00623
3

UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE CONTADURÍA Y ADMINISTRACIÓN

APLICACIÓN DE REFERENCIA PARA JÓVENES DESARROLLADORES QUE ADOPTAN LA TECNOLOGÍA J2EE.

CASO PRÁCTICO: VIDEOJUEGO EN LÍNEA

DISEÑO DE UN SISTEMA PARA UNA ORGANIZACIÓN
QUE PARA OBTENER EL TÍTULO DE:

LICENCIADO EN INFORMÁTICA
PRESENTAN:

RENATO BARAHONA NERI

ABRAHAM OMAR LUGO LEÓN

KARLA YADIRA MARROQUÍN LÓPEZ

MARÍA IVET NEGRETE GUTÍERREZ

ASESOR:
MTRA. GRACIELA BRIBIESCA CORREA

MÉXICO, D.F. 2003

A

AGRADECIMIENTOS.

BARAHONA NERI RENATO

A Dios, por todo lo que me ha dado y sobre todo por aquello que todavía no me he ganado.

A mis Padres, Susana Neri y Octavio Barahona, por todo el amor, consejos y apoyo que he recibido a lo largo de mi vida, así como por el mejor regalo que se le puede hacer a un hijo: su educación.

A mi hermana Alejandra, tíos, abuelos y primos. Gracias por estar siempre cuando los he necesitado.

A Omar, Karla e Ivet, por permitirme compartir la experiencia de trabajar a su lado durante toda la carrera, así como durante el presente proyecto. De igual forma, agradezco su apoyo, comprensión, amistad y por permitirme ser una mejor persona día con día.

Al M. en C. Gerardo León Lastra, por su amistad, confianza y apoyo a lo largo del tiempo que me ha permitido formar parte de su equipo de trabajo.

A la Mtra. Graciela Bribiesca Correa, por el apoyo brindado a lo largo del desarrollo de éste proyecto.

A la Lic. Rosario Salinas Cuellar y a todos los participantes del Programa de Becas de la DS, que me han permitido crecer tanto dentro de mi profesión como en el desarrollo de mi persona.

A la Mtra . Gudalupe Ferrer Andrade, así como a todo el personal de la Dirección General de Televisión Universitaria (TVUNAM), en especial a Margarita Macías, por toda su ayuda, y soporte para la realización de este proyecto.

A la Facultad de Contaduría y Administración, así como a todos los profesores que participaron en mi formación académica.

A la UNIVERSIDAD NACIONAL AUTONOMA DE MÉXICO.

A Alma, Ana, Ariadna, Doris, Jacqueline, Juan Carlos, Linda, Miguel, Sandra, así como todas aquellas personas que a lo largo de mi vida he tenido la fortuna de conocer.

日本でみんなどうもありがとうございます。

失敗は成功の元
[Los errores son el origen de la perfección]

AGRADECIMIENTOS.

LUGO LEÓN ABRAHAM OMAR

A Dios.

A mis padres, Abraham Enrique y Friné. Su entrega, dedicación, ejemplo, amor y confianza me hacen ser quien soy. Son las personas que más admiro y que más orgulloso me hacen sentir, son una gran motivación para hacer lo correcto.

A mi hermano, Luis por su ayuda y todos los buenos momentos.

A mis tíos, Samuel y Malena, por su apoyo, amor, consejo, y por estar conmigo en todo momento.

A mis excelentes compañeros de equipo, Renato, Karla e Ivet, por todo lo que han aportado a mi vida y a este trabajo, por demostrar que los conceptos de amistad y compañerismo no tiene límites insuperables, por ser excelentes personas, y motivarme a ser mejor.

Al M. en C. Gerardo León Lastra por su amistad, por su paciencia y apoyo, por todas las oportunidades que me ha dado de aprender, de aportar y de crecer.

A nuestra asesora Mtra. Graciela Bribiesca Correa por creer en este proyecto, por su confianza, su apoyo y sus aportaciones a nuestro trabajo.

A TVUNAM por el soporte para el presente proyecto y su equipo de desarrollo. Por la oportunidad de crecer profesionalmente, de continuar mi formación y de permitirme aportar un poco de lo mucho que he recibido de mi universidad.

A la Maestra Guadalupe Ferrer Andrade, finísima persona, por su interés a la realización del presente trabajo, la facilitación de los recursos empleados, y su apoyo y motivación a lo largo de nuestra estancia en la dependencia.

A Margarita Macías por su ayuda consejo y afecto, mismos que no solo incentivaron nuestro trabajo, sino que el conocerla, me dio una razón más para sentirme afortunado.

A la Facultad de Contaduría y Administración por enseñarme a aprender, y por la formación recibida, que hoy me permite llegar a esta etapa de mi vida.

A mis profesores y compañeros por su colaboración en mi formación profesional y personal.

A Miguel, mi amigo, por su confianza, su paciencia y complicidad y por apoyarme siempre.
A Sandra, a Ariadna, a Linda, a Jacqueline y a Ana, por su amistad incondicional, por compartir este camino de vida conmigo, por su ayuda, su comprensión; por estar ahí y por todas las experiencias de vida. Así como a todas las personas que he tenido la fortuna conocer y que no me es posible incluir.

A la UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO por el honor y orgullo de ser universitario.

AGRADECIMIENTOS

MARROQUÍN LÓPEZ KARLA YADIRA

A mi MADRE, Guadalupe López Pérez. Porque gracias a su completa dedicación, paciencia y amor, he pasado arduamente por cada una de las etapas de mi vida de estudiante para llegar a esto. Espero continuar llenando su vida de orgullo hacia las afrentas que tomaré en el futuro.

A mis compañeros de equipo, quienes han demostrado ser excelentes tanto en trabajo como en la vida personal. Agradezco su ayuda, consejo, los momentos felices y la oportunidad de ser parte de su felicidad.

A un maestro que me encontró en TVUNAM, M.en C. Gerardo León Lastra, por su paciencia y dedicación a jóvenes becarios que tienen mucho empeño en aprender. Gracias por la oportunidad de ser auxiliada y de proporcionar la misma cortesía para la institución.

A nuestra asesora Ing. Graciela Bibriesca por su confianza en el proyecto y su apoyo en el mismo.

A los compañeros del plan de becas de la Dirección de Sistemas de la Dirección General de Servicios de Cómputo Académico que fueron tanto amigos como maestros. Gracias a la formación que me brindaron he podido apoyar en los proyectos que se me asignan.

A la Lic. Rosario Salinas Cuellar, quien ha permitido que el programa de becas de DS siga promoviendo el conocimiento y la difusión del mismo.

A la Lic. Laura Liyén Galicia Peñaloza por su confianza en mí al asignarme a los proyectos de TVUNAM por parte del plan de becas de DS.

A la Maestra Guadalupe Ferrer Andrade, gracias a su confianza en el plan de becarios y a la oportunidad que nos brindó.

A Margarita Macia, gracias a su apoyo incondicional y confianza en nosotros pudimos superar muchos obstáculos.

A TVUNAM por toda la ayuda proporcionada para el desarrollo de este proyecto, su consideración para los estudiantes y encargados de áreas de sistemas.

A la Facultad de Contaduría y Administración por proporcionar las herramientas iniciales que me abrirían las puertas en un campo laboral.

