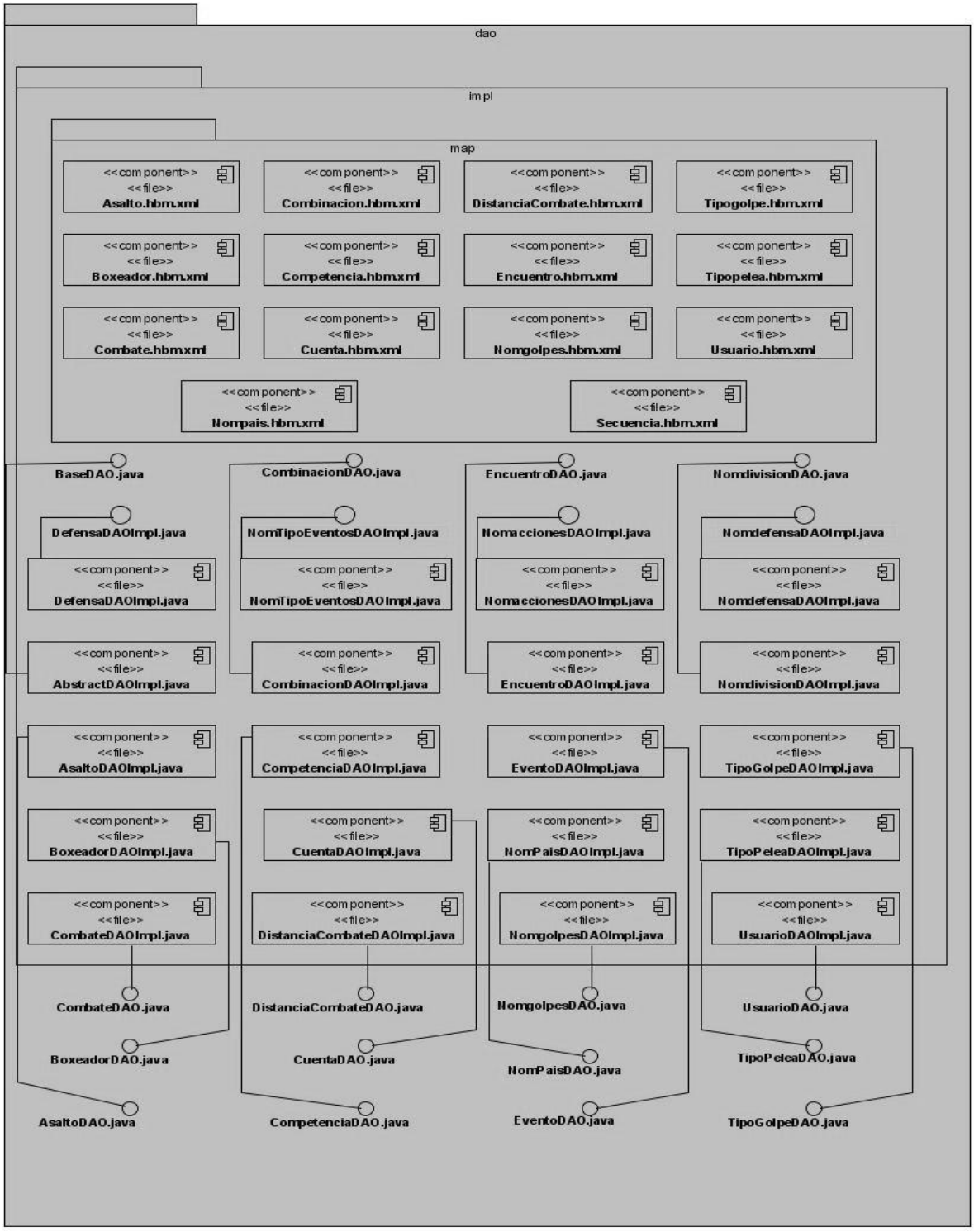


Figura 34. Diagrama de componentes del paquete dao.