A la Universidad Nacional Autónoma de México por haberme dado el privilegio de ser parte de su comunidad.

A todos los amigos y personas que creen en mi y gracias a los cuales he llegado a ser mejor persona.

AGRADECIMIENTOS.

NEGRETE GUTIERREZ MARIA IVET

A Dios, gracias por ayudarme en todo lo que sea su voluntad y gracias por las veces que dijo "no".

A mi Padre Manuel Negrete por encaminarme y darme las bases para estudiar lo que yo quería.

A mi Madre Guadalupe Gutiérrez por su apoyo incondicional, paciencia, comprensión, y guía.

A mi Hermano Emanuel Negrete por la alegria que me transmitió en los tiempos difíciles.

A mis amigos y compañeros de tesis, por permitirme compartir la experiencia de trabajar a su lado durante toda la carrera y este momento tan importante. Saben que sin ustedes este trabajo no sería igual.

A Ariadna,Ana, Jaqueline, Linda, Sandra, Miguel, Nadia y la Sra Ma. del Socorro Gurrola Mendoza y otras tantas personas que he tenido la oportunidad de conocer un simple gracias no podria ser suficiente.

Al M. en C. Gerardo León Lastra, por su amistad, confianza y apoyo a lo largo del tiempo que me ha permitido formar parte de su equipo de trabajo. Realmente es un privilegio trabajar con alguien a quien admiras.

A la Mtra. Graciela Bribiesca Correa, por el apoyo brindado a lo largo del desarrollo de éste proyecto.

A la Lic. Rosario Salinas Cuellar y a todos los participantes del Programa de Becas de la Dirección de Sistemas de DGSCA, que me han permitido formar parte de ese círculo de personas tan profesionales haciéndome crecer tanto dentro de mi profesión como en el desarrollo de mi persona.

A la Maestra Guadalupe Ferrer Andrade, quien nos apoyo en todo momento para la realización de esta tesis

A Margarita Macias quien fue una de esas personas a la que simplemente esperabas conocer. Gracias por estar ahí.

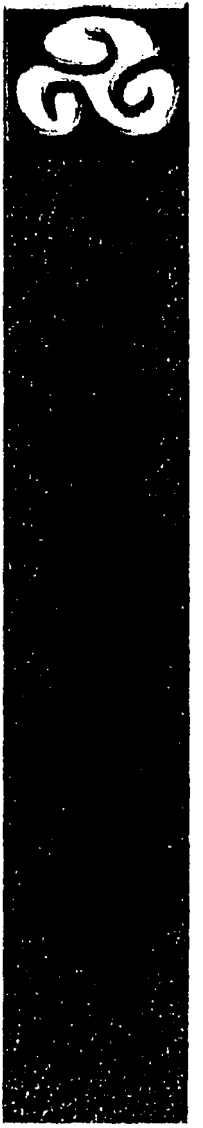
A la Dirección General de Televisión Universitaria (TVUNAM) por facilitarme las herramientas para desarrollarme profesionalmente.

A la Facultad de Contaduría y Administración, así como a todos los profesores que participaron en mi formación académica.

A la UNIVERSIDAD NACIONAL AUTONOMA DE MÉXICO por formar parte de la máxima casa de estudios. El haber entrado es algo que nunca terminare de agradecer.

"El Sol nunca se opone a un argumento."

ÍNDICE



Introducción

PAGINACIÓN

DISCONTINUA

INTRODUCCIÓN

PRESENTACIÓN DEL PROBLEMA

La Dirección General de Televisión Universitaria (TVUNAM) esta dedicada a la elaboración de programas de carácter cultural y científico, que constituyen un vínculo entre la comunidad universitaria y la comunidad nacional, y, además, ofrece servicios de producción, postproducción, venta de stock y distribución de programas. Esas actividades, aunadas a la tendencia actual hacia el aprovechamiento de la tecnología, derivan en necesidades de desarrollo de sistemas a la medida de la dependencia.

La Subdirección de Enlace Tecnológico y Sistemas, en su búsqueda de la optimización de los procesos de TVUNAM, como producto de una exhaustiva evaluación de las opciones para desarrollo, ha decidido adoptar la tecnología J2EE como herramienta para la creación de soluciones automatizadas en diversas áreas administrativas. Para su aprovechamiento, y tomando en cuenta la rotación de personal a que conlleva el manejo de Programas de becas para reclutamiento de personal en las áreas de desarrollo, se hace necesario contar con documentos y aplicaciones que faciliten la transmisión de la experiencia, dotando a los nuevos elementos, de la información necesaria para su rápida integración a las funciones asignadas, a la vez que se ahorra tiempo y recursos por conceptos de instrucción y adiestramiento por parte de personal con mayor experiencia.

Todo lo anterior, aunado a la escasez de literatura accesible en español sobre las tecnologías empleadas y adecuada a las necesidades de desarrollo en la organización, nos lleva a la evaluación de estrategias que ataquen la problemática, y que deriven en el aprovechamiento eficaz de los recursos destinados a capacitación y asesoría hacia las tecnologías y métodos adoptados por la dependencia.

INTRODUCCIÓN AL PROYECTO.

Se presenta la literatura, documentación y herramientas generadas a partir del estudio del proceso de construcción de software empresarial, incluyendo una aplicación desarrollada con base en la tecnología J2EE (principalmente EJB), siguiendo la metodología de desarrollo utilizada en TVUNAM.

Los productos de este trabajo abarcan las características identificadas como principales dentro del modelo J2EE y las aplicaciones empresariales y las desarrollan con carácter de explicación a través de los temas distinguidos como importantes.

La utilidad de los documentos que integran el presente trabajo, junto con el sistema desarrollado como referencia a la aplicación de las tecnologías, radica en servir como herramientas en la introducción del lector tanto a la tecnología estudiada, como al caso práctico, videojuego en línea, seleccionado.

JUSTIFICACIÓN DEL PROYECTO.

La motivación principal de este proyecto, tratada en el apartado de "Presentación de la Problemática", responde a la necesidad de contar con una herramienta de referencia (tanto aplicación tangible como documentación explicativa) que facilite la transmisión de la experiencia de los elementos humanos con más tiempo dentro de la organización hacia aquellos que recién se incorporan al área de desarrollo. Lo anterior se hace tomando en cuenta que es deseable que la aplicación guía reduzca al mínimo el esfuerzo relacionado con la asimilación de los conceptos particulares al ejemplo, para seguir el desarrollo del caso práctico planteado, centrándose en la aplicación de la tecnología. Para esto se ha evaluado el perfil promedio de los nuevos desarrolladores y se ha seleccionado un tema con el cual sea altamente probable que ya estén familiarizados.

El programa de becas, fuente principal de nuevos elementos para el área de desarrollo, recluta alumnos de las carreras afines al Cómputo para integrarlos a la actividad productiva del desarrollo de sistemas.

La selección de un videojuego como caso práctico de este Proyecto en particular, se basa en el hecho de que nuestro lector prospecto, cuenta en su mayoría con conocimiento y preferencia por esta forma de entretenimiento, permitiendo enfocar el esfuerzo de explicación a los tópicos nuevos, es decir, la tecnología (J2EE) como tal, dejando aparte una introducción exhaustiva a los videojuegos que, dicho sea de paso, representa, a nuestro parecer, una excelente herramienta de apoyo a la educación.

Aunado a lo anterior, un videojuego en línea presenta características particulares que apoyan el tratamiento de la tecnología EJB, tales como:

- *Cero tolerancia a fallas. Es decir, que se trata de un producto que requiere de altos niveles de confiabilidad, evitando errores, ya que todos los usuarios comparten información que debe ser completa y correcta en todo momento, esto a pesar de la susceptibilidad de los sistemas a errores y casos excepcionales.*
- *Componentes distribuidos en un servidor. Un videojuego en línea puede ser organizado en una estructura de componentes distribuidos, donde las partes funcionales para el sistema, como un todo integrado, residen en sistemas dedicados a funciones particulares, haciendo accesible a todos los usuarios las herramientas que requieren para el desempeño de sus funciones, manteniendo los niveles de calidad y seguridad de una aplicación empresarial.*
- *Manejo de transacciones. Dado que existen varios clientes consultando y modificando una base de información común, se requiere el manejo del concepto de transacción, en donde cada acción se da en un ambiente aislado, a la vez que dependiente de otras acciones generadas; este esquema permite que todas las acciones convivan manteniendo la integridad tanto de las acciones como del estado del sistema.*

En resumen, el actual proyecto, independientemente del caso práctico seleccionado, es una iniciativa enfocada a favorecer la adquisición de conocimiento necesario para impulsar el desarrollo de software adecuado a las necesidades de empresas y organizaciones mexicanas.

RESUMEN DE CAPÍTULOS.

Capítulo 1: *Marco de Referencia.*
Se da una breve introducción a los temas que debe conocer el lector de este proyecto para aprovechar mejor el contenido central del mismo. Incluye aspectos básicos de Orientación a Objetos , Java y UML. Así mismo se explican brevemente los conceptos de aplicaciones distribuidas y de la tecnología Entreprise Java Beans junto con XML. Por último se da una breve introducción a los videojuegos, su desarrollo, historia y aplicación.

Capítulo 2. *Objetivos.*
Se presenta claramente el objetivo principal del presente proyecto así como los objetivos específicos.

Capítulo 3. *Alcance.*
Se explica el grado de desarrollo del presente proyecto, se definen los aspectos que se incluyen y aquellos que no se abarcan dentro del trabajo.

Capítulo 4. *Metodología .*
Se detalla la metodología elegida para el desarrollo del proyecto así como su utilidad y estructura, en términos de las etapas que la definen.

Capítulo 5. *Desarrollo.*
Se describen las etapas de análisis y diseño del caso practico junto con los artefactos y productos que ayuden a la mejor comprensión del proyecto.
En la etapa de análisis, se describe la detección de necesidades, la definición del problema, su factibilidad, la especificación de requerimientos, mientras que la etapa de diseño se describe la arquitectura del proyecto dentro de un modelo de capas, desarrollando la descripción de cada nivel de aplicación (presentación, negocio, datos).

Capítulo 6. *Marco de Aplicación.*
Se explica la utilidad del proyecto dentro de TVUNAM en el área de Desarrollo de Sistemas, así como la aportación que representa cada una de sus secciones. Se plantean futuros trabajos relativos a el enriquecimiento de este proyecto.

Capítulo 7. *Conclusiones*
Se resumen los aspectos derivados tanto del desarrollo del proyecto como de la aplicación de sus productos en el área de Sistemas de la dependencia. Se analiza la forma en la que los objetivos planteados, encontraron solución con el trabajo presentado, dentro del marco de alcance planteado inicialmente.

Capítulo 1.

Marco de Referencia

CAPÍTULO 1. MARCO DE REFERENCIA

CONCEPTOS BÁSICOS SOBRE ORIENTACIÓN A OBJETOS

Orientación a Objetos

Para el inicio de los años setentas, los problemas más recurrentes en cuanto a desarrollo de sistemas eran que muchos de los proyectos no lograban terminarse, pocos se terminaban cumpliendo con los requisitos iniciales y no todos los que se terminaban, eran usados según lo previsto. Todo esto residía en la dificultad de adaptar el software a nuevos requerimientos no planteados inicialmente.

El alto grado de planificación y previsión requerida por los sistemas es contrario a la propia realidad. "El hombre aprende y crea a través de la experimentación, no de la planeación"[1].

Aprovechando las capacidades inherentes al ser humano para clasificar, generalizar y abstraer objetos, de tal forma que puede tratar con el mundo complejo que le rodea, surge la Orientación de Objetos como un paradigma donde un sistema se descompone en objetos, siendo un objeto una representación abstracta de elementos que cooperan entre si. La Orientación a Objetos brinda, en cierta forma, los métodos de experimentación que permiten la generación de sistemas más adaptables a cambios, ya que no exige la planificación completa y rigurosa de todos los requerimientos de un proyecto antes de escribir la primer línea de código.

Por lo tanto podemos definir a la Orientación a Objetos como un conjunto de disciplinas de ingeniería de software que facilitan el desarrollo de sistemas complejos a partir de componentes individuales. Las características de este paradigma de desarrollo son:

- *Flexibilidad.* Modela el mundo real de manera muy cercana a la perspectiva del usuario.
- *Compatibilidad.* Interactúa de manera aceptable con un ambiente de computación.
- *Reusabilidad.* Construye componentes reutilizables y librerías fácilmente extensibles a otras aplicaciones, ya que modifica y extiende fácilmente implementaciones de componentes sin necesidad de recodificar desde el inicio.

El paradigma de Orientación a Objetos esta definido por conceptos como clase, objeto, encapsulamiento, herencia y polimorfismo, mismos que trataremos a continuación.

Objetos y Clases

Un *objeto* es el modelo de alguna cosa, real o abstracta, que conforma un sistema, definido por sus características particulares que lo diferencian de otros que comparten la misma estructura o funcionalidad. Cada objeto posee capacidades y funciones que le permiten realizar tareas que, en conjunto con otros objetos, conforman los diferentes procesos del sistema. Un objeto es la instanciación de una

[1] García Zavala Angel, et al, Orientación a Objetos

clase y posee un estado, definido por sus atributos, un comportamiento, determinado por sus métodos y una identidad, es decir, un objeto es un elemento particular dentro de una clasificación determinada.

- Estado. Define la situación del objeto en un momento determinado.
- Comportamiento. Describe la manera en que reacciona el objeto ante mensajes recibidos.
- Identidad. Lo identifica de otros objetos de la misma clase.

Una *clase* es una agrupación de objetos que comparten estructura y comportamiento. En la orientación a objetos, todo forma parte de una clase, describe a una clase o es una clase. Responde a la idea de conceptualizar y abstraer, ya que es como un molde o plantilla desde el cual se crean los objetos, la declaración de una clase siempre tiene la descripción de la estructura interna (variables o atributos) y del comportamiento (métodos) comunes a todos los objetos a que define.