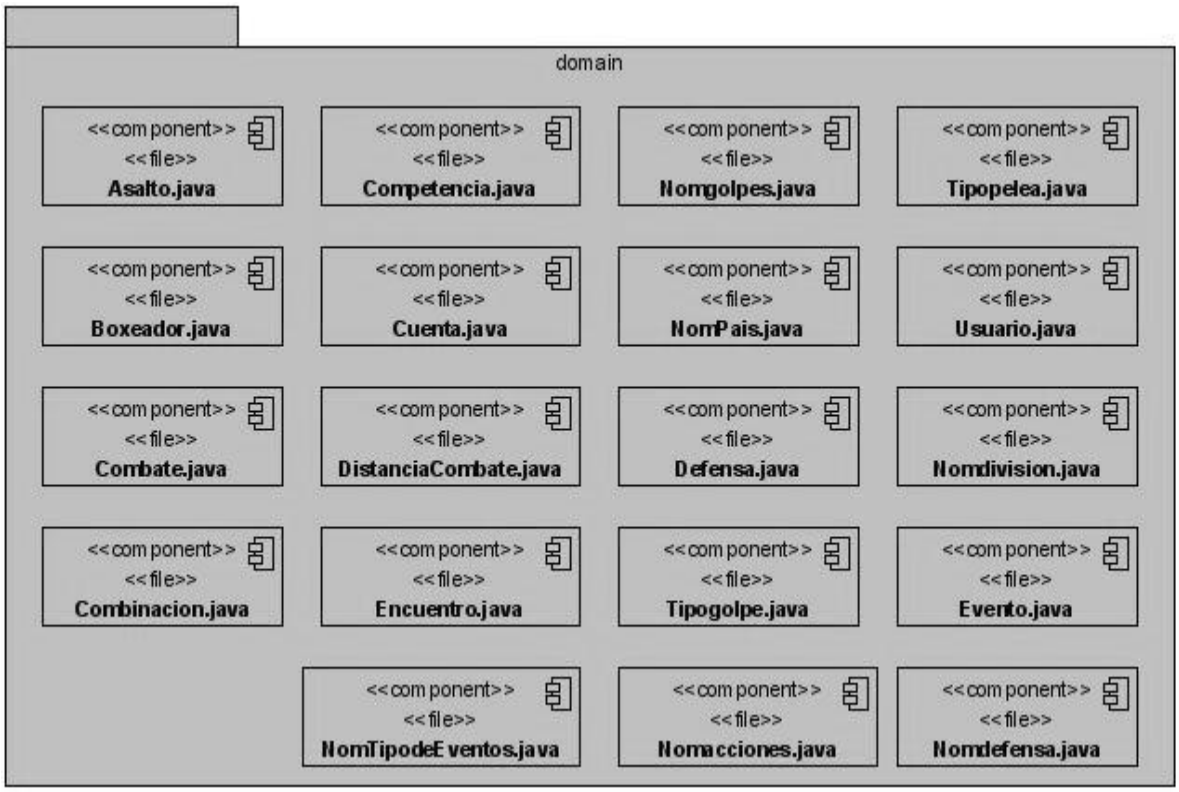


Figura 35. Diagrama de componentes del paquete domain.

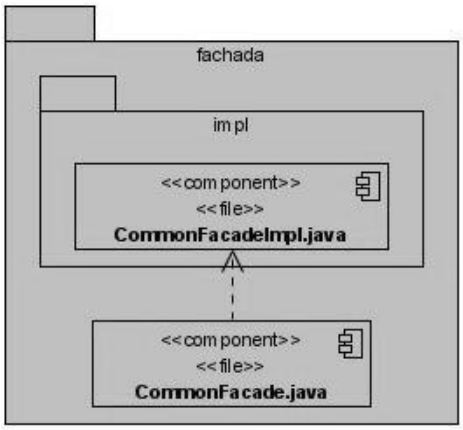


Figura 36. Diagrama de componentes del paquete fachada.

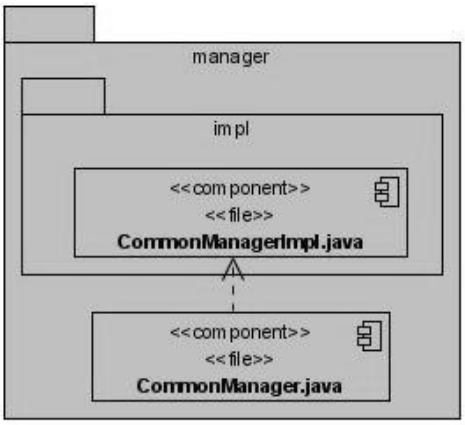


Figura 37. Diagrama de componentes del paquete manager.