Variables.		
Tipo	***Descripción.***	***Prefijo en Java.***
De instancia	Cada objeto (instancia de la clase) posee su propia copia de la variable. El valor de cada variable es independiente a las demás	
De clase (Estáticas)	Todos los objetos comparten una sola variable. La variable tiene el mismo valor en todas las instancias.	static
Constantes	Una vez declarada e inicializada, la variable no permite modificaciones.	final

Métodos.		
Tipo	***Descripción.***	***Prefijo en Java.***
De instancia	Es un método particular al objeto (instancia de la clase), como si existiera una copia del mismo por cada objeto creado.	
De clase (Estáticos)	Pueden ser invocados sin necesidad de instanciar un objeto de la clase.	static
Existen otras clasificaciones de métodos que incluyen métodos modificadores, analizadores, constructores, etc. pero su descripción sale de la finalidad de referencia básica de este apartado.		

La relación que existe entre Clase y Objeto es la instancia. Una instancia es un objeto particular de la clase, que asigna valor especifico a todos los atributos.

Para poder manipular los objetos de un sistema, los métodos de un objeto pueden ser llamados por el mismo objeto o por otros objetos. Al utilizar los métodos surge una comunicación, la cual permite crear operaciones más complicadas. Esa comunicación implica mandar mensajes a los objetos, los cuales serán los parámetros con los que trabajara el método.

Encapsulamiento

Se entiende como encapsulamiento cuando toda la información de un objeto se encuentra empaquetada y puede reutilizarse como una especificación o componente de un programa. Significa que el objeto es auto-contenido, es decir, que la definición del objeto incluye tanto los datos que usa (atributos) como los procedimientos (métodos) que actúan sobre los mismos.

El objeto esconde sus datos de los demás objetos y permite el acceso a los datos mediante sus propios métodos, lo que se conoce como ocultamiento de información. Este mecanismo evita la corrupción de los datos de un objeto. Si todos los programas pudieran tener acceso a los datos de cualquier forma que quisieran los usuarios, los datos se podrían corromper o utilizar de mala manera. El encapsulamiento protege los datos del uso arbitrario y no pretendido.

Herencia

Permite definir a una clase puede ser definida a partir de otra clase anteriormente creada. La clase definida de esta manera tendrá la estructura y funcionalidad de la clase existente, con la posibilidad de adicionar funcionalidad extra. La clase de la que se hereda la funcionalidad se llama comúnmente clase *padre*(superclase) y la que hereda es la clase *hija*(subclase).

Al crear clases que hereden a otras, se debe tomar en cuenta las siguientes condiciones:

1. Al heredar métodos y atributos existe una reutilización, ya que adecuamos lo que se definió anteriormente a nuevas necesidades.
2. Otras forma de reutilización es por composición donde una clase contiene un atributo que es un objeto de otra clase.

En cuando a la herencia múltiple, esta ocurre cuando una clase hija hereda de dos o más clases padre a la vez. Esto puede generar ambigüedades cuando los atributos o los métodos de dos o más superclases de una subclase tienen el mismo nombre o generar confusión sobre hasta que punto se debe heredar métodos o atributos.

Cabe señalar que en el caso del lenguaje de programación Java todas las clases son heredadas. Aún cuando no se indique explícitamente, existe una jerarquía de objetos única, lo que significa que existe una clase de la cual son hijas todas las demás: la clase *Object*, esto hace posible que todas las clases tengan algunas cosas en común dentro del lenguaje.

Polimorfismo

El polimorfismo es la propiedad que tienen los métodos para mantener una respuesta de manera unificada, con la misma semántica pero con diferente implementación, es decir, un mismo método se comporta de forma diferente en dos clases. El polimorfismo permite dar el mismo nombre a servicios implementados en diferentes objetos, estos servicios pueden implementarse de forma diferente pero producirán el mismo tipo de resultados. Se utiliza el polimorfismo cuando se quiere enviar el mismo mensaje a diferentes objetos sin saber el objeto específico al que se esta enviando en el mensaje. Hay varias formas de implementarla.

- *SobreEscritura:* se define un método cuya firma sea igual a la del método de la clase que hereda, dándole un comportamiento en particular.
- *SobreCarga:* se define un método cuyo nombre sea igual al del método de la clase que hereda, alterando la firma, cambiando sus parámetros.
- *Polimorfismo por Reemplazo*: se define un método cuyo parámetro sea un objeto de una clase padre, del cual sólo se conocen atributos y métodos, existiendo una reutilización con un solo método para diferentes clases.
- *Métodos Virtuales*: implica a un método sobrescrito hasta el momento de ejecución.

Historia del lenguaje Java

Para apreciar el significado e importancia de Java, es muy importante conocer su lugar de origen y cuales fueron sus propósitos.

La compañía Sun Microsystems decidió intentar introducirse en el mercado de los electrónico de consumo y desarrollar programas para pequeños dispositivos electrónicos, creando la filial FistPerson Inc., su mercado original eran los equipos domésticos. Dado que este mercado requería interfaces más cómodas e intuitivas, la fiabilidad del código y la facilidad de desarrollo deberían ser características importantes del lenguaje de programación elegido.

En Diciembre de 1990, Patrick Naughton, un empleado de la empresa Sun, reclutó a sus colegas James Gosling y Mike Sheridan para trabajar sobre un nuevo tema conocido como "Green". Este a su vez estaba auspiciado por la compañía "Sun founder Bill Joy" y tenía como objetivo principal crear un lenguaje de programación accesible, fácil de aprender y de usar, que fuera universal, y que estuviera basado en un ambiente C++ ya que había mucha frustración por la complejidad y las limitaciones de los lenguajes de programación existentes.

En abril de 1991, el equipo, principalmente James Gosling creo un compilador original y lo denominó "Oak", y con la ayuda de los otros miembros del equipo desarrollaron un decodificador que mas tarde se convertiría en el lenguaje Java

El proyecto "Green" coincidió con el nacimiento del fenómeno mundial Web. Al examinar las dinámicas de Internet, lo realizado por el ex equipo verde se adecuaba a este nuevo ambiente ya que cumplía con los mismos requerimientos de las set-top box OS que estaban diseñadas con un código de plataforma independiente pero sin dejar de ser pequeñas y confiables.

Patrick Naugthon procedió a la construcción del lenguaje de programación Java que se accionaba con un browser[2] prototipo, más tarde se le fueron incorporando algunas mejoras y el browser Hot Java fue dado a conocer al mundo en 1995.

Con el paso del tiempo el Hot Java se convirtió en un concepto práctico dentro del leguaje Java y demostró que podría proporcionar una forma segura multiplataforma para que el código pueda ser bajado y corrido del Host del World Wide Web y que de otra forma no son seguros.

Una de las características más atractivas del Hot Java fue su soporte para los "applets", que son las partes del código Java que pueden ser cargadas mediante una red de trabajo para después ejecutarlo localmente y así lograr o alcanzar soluciones dinámicas en computación acordes al rápido crecimiento del ambiente Web.

Para dedicarse al desarrollo de productos basados en la tecnología Java, Sun formó la empresa Java Soft en enero de 1996, de esta forma de se dio continuidad al fortalecimiento del programa del lenguaje Java y así trabajar con terceras partes para

[2] Navegador

crear aplicaciones, herramientas, sistemas de plataforma y servicios para aumentar las capacidades del lenguaje.

Durante ese mismo mes, Java Soft dio a conocer el Java Developmet Kit (JDK) 1.0, una rudimentaria colección de componentes básicos para ayudar a los usuarios de software a construir aplicaciones de Java. Dicha colección incluía el compilador Java, un visualizador de applets, un debugger[3] prototipo y una máquina virtual Java(JVM), necesaria para correr programas basados en Java, también incluía paquetería básica de gráficos, sonido, animación y trabajo en red.

Asimismo el Netscape Comunications Inc, mostró las ventajas de Java y rápidamente se asoció con Java Soft para explotar su nueva tecnología. No pasó mucho tiempo antes de que Netscape Communications decidiera apoyar a los Java applets en Netscape Navigator 2.0. Este fue el factor clave que lanzó a Java a ser reconocido y famoso, y que a su vez forzó a otros vendedores para apoyar el soporte de applets en Java.

Como parte de su estrategia de crecimiento mundial y para favorecer la promoción de su nueva tecnología, Java Soft otorgó permisos a otras compañías para que pudieran tener acceso al código fuente de Java y al mismo tiempo mejorar sus navegadores , dicha licencia también les permitía crear herramientas de desarrollo para programación Java y los facultaba para acondicionar Máquinas Virtuales Java (JVM), a varios sistemas operativos.

Muy pronto las licencias o permisos contemplaban a prestigiadas firmas como IBM, Microsoft, Symantec, Silicon Graphics, Oracle, Toshiba y Novell.

Desde su aparición, Java se ha ganado una impresionante cantidad de apoyo. Virtualmente cada vendedor importante de software ha obtenido autorización de Java y ahora ha sido incorporado en los principales sistemas operativos base de PC's de escritorio hasta estaciones de trabajo UNIX.

Los applets Java (basados en JDK 1.02) son apoyados por los dos más populares navegadores web (Netscape Navigator 3.0 y Microsoft Internet Explorer 3.0). I.B.M./Lotus, Computer Asociates, Symantec, Informix, Oracle, Sybase y otras poderosas empresas de software están construyendo Software 100% puro JAVA.

[3] Depurador de código.

Características

1. *Simplicidad*. Reduce en un 50% los errores mas comunes de programación como:
   - Aritmética de punteros.
   - No existen referencias.
   - Registro.
   - Definición de tipos.
   - Macros.
   - Necesidad de manejar dinámicamente la memoria porque cuenta con un Colector de Basura.
2. *Orientado a Objetos*. Implementa la tecnología básica de C++. Soporta la encapsulamiento, herencia y polimorfismo.
3. *Distribuido*. Aún cuando Java en sí no es distribuido, existen librerías de rutinas para acceder e interactuar con protocolos http y ftp, facilitando el trabajo con archivos, permitiendo así el que corran procesos en varias máquinas.
4. *Robusto*. Realiza verificaciones en busca de problemas tanto en tiempo de compilación como en tiempo de ejecución. La comprobación de tipos en Java ayuda a detectar errores en el ciclo de desarrollo. Una de sus ventajas es que obliga a la declaración explícita de métodos.
5. *Portabilidad*. El compilador de Java compila el código a un archivo objeto de formato independiente de la arquitectura de la máquina . Al compilar genera un código intermedio entre el lenguaje máquina del procesador y Java, conocido como ByteCode. Por medio de la Máquina Virtual de Java(JVM), se ejecuta el ByteCode, por lo que esta aplicación esta en diferentes plataformas(Solaris, Linux ,Windows, HP-UX,MacOs,etc.).Así mismo la Máquina Virtual Java desempeña otra funciones como la de aislar los programas Java al entorno de la JVM. La principal desventaja de esto, es que la velocidad de ejecución se ve afectada, por lo que se ha recurrido a los compiladores JIT(Just In Time) los cuales están situados a la entrada de la JVM, traduciendo el ByteCode al lenguaje máquina del procesador sin ejecutarlo.
6. *Seguro*. Java es seguro porque previene el acceso ilegal a la memoria. El código Java pasa muchas comprobaciones antes de ejecutarse en máquina. El código se pasa a través de un verificador de ByteCode que comprueba el formato de los fragmentos de código aplicando un probador de teoremas para detectar fragmentos de código ilegal.
7. *Manejo de Archivos*. El cargador de clases separa el espacio de nombres del sistema, de los procedentes de la red, previniendo aplicaciones del tipo Caballo de Troya. Cuando una clase importada de la red accede a otra clase, primero se busca en las clases predefinidas y luego en el espacio de nombres de la clase que hace la referencia.
8. *Interpretado*. El intérprete Java puede ejecutar directamente el código objeto.
9. *Multitarea*. Java permite realizar muchas actividades simultáneas en un programa. Las tareas son básicamente pequeños procesos o piezas independientes de un gran proceso.
10. Dinámico. No intenta conectar todos los módulos que comprenden una aplicación hasta el mismo tiempo de la ejecución. Las librerías nuevas o actualizadas no paralizarán la ejecución de las aplicaciones actuales siempre que mantengan las librerías anteriores conocidos como Application Programming Interface (API). La **Java API** se divide en dos grupos: La **API** básica y la **API** extendida.

Sintaxis

Comentarios

En Java hay tres tipos de comentarios:

Sintaxis	Uso
`// ...comentario...`	Comentarios para una sola línea
`/* ...comentario... */`	Comentarios para mas de una línea
`/** ...comentario... */`	Comentarios de documentación (java doc)

Identificadores

Los identificadores nombran variables, funciones clases y objetos. Los nombres validos para un identificador deben seguir las siguientes reglas:

- Debe comenzar con una letra, guión bajo(_) o un símbolo de dólar($).
- Los siguientes caracteres pueden ser letras o dígitos.
- Se distinguen las mayúsculas y de las minúsculas
- No hay longitud máxima establecida.
- Existen ciertas palabras reservadas por el lenguaje que no pueden ser utilizadas como identificadores.

Tipos de Valores literales.

Un valor en Java se crea utilizando una representación literal de él. Los tipos de datos primitivos son tipos de datos abstractos no definidos en función de otros tipos, Java reconoce los siguientes:

Enteros	Reales	Booleanos
byte	*float*	*char*
short	*double*	
int		
long		

Separadores

Los separadores son los caracteres que van a definir la forma y función del código.

Separador	Uso
( ) paréntesis	Para contener lista de parámetros en la definición y llamada a métodos. También se utiliza para definir precedencia en expresiones, contener expresiones para control de flujo y rodear las conversiones de tipo (casting).
{} llaves	Para contener los valores de matrices inicializadas automáticamente. También se utiliza para definir un bloque de código, para clases, métodos.
[ ] corchetes	Para declarar tipos matriz. También se utiliza cuando se referencian valores de matriz.
; punto y coma	Para separar sentencias.
, coma	Para separar identificadores consecutivos en una declaración de variables.
. punto	Para separar nombres de paquetes de paquetes y clases.

Operadores

Los operadores realizan algunas funciones en uno o dos operandos. Los operadores que requieren un operador se llaman operadores unarios. Por ejemplo, "++" es un operador unario que incrementa el valor su operando en uno. Los operadores que requieren dos operandos se llaman operadores binarios. Por ejemplo, el operador "=" es un operador binario que asigna un valor del operando derecho al operando izquierdo.

Los operadores unarios en Java pueden utilizar la notación de prefijo o de sufijo. La notación de prefijo significa que el operador aparece antes de su operando. mientras la notación de sufijo significa que el operador aparece después de su operando.