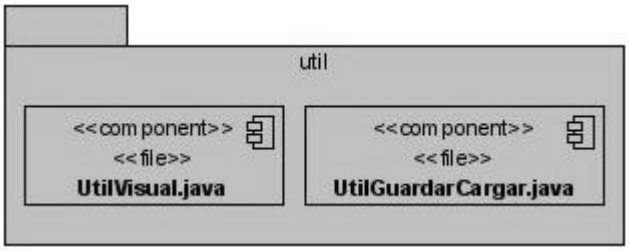


Figura 38. Diagrama de componentes del paquete útil.

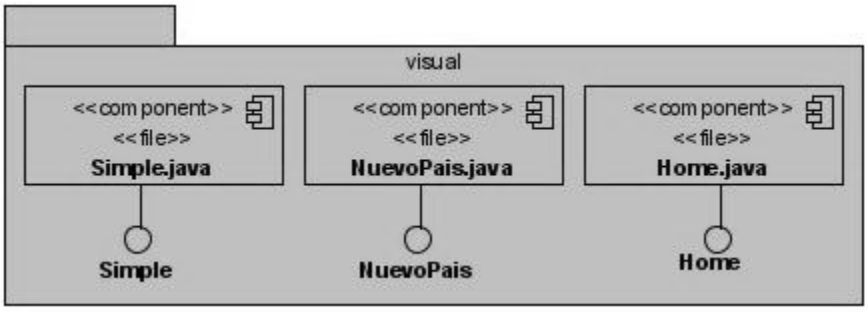


Figura 39. Diagrama de componentes del paquete visual.

4.3 Conclusiones

En este capítulo se realizó el modelo de implementación, compuesto por los diagramas de componentes y el de despliegue. Este modelo se encuentra estrechamente vinculado al lenguaje de programación, que es empleado para desarrollar la aplicación.

CONCLUSIONES

Durante el desarrollo del Sistema Informatizado para gestionar el Estudio de Adversario en el Boxeo, se realizó el diseño teórico de la investigación, lo cual permitió planificar, organizar y ejecutar la misma, de manera eficiente. También se determinó la metodología, el lenguaje y las herramientas más apropiadas para desarrollar la propuesta de software, siendo seleccionado el *Rational Unified Process* (RUP) como metodología de desarrollo, el *Visual Paradigm Enterprise Edition*, como herramienta CASE, el *Java* como lenguaje de programación, además los Entornos de Desarrollo Integrados *(IDEs)* empleados fueron el *iReport,* para generar los reportes y el *Eclipse* y el *NetBeans* para implementar el sistema. Además, se describieron las características fundamentales que debería poseer la aplicación y se realizaron las actividades correspondientes al análisis y diseño, e implementación del sistema propuesto.

Por tanto, se desarrolló un software capaz de automatizar los procesos correspondientes al Estudio de Adversarios en el Boxeo, dando solución de esta manera a los problemas presentados por los entrenadores del Equipo Nacional, durante la realización del mismo.

RECOMENDACIONES

Con el propósito de enriquecer el software desarrollado, se recomienda agregar las siguientes funcionalidades:

- Desarrollar la ayuda del sistema.
- Implementar la búsqueda avanzada, sobre los datos del boxeador que es objeto de estudio.
- Implementar la búsqueda simple y avanzada, sobre los datos de las competencias y los combates que son objeto de estudio.
- Visualizar los reportes con la información seleccionada por el usuario, es decir, se debe brindar al usuario la posibilidad de escoger los datos que serán generados en el reporte, de forma aleatoria**.**
- Facilitar la gestión de las grabaciones de los combates objeto de estudio, con el propósito de guardar, localizar y reproducir el video deseado por el usuario.
- Realizar la simulación de un combate, entre dos boxeadores seleccionados por el usuario, para estimar el comportamiento y la probabilidad de vencer del boxeador objeto de estudio. Se debe tener en cuenta los datos almacenados en la aplicación.

REFERENCIAS BIBLIOGRÁFICAS

ANÓNIMO. *Modelo Vista Controlador*, 2009. [Disponible en: http://es.wikipedia.org/wiki/Modelo_Vista_Controlador

AUTORES, C. D. *Bienvenido a NetBeans y a www.netbeans.org*, 2009a. [Disponible en: http://www.netbeans.org/community/articles/welcome-template_es.html

---. *Paradigma visual para UML (Plataforma Java)*, 2008. [Disponible en: http://www.freedownloadmanager.org/es/downloads/Paradigma_Visual_para_UML_%5Bcuenta_de_Plataforma_de_Java_14715_p/

---. *Paradigma Visual para UML Community Edition*, 2009b. [Disponible en: http://www.visual-paradigm.com/product/vpuml/communityedition.jsp

--- Paradigma Visual para UML Enterprise Edition, 2009c. [Disponible en: http://www.freedownloadmanager.org/es/downloads/Paradigma_Visual_para_UML_%5Bcuenta_de_Plataforma_de_Java_14715_p/

---. *Rational Rose*, 2009d. [Disponible en: http://www.ibm.com/developerworks/rational/products/rose/

--- Sistema Gestor de Base de Datos PostgreSQL, 2009e.

BETTYE ROSE CONNELL, M. J., RON MACE, JIM MUELLER, ABIR MULLICK, ELAINE OSTROFF, JON SANFORD, ED STEINFELD, MOLLY STORY, Y GREGG VANDERHEIDEN. *Principios del Diseño Universal o Diseño para Todos*, 1997. [Disponible en: http://www.sidar.org/recur/desdi/usable/dudt.php

GRACIA, J. *Patrones de diseño. Diseño de Software Orientado a Objetos*, 2005. [Disponible en: http://www.ingenierosoftware.com/analisisydiseno/patrones-diseno.php

JUAREZ, A. *DAO: Data Access Object*, 2008. [Disponible en: http://www.alfrek.net/blog/2008/07/dao-data-access-object/

LAGO, R. *Patrón "Data Access Object"*, 2007. [Disponible en: http://www.proactiva-calidad.com/java/patrones/DAO.html

MENDEZ, J. *Las tendencias en los lenguajes de programación*, 2008. [Disponible en: http://www.monografias.com/trabajos/tendprog/tendprog.shtml

MICHAEL, M. H. H. and Q. L. N. SUSANA. *ANALISIS Y DISEÑO DE SISTEMAS II. Diagrama de Despliegue*, 2001. [Disponible en: http://74.125.47.132/search?q=cache:sYgM_hr-g50J:virtual.usalesiana.edu.bo/web/practica/archiv/despliegue.doc+ANALISIS+Y+DISE%C3%91O+DE+SISTEMAS+II.+Diagrama+de+Despliegue+%2B+Susana&cd=1&hl=es&ct=clnk&gl=cu&lr=lang_es

RUMBAUGH, J.; I. JACOBSON*, et al.* El Lenguaje Unificado de Modelado. Manual de Referencia, 1999.

--- El Proceso Unificado de Desarrollo de Software, 2000.

SAGARRA, D. A.; D. P. L. DÍAZ*, et al. ¿Por qué la categoría Escuela Cubana de Boxeo?. Una respuesta desde una visión académica* 2007 [Disponible en: http://www.efdeportes.com/efd104/escuela-cubana-de-boxeo.htm

VILAS, A. F. *Diagrama de Componentes* 2001. [Disponible en: http://tvdi.det.uvigo.es/~avilas/UML/node49.html

ZAPATA, S. B. G. *Aplicaciones distribuidas. Para Principiantes.*, 2009. [Disponible en: http://www.slideshare.net/soreygarcia/aplicaciones-distribuidas-presentation

ZUAZO, A. *utiliusVS*, 2003. [Disponible en: http://balonmano.mforos.com/142972/5272139-utilius-vs-edicion-de-video/

BIBLIOGRAFÍA

ANÓNIMO. *Bean*, 2008a. [Disponible en: http://es.wikipedia.org/wiki/Bean

---. *Características IReport*, 2009a. [Disponible en: http://www.programa.us/descargar/peliculas/caracteristicas-ireport/

---. *Eclipse (software)*, 2009b. [Disponible en: http://es.wikipedia.org/wiki/Eclipse_(software)

---. *El lenguaje C++*, 2008b. [Disponible en: http://www.zator.com/Cpp/E1_2.htm

---. *Guía de inicio para IReport*, 2008c. [Disponible en: http://www.javamexico.org/blogs/gabo/guia_de_incio_para_ireport