Operador Unario	
Notación prefijo	*Notación sufijo*
`operador operando` `++ i`	`operando operador` `i ++`

Todos los operadores binarios de Java tienen la misma notación, es decir aparecen entre los dos operandos:

Operador Binario
`op1 operator op2` `x = 1`

Además de realizar una operación también devuelven un valor. El valor y su tipo dependen del tipo del operador y del tipo de sus operandos. Por ejemplo, los operadores aritméticos (realizan las operaciones de aritmética básica como la suma o la resta) devuelven números, el resultado típico de las operaciones aritméticas. El tipo de datos devuelto por los operadores aritméticos depende del tipo de sus operandos: si sumas dos enteros, obtendrás un entero. Se dice que una operación evalúa su resultado.

A continuación se muestran los diferentes operadores válidos en Java.

Operador	Tipo	Uso	Descripción
+	*Aritmético*	`op1 + op2`	Suma op1 y op2. Java extiende la definición del operador + para incluir la concatenación de cadenas.
-		`op1 - op2`	Resta op2 de op1
*		`op1 * op2`	Multiplica op1 y op2
/		`op1 / op2`	Divide op1 por op2
%		`op1 % op2`	Obtiene el resto de dividir op1 por op2
+		`+ op`	Indica un valor positivo
-		`- op`	Niega el operando
++		`op ++`	Incrementa op en 1; evalúa el valor antes de incrementar
++		`++ op`	Incrementa op en 1; evalúa el valor después de incrementar
--		`op --`	Decrementa op en 1; evalúa el valor antes de decrementar
--		`-- op`	Decrementa op en 1; evalúa el valor después de decrementar

Operador	Tipo	Uso	Descripción
>	*Relacional*	op1 > op2	Devuelve true si op1 es mayor que op2
>=		op1>= op2	Devuelve true si op1 es mayor o igual que op2
<		op1 < op2	Devuelve true si op1 es menor que op2
<=		op1 <= op2	Devuelve true si op1 es menor o igual que op2
==		op1 == op2	Devuelve true si op1 y op2 son iguales
!=		op1 != op2	Devuelve true si op1 y op2 son distintos
&&	*Condicional*	op1 && op2	Devuelve true si op1 y op2 son verdaderos
||		op1 || op2	Devuelve true si uno de los dos es verdadero
!		! op	Devuelve true si op es falso
=	*Asignación*	op1 = op2	Asigna el valor de op2 a op1
+=		op1 += op2	Equivale a op1 = op1 + op2
-=		op1 -= op2	Equivale a op1 = op1 - op2
*=		op1 *= op2	Equivale a op1 = op1 * op2
/=		op1 /= op2	Equivale a op1 = op1 / op2
%=		op1 %= op2	Equivale a op1 = op1 % op2
&=		op1 &= op2	Equivale a op1 = op1 & op2

Expresiones

Las expresiones se utilizan para calcular y asignar valores a las variables y para controlar el flujo de un programa Java. El trabajo de una expresión se divide en dos partes: realizar los cálculos indicados por los elementos de la expresión y devolver algún valor. La tabla siguiente muestra la precedencia asignada a los operadores de Java. Los operadores se han listado por orden de precedencia de mayor a menor. Los operadores con mayor precedencia se evalúan antes que los operadores con un precedencia relativamente menor. Los operadores con la misma precedencia se evalúan de izquierda a derecha.

Operadores (En orden de prededencia)	
operadores sufijo	[ ] . (params) expr++ expr--
operadores unarios	++expr --expr +expr -expr ~ !
creación o tipo	new (type)expr
multiplicadores	* / %
suma/resta	+ -
desplazamiento	<< >> >>>
relacionales	< > <= >= instanceof
igualdad	== !=
bitwise AND	&
bitwise exclusive OR	^
bitwise inclusive OR	|
AND lógico	&&
OR lógico	||
condicional	? :
asignación	= += -= *= /= %= ^= &= |= <<= >>= >>>=

Una sentencia de control de flujo sirve para saber el orden de ejecución de los comandos, expresiones dentro de un programa. La siguiente tabla resume los diferentes tipos de sentencias de control de flujo.

Sentencias	Palabras clave
toma de decisiones	`if-else, switch-case`
bucles	`while, do-while, for`
excepciones	`try-catch-finally, throw`
miscelaneas	`break, continue, return`

Sentencia if / else

```
if(expression-booleana){
   sentencias;
}else{
   sentencias;
}
```

Esta construcción hace que el programa atraviese un conjunto de decisiones booleanas que determinan la ejecución de distintos fragmentos de código.

La cláusula else es opcional. Cada una de las sentencias puede ser compuesta y la expresión-booleana puede ser:

- Una variable declarada de tipo booleano
- Una expresión que utilice operadores relacionales para generar el resultado de una comparación.

Sentencia switch-case

```
switch(expresión){
      case valor1: sentencias;break;
      case valor2: sentencias;break;
      default: sentencias;
}
```

Proporciona una forma limpia de enviar la ejecución a partes diferentes del código en base a un valor o expresión que regrese un entero.

El valor de la *expresión* se compara con cada uno de los valores literales de las sentencias *case*. Si coincide con alguno (*expresión* == *valor*X), se ejecuta el código que sigue a la sentencia case (*sentencias*). Si no coincide ninguno se va a las sentencias enmarcadas en default. En caso de que no se ponga la sentencia "*break*", se seguirá evaluando la expresión, si coincide con alguna condición posterior y de existir el default lo ejecutará.

Sentencia while

```
[inicialización]
while(terminación-expresion-booleana){
      sentencias;
      [iteracion]
}
```

Ejecuta repetidamente una vez tras otra una sentencia mientras una expresión booleana (*terminación-expresion-booleana*) sea verdadera.

Esta sentencia se utiliza para crear una condición de entrada, siendo el control de entrada la expresión condicional que controla el bucle, de tal modo que si esta comprobación es false la primera vez, las sentencias no se ejecutarán, por lo que puede existir una condición inicial fuera del ciclo, la cual irá cambiando dentro de ella (*iteración*).

Sentencia do/while

```
[inicializacion]
do{
   sentencias;
}while(terminación-expresion-booleana).
```

Cuando se desea ejecutar el cuerpo de un ciclo while, al menos una vez, se utilizará esta sentencia. *Ver Sentencia while.*

Sentencia for

```
for(inicialización;terminación;incremento){
   sentencias;
}
```

Esta sentencia involucra a tres acciones en su ejecución:

- Inicialización de la variable de control (*inicializacion*)
- Comprobación del valor de la variable de control en una expresión condicional (*terminación*)
- Actualización de la variable de control (*incremento*)

La cláusula de inicialización y la cláusula de incremento pueden estar compuestas por varias expresiones separadas mediante el operador coma, garantizando que se ejecute de izquierda a derecha. La primera cláusula (de inicialización) solo se ejecuta una vez, al momento de iniciar el ciclo. La segunda cláusula (terminación) consiste en una única expresión que regrese un booleano, si es falso se finaliza el ciclo. El valor de la segunda cláusula se comprueba cuando la sentencia comienza la ejecución y en cada una de las iteraciones posteriores.

La tercera cláusula, de incremento, no se ejecuta hasta que se han ejecutado todas las sentencias que componen el ciclo.

Sentencia return

Se utiliza para terminar un método o función y opcionalmente devolver un valor al método de llamada. El valor de retorno debe ser congruente con la firma de la clase, si indica que regresara un entero, la variable o el resultado de la expresión debe ser entera. En caso de que la firma indique que no se regresará nada(void) no debe incluirse esta sentencia.

Sentencias break

Setencia de ruputura que salta a la siguiente sentencia, si se esta dentro de un ciclo, se finaliza el ciclo de control (lo "rompe").

Sentencia continue

Se utiliza para saltar de la sentencia actual hacia el principio del ciclo, a diferencia del anterior, al encontrarse dentro de un ciclo, salta hacia el inicio de la siguiente iteración sin finalizar el ciclo.

Definición de clases

Recordando lo escrito acerca de las Clases en el apartado de Orientación a Objetos, en Java, la definición de una clase consta de dos partes, la declaración y el cuerpo.

```
DeclaraciónClase{
    CuerpoClase
}
```

La declaración de la clase indica al compilador el nombre de la clase, la clase de la que deriva, los privilegios de acceso y si implementa interfaces. El nombre de la clase debe ser un identificador válido en Java. Por convención el nombre de una clase empieza con mayúscula. El cuerpo de la clase contiene las declaraciones de variables y los métodos. La declaración de variables puede estar dentro del cuerpo de la clase o dentro del cuerpo de un método. Sin embargo, estas últimas no son variables miembro de la clase, sino variable locales del método.

La sintaxis general de la definición de una clase es la siguiente:

```
NombreDeLaClase extends ClasePadre implements ClaseInterfaz{

  //Declaración de las variables de instancia
  Tipovariable nombreVariable = valorInicializacion;
  final TipoConstante nombreConstante = valorInicialización;

  //Declaración de las variable de la clase
  static TipoVariable nombreVariable = valorInicializacion;

   metodoConstructor(Clase argumento1){
         Variable_Intancia = objeto;
   }

   void metodoInstancia(){
         variables locales;
         código
   }
   static int metodoDeLaClase(){
         int i =8;
         return i+5
   }
}
```

Variables y constantes (Atributos).

Las variables pueden ser de tipo primitivo u objetos (instancias) de una clase.

El tipo de una variable determina los valores que se le pueden asignar y las operaciones que se pueden realizar con ella. El nombre de una variable debe ser un identificador válido en Java y han de ser únicos dentro de la clase.

Las variables en Java incluyen variables de instancia, estáticas y de valor constante (Constantes).

Variables (atributos).		
Tipo	***Descripción.***	***Prefijo en Java.***
De instancia	Cada objeto (instancia de la clase) posee su propia copia de la variable. El valor de cada variable es independiente a las demás	
De clase (Estaticas)	Todos los objetos comparten una sola variable. La variable tiene el mismo valor en todas las instancias.	`static`
Constantes	Una vez declarada e inicializada, la variable no permite modificaciones.	`final`

Métodos

Los métodos son funciones que pueden ser llamadas dentro de la clase o por otras clases. Como en la declaración de clases, la implementación de un método consta de dos partes, una declaración y un cuerpo, de acuerdo a la siguiente estructura básica:

```
tipoRetorno nombreMetodo ([lista_de_argumentos]){
    cuerpoMétodo
}
```

Cada vez que se especifica en la firma de un método un tipo de valor de retorno (*tipoRetorno*) el método debe regresar un valor. Esto se hace por medio de la cláusula return; *nombreMetodo* debe ser un identificador válido en Java. *La lista_de_argumentos* es opcional, y en caso de incluirse requiere especificar la clase del argumento y el nombre con el que se le conocerá dentro del método.

La sintaxis completa de un método puede incluir numerosos atributos y modificadores a la hora de declararlos, para determinar el control de acceso, si es un método estático o no, si es abstracto, etc.

```
especificadorAcceso [static,abstract,  final, native, synchronized]
tipoRetorno nombreMetodo ([lista_de_argumentos])
                                                    throws listaExcepciones{
    cuerpoMétodo
}
```

especificadorAcceso *determina si otros objetos pueden acceder al método y como pueden hacerlo, puede ser public, package, protected o private. Más adelante se trata sobre cada uno de estos modificadores de control de acceso.*

Modificadores	Descripción
`static`	Indica que los métodos pueden ser accedidos sin la necesidad de instanciar un objeto del tipo que determina esta clase
`Abstract`	Indica que el método no está definido en la clase, sino que se encuentra sobrescrito en una subclase.
`Final`	Evita que un método pueda ser sobrescrito (como en el caso de los atributos constantes).
`Native`	Métodos escritos en C o C++ o en otros lenguajes ajenos a Java.
`Synchronized`	Usado en el soporte de multitarea.

throws ListaExcepciones define las excepciones que el método es capaz de generar y manipular.

Método Constructor

El constructor es un tipo específico de método que siempre tiene el mismo nombre que la clase y se utiliza para construir objetos de esa clase. No tiene valor de retorno explícito ya que por omisión devuelve un objeto de la misma clase.

```
nombreDeLaClaseQueConstruye ([lista_de_argumentos]){
    cuerpoMétodo
}
```

Los constructores pueden sobrecargarse, definiendo varios dentro de la misma clase, siempre que reciban parámetros distintos y, aunque pueden contener código, su función principal es inicializar el nuevo objeto. Cuando se instancia una clase, es necesario hacer una llamada explícita a uno de sus métodos constructores. Las clases Java cuentan con un método constructor por omisión que no recibe argumentos, aun cuando no aparece en el código. Si se especifica un constructor para la clase, ya no se cuenta con el método constructor por omisión.

Paso de Información a un Método

Los parámetros (o argumentos) son variables locales declaradas en el cuerpo del método que están inicializadas al valor que se pasa como parámetro en la invocación del método.

Modalidad	*Descripción.*
Paso por Valor	Todos los argumentos de tipos primitivos deben pasarse por valor. Cuando se le llama, el método recibe el valor de la variable pasada.
Paso por Referencia	Cuando se pasa un objeto por referencia, se está pasando la dirección de memoria en la que se encuentra almacenado el objeto. Significa que el método no puede cambiar el objeto referenciado, pero si puede invocar a los métodos del objeto y puede modificar las variables accesibles dentro del objeto.

Control de Acceso

En Java, las clases y cualquiera de los miembros que la integran, pueden ser designados public, protected, package o private en términos de quién puede acceder a ellos, haciendo uso del encapsulamiento, logrando proteger sus variables y métodos miembros frente al acceso de otros objetos.

El control de acceso aplica a nivel de clase, no de objeto, ya que es parte de la definición de la estructura de los objetos que instancian a la clase.

Nivel de acceso	*Misma clase*	*Subclases de la Clase*	*Clases dentro del Paquete de la Clase*	*Todos*
`private`	X			
`protected`	X	X	X	
`package`	X		X	
`public`	X	X	X	X

La descripción detallada de cada nivel de acceso es la siguiente:

Especificador	Descripción
`private`	Un miembro privado es accesible sólo para la clase en la que está definido. Se utiliza este acceso para declarar miembros que sólo deben ser utilizados por la clase. Esto incluye las variables que contienen información que si se accede a ella desde el exterior podría colocar al objeto en un estado de inconsistencia, o los métodos que llamados desde el exterior pueden poner en peligro el estado del objeto o del programa donde se está ejecutando.
`protected`	Permite a la propia clase, las subclases y todas las clases dentro del mismo paquete que accedan a los miembros. Este nivel de acceso se utiliza cuando es apropiado para una subclase de la clase tener acceso a los miembros, pero no las clases no relacionadas.
`public`	Todas las clases, en todos los paquetes tienen acceso a los miembros públicos de la clase. Los miembros públicos se declaran sólo si su acceso no produce resultados indeseados si un extraño los utiliza.
`package`	Se obtiene si no se especifica ningún otro nivel de acceso a los miembros. Este nivel de acceso permite que las clases del mismo paquete que la clase tengan acceso a los miembros.