--- Hibernate 2009c. [Disponible en: http://mundogeek.net/archivos/2007/01/27/hibernate/

---. *JFreeChart*, 2006a. [Disponible en: http://www.freedownloadmanager.org/es/downloads/JFreeChart_3478_p/

---. *Modelo Vista Controlador*, mayo 2009. [Disponible en: http://es.wikipedia.org/wiki/Modelo_Vista_Controlador

---. *NetBeans IDE 4.0 Beta*, 2009d. [Disponible en: http://www.ciao.es/Opiniones/NetBeans_IDE_4_0_Beta__418147

---. *Patrón "Modelo-Vista-Controlador"*, 2009e. [Disponible en: http://www.proactiva-calidad.com/java/patrones/mvc.html

---. *Rational Rose Enterprise* 2009f. [Disponible en: http://www.rational.com.ar/herramientas/roseenterprise.html

---. *Ventajas y desventajas del Java* 2006b. [Disponible en: http://meetingjava.blogspot.com/2006/08/ventajas-y-desventajas-del-java.html

AUTORES, C. D. *Rational Rose Enterprise*, 2009a. [Disponible en: http://www-142.ibm.com/software/dre/ecatalog/detail.wss?locale=es_ES&synkey=M221280M46834Z27

---. *UtiliusVS*, 2005. [Disponible en: http://www.ansa-video.de/esp/soft01.htm

---. *Visual Paradigm for UML*, 2009b. [Disponible en: http://www.visual-paradigm.com/product/vpuml/

BARBONE, V. A. G. *XP: Extreme Programming. (Programación Extrema)*, 2009. [Disponible en: http://iie.fing.edu.uy/~nacho/blandos/seminario/XProg1.html

BENDAHAN, M. *Proceso de desarrollo de software*, 2009. [Disponible en: http://www.monografias.com/trabajos5/desof/desof.shtml

BERROCAL, A. J. *Java vs C#: La licencia GPL vs los riesgos de ECMA en Mono y .NET*, noviembre 2007. [Disponible en: http://pintucoperu.wordpress.com/2007/11/26/la-licencia-gpl-de-java-vs-mono-y-net/

EDISONCOR. *Como crear una aplicación de escritorio con NetBeans*, abril 2008. [Disponible en: http://edisoncor.wordpress.com/2008/04/04/como-crear-una-aplicacion-de-escritorio-con-netbeans/

ENRÍQUEZ, A. M. B. *El desarrollo de sistemas de información empleando el lenguaje de modelado unificado UML*, 2009. [Disponible en: http://www.monografias.com/trabajos16/lenguaje-modelado-unificado/lenguaje-modelado-unificado.shtml

GARCÍA, F. U. P. *El Lenguaje C++* 2006. [Disponible en: http://www.articulandia.com/premium/printable.php

GONZALO GÉNOVA, J. M. F., JUAN LLORENS *Evaluación de herramientas CASE para UML*, 2004 [Disponible en: http://74.125.47.132/search?q=cache:ioJ-6ILKamkJ:www.procuno.com/users/taller/Presentaciones/PresentacionIEG-UC3.ppt+Evaluaci%C3%B3n+de+herramientas+CASE+para+UML&cd=2&hl=es&ct=clnk&gl=cu&lr=lang_es

GRACIA, J. *Patrones de diseño. Diseño de Software Orientado a Objetos*, mayo 2005. [Disponible en: http://www.ingenierosoftware.com/analisisydiseno/patrones-diseno.php

JOHN, S. *¿Cuáles son las principales características del C++ y qué librerías utiliza?*, 2007. [Disponible en: http://mx.answers.yahoo.com/question/index?qid=20070811200816AAA6CvJ

LAGO, R. *Patrón "Data Access Object"*, Abril 2007. [Disponible en: http://www.proactiva-calidad.com/java/patrones/DAO.html

LARA, J. I. S. *Elegir un IDE para Java* junio 2007. [Disponible en: http://iiso.blogspot.com/2007/06/elegir-un-ide.html

OCAÑA, A. L. O. *¿Cómo investigar en educación?*, 2008. [Disponible en: http://www.monografias.com/trabajos13/artinves/artinves.shtml

OCHOA, A. B. *Métodos*, 2009. [Disponible en: http://www.monografias.com/trabajos11/metods/metods.shtml

PAVÓN, E. L. *Visual Paradigm, una herramienta de lo más útil.* , abril 2007. [Disponible en: http://slion2000.blogspot.com/2007/04/visual-paradigm-una-herramienta-de-lo.html

QUIRINO, A.; E. RAMÍREZ*, et al. Programación Extrema (XP)* 2008. [Disponible en: http://74.125.47.132/search?q=cache:r_DiuOR9u6kJ:homepages.mty.itesm.mx/al1031357/XP.ppt+XP:+Extreme+Programming+(Programaci%C3%B3n+Extrema).&cd=2&hl=es&ct=clnk&gl=cu&lr=lang_es

RODRÍGUEZ, G. J. S. *Evolución de los lenguajes de programación. ¿Por qué cambiarse a la Programación Orientada a Objetos?*, 2006. [Disponible en: http://74.125.47.132/search?q=cache:eBZt8kSnEtIJ:www.portalunse.com.ar/component/option,com_docman/task,doc_download/gid,70/Itemid,72/+Evoluci%C3%B3n+de+los+lenguajes+de+programaci%C3%B3n+%C2%BFPor+qu%C3%A9+cambiarse+a+la+Programaci%C3%B3n+Orientada+a+Objetos%3F&cd=3&hl=es&ct=clnk&gl=cu&lr=lang_es

SAGARRA, D. A. Experiencia de la Escuela Cubana con el Boxeo Cubano, 1999.

SIERRA, M. *Trabajando con Visual Paradigm for UML*, 2008. [Disponible en: http://www.google.com.cu/search?hl=es&q=Trabajando+con+Visual+Paradigm+for+UML&btnG=Buscar&meta=lr%3Dlang_es

SUSANA, M. H. H. M. Q. L. N. *Diagrama de Despliegue*, 2001. [Disponible en: http://74.125.47.132/search?q=cache:sYgM_hr-g50J:virtual.usalesiana.edu.bo/web/practica/archiv/despliegue.doc+ANALISIS+Y+DISE%C3%91O+DE+SISTEMAS+II.+Diagrama+de+Despliegue&cd=1&hl=es&ct=clnk&gl=cu&lr=lang_es

TATYANA. *Sistema de gestión de base de datos.*, 2007. [Disponible en: http://html.rincondelvago.com/sistemas-gestores-de-bases-de-datos_2.html

VILAS, A. F. *Diagrama de Componentes*, 2001a. [Disponible en: http://tvdi.det.uvigo.es/~avilas/UML/node49.html

---. *Diagrama de Despliegue* 2001b. [Disponible en: http://tvdi.det.uvigo.es/~avilas/UML/node50.html

ZAPATA, L. G. Y. *Herramientas de desarrollo de Ingeniería de Software para Linux*, 2005. [Disponible en: http://hugolopez.phi.com.co/docs/download/file=Giraldo-Zapata-Herramientas%20de%20ISW.pdf,_id=17

GLOSARIO DE TÉRMINOS

Adversario objeto de estudio: Adversarios que tienen mayor ranking internacional, por lo que son titulares de olimpiadas, o los que la dirección pedagógica de la Escuela Cubana de Boxeo, junto a los entrenadores, decidan que deben estudiar, dado el caso que hayan vencido en competencia a algún deportista de Cuba, en uno o más eventos importantes.

Calendario Competitivo: Documento donde se establecen las prioridades de participación competitiva de los deportistas que conforman el Equipo Nacional de Boxeo.

Cantidad de Combinaciones: Secuencia de más de un golpe.

Contraataque: El adversario golpea, el boxeador se defiende, acto seguido golpea y es efectivo.

Corta: Distancia de combate en que los boxeadores se encuentran cuerpo a cuerpo.

Combinaciones: Está conformada por un conjunto de golpes que un boxeador puede propinar a su adversario. Existen muchos tipos de combinaciones, por lo que no es posible registrar el nombre de todas, teniendo en cuenta este aspecto, se debe ofrecer la posibilidad de ingresar el nombre de determinada combinación según el boxeador la realice.

Condiciones físicas: Se refiere al estado en que el boxeador termina el asalto o el combate, puede ser: bien, regular o mal.

Culminación del Asalto: En este aspecto se especifica si el boxeador perdió o ganó el combate y cómo fue.

Documentación: Documentos donde se recoge (mediante la observación de los combates), información técnico-táctica sobre el desempeño del adversario objeto de estudio.

Entrenador: Es un cuadro pedagógico que labora en la Escuela Cubana de Boxeo. Utiliza la información recopilada por los expertos, mediante el estudio de determinados adversarios, para trazar planes de entrenamiento adecuados a los boxeadores que entrena.

Estudio de Adversario: Estrategia que utilizan los entrenadores de la Escuela Cubana de Boxeo, para obtener información técnico-táctica sobre los adversarios objeto de estudio. Tiene como propósito lograr una mejor preparación física y psicológica de los boxeadores cubanos, con vista a que obtengan la victoria en competencias.

Experto: Entrenadores que son designados para viajar al exterior a realizar el Estudio de Adversarios de uno o más boxeadores de un continente en específico. Dichos entrenadores se seleccionan por sus características fisiológicas, sus conocimientos sobre el idioma de los países que va a visitar y sus conocimientos técnico-tácticos sobre el boxeo.

Frecuencia: Tiempo de pausa que hace un boxeador entre el lanzamiento de un golpe y otro**.**

Golpe Aislado**:** Un solo golpe.

Golpes Efectivos**:** Coincidencia de más de 3 jueces (de los 5 que observan la pelea) de que el golpe es válido, es decir, puntos marcados.

Jefe de Cátedra**:** Cuadro pedagógico de la Escuela Cubana de Boxeo, que dirige el proceso de Estudio de Adversarios, en coordinación con el jefe de colectivo técnico.

Jefe de Colectivo Técnico**:** Cuadro pedagógico de la Escuela Cubana de Boxeo, que dirige el proceso de Estudio de Adversarios, en coordinación con el jefe de cátedra.

KO**:** Cuando el boxeador recibe un golpe y no se recupera en el tiempo establecido.

Larga**:** Distancia de combate donde el boxeador alcanza al adversario dando un paso.

Media**:** Distancia de combate donde un boxeador con la extensión de los brazos hace impacto en el adversario.

Probabilidad de Combinaciones: La combinación que más empleó. El total de veces que empleó esa combinación, sobre el total de combinaciones. Dicho resultado determina la probabilidad de que la utilice en un combate.

Probabilidad de Golpes: Es el golpe que más lanzó el boxeador en el combate. Se determina por el total de veces que lo lanzó, sobre el total de golpes. Lo cual determina la probabilidad de que lance ese golpe.

Promedio de Combinaciones por asalto: Es la cantidad de combinaciones que empleó el boxeador durante el combate, entre la cantidad de asaltos.

Promedio de Golpes por asalto: Es la cantidad total de golpes que lanzó durante el combate el boxeador, dividido entre la cantidad de asaltos.

Promedio de Tiempo de Pausa por asalto: Es la suma de los tiempos de pausa efectuados por el boxeador objeto de estudio durante el combate, entre el número de asaltos.

Ranking: Lugar que otorga la Federación Internacional a un boxeador según su división y los resultados que ha obtenido en competencias internacionales anualmente, que puede variar en la medida en que participa en dichas competencias.

Resultados de la competencia: Oro, Plata y Bronce.

Riposta: Es cuando el adversario ataca, (sin efectividad) y el boxeador golpea automáticamente de forma efectiva. Se diferencia del contraataque en que no hay defensa.

RSC: El árbitro suspende el combate.

RSCH: El árbitro suspende el combate por golpe fuerte en la cabeza.

Superioridad Técnica: Se suspende el combate por más de 15 puntos de ventaja.

Tipos de Competencias:

- Internacionales.
- Juegos Centroamericanos.
- Juegos Panamericanos.
- Mundiales.
- Olimpiadas.
- Preolímpicos.
- Torneo en Holanda.
- Match de Retadores.
- Torneo en Carabobo.

Tipos de Golpes:

- Recto:
  - Derecho a la cara.
  - Derecho al tronco.
  - Izquierdo a la cara.
  - Izquierdo al tronco.
- Swing:
  - Derecho a la cara.
  - Derecho al tronco.
  - Izquierdo a la cara.
  - Izquierdo al tronco.
- Gancho:
  - Derecho a la cara.
  - Derecho al tronco.
  - Izquierdo a la cara.
  - Izquierdo al tronco.

Video del combate**:** Grabación de un combate donde participa un adversario objeto de estudio.