Clases Estáticas

Una clase estática es aquella que no se necesita instanciar un objeto de esa clase para acceder a sus variables y/o métodos. Para hacer estática una clase se necesita poner el modificador static en cada uno de los métodos que se desea hacer accesibles.

Clases Internas

Una clase interna es una clase dentro de otra, cuyo propósito es evitar la definición de clases muy pequeñas que no necesitan conocer los usuarios de un paquete. Estas tienen visibilidad completa de la clase que las contiene, incluyendo métodos privados.

Hay tres tipos de Clases Internas:

- *Clase miembro*: Clase interna no estática. No puede tener atributos o métodos estáticos.
- *Clase local*. Clase interna definida dentro de un bloque de código, siendo sólo visible y utilizable dentro de ese pedazo de código.

Clase anónima. Clase interna local sin nombre que se define como si fuera una sentencia.

Clases Abstractas.

Una clase abstracta es una descripción "incompleta" de algo. Se declara cuando la clase sea una generalización de otras clases y tengamos la seguridad de que no tiene sentido la creación de objetos de esta clase.

Herencia en Java

En Java, como en otros lenguajes de programación orientados a objetos, las clases pueden derivar de otras clases. La herencia en Java es el mecanismo por medio del cual se crean nuevos objetos definidos en términos de objetos ya existentes. La clase derivada (la clase que proviene de otra clase) se denomina subclase, mientras la clase de la que está derivada es llamada superclase.

Para este lenguaje de programación, todas las clases derivan de alguna otra, favoreciendo la estructura del lenguaje al definir características comunes a todas las clases, por lo que la clase más alta, de la que todas las demás heredan, es la clase *Object*, definida en el paquete *java.lang*. Object es la raíz de la herencia de todas las clases en Java.

Para indicar que una clase es subclase de otra se utiliza la palabra reservada "extends" la cual es declarada en el nombre de la clase. Asi, se indica que la nueva clase "extiende" a una clase predefinida.

```
public class NuevaClase exteds ClasePredefinida{
}
```

En relación con los miembros que se heredan, las subclases:as subclases:

- heredan aquellos atributos y métodos declarados como ***public*** o ***protected***
- heredan aquellos atributos y métodos declarados sin especificador de acceso, siempre que la subclase esté en el mismo paquete que la clase, ya que en java, el especificador de acceso por omisión es package.
- no hereda los atributos y métodos de la superclase si la subclase declara alguno que utiliza el mismo nombre o firma. Se dice que el miembro de la subclase sobreescribe al de la superclase (lo oculta).
- no hereda miembros declarados como ***private.***

En casos en que se ve involucrada la herencia, los constructores toman un significado especial, ya que primero se ejecuta el constructor de su superclase para que se inicialicen correctamente aquellas variables que deriven de la súper clase.

El objeto this

Sirve para especificar que las variables o métodos a los que se hace referencia pertenecen al objeto instanciado. Normalmente, dentro del cuerpo de un método de un objeto se puede referir directamente atributos del objeto. Sin embargo, algunas veces no se querrá tener ambigüedad entre el nombre de algun atributo propio del objeto y algun parámetro recibido que tengan el mismo nombre.

```
this.[variable o metodo]
```

El objeto super

Si el método oculta algún miembro de la superclase, aún es posible referirse a la variable oculta utilizando **super**. Esto es, si el método sobreescribe uno de los métodos de la superclase, se puede llamar al método sobreescrito a través de **super**.

```
super.[variable o metodo]
```

Polimorfismo en Java

El polimorfismo es una de las propiedades importantes de la Programación Orientada a Objetos y para comprenderlo cabalmente debemos primero entender la herencia ya que son conceptos muy relacionados.
El polimorfismo significa que objetos similares pueden responder al mismo mensaje de diferentes maneras. Así un objeto muestra "múltiples formas".

El polimorfismo es una de las técnicas más importantes que permiten al programador separar las cosas que cambian de las cosas que deben permanecer igual, así, se puede reutilizar código existente o crear código nuevo de acuerdo a las necesidades del sistema.

Al usar métodos polimórficos sólo hay que preocuparnos de su nombre y no de las variables que son pasadas como parámetros. Java permite la simplificación de código usando 3 formas de polimorfismo, que son herencia, sobrecarga e interfaces.

El tipo de polimorfismo más sencillo se refiere a la ***herencia***. Una subclase puede sobreescribir completamente la implementación de un método heredado o puede mejorarlo añadiéndole funcionalidad. Para reemplazar completamente la implementación de un método de la superclase, simplemente se define a un método con el mismo nombre y firma que el del método de la superclase , sobreescribiendolo. Una subclase no puede sobreescribir métodos que hayan sido declarados como **final** en la superclase (por definición, los métodos finales no pueden ser sobreescritos).
Este tipo particular de polimorfismo también permite aprovechar las clases abstractas. La herencia de una clase abstracta genérica permite agrupar clases que comparten métodos comunes pero que tienen implementaciones diferentes.

Otro tipo de polimorfismo es conocido como ***sobrecarga*** de métodos. En Java es posible definir el mismo método más de una vez utilizando diferentes parámetros de llamada en cada definición. Cuando tenemos necesidad de que un método en una clase tenga el mismo comportamiento para distintos tipos de datos en sus argumentos o el valor que regresa, utilizamos la propiedad de sobrecarga de métodos.

Finalmente, el polimorfismo también se expresa implementando los mismos métodos en clases diferentes, lo que se denomina uso de ***interfaces***.

Interfaces

Una interfaz es una clase que proporciona un mecanismo para abstraer los métodos a un nivel superior, ya que es como un molde donde solamente permite declarar nombre de métodos, lista de argumentos, los cuales sobrescribirán las clases que lo implementen.

Una clase implementa una interfaz para que múltiples objetos de clases diferentes puedan ser tratados como si fueran de un mismo tipo común, recogiendo similitudes entre clases no relacionadas, simulando la herencia múltiple. Sin embargo, a pesar de que las interfaces podrían resolver algunos problemas de la herencia múltiple, conceptos diferentes.

En particular:

- No se pueden heredar variables desde una interfaz.
- No se pueden heredar implementaciones de métodos desde una interfaz.
- La herencia de una interfaz es independiente de la herencia de la clase. Las clases que implementan la misma interfaz pueden o no estar relacionadas a través del árbol de clases.

Una clase sólo puede extender de una sola clase pero puede implementar cualquier cantidad de interfaces, permitiendo a cada clase compartir la interfaz de programación independiente de la implementación que hagan otras clases que implementan a la interfaz.

Declaración de interfaces:

```
[public] interface NombreInterfaz [extends superInterfaces]{
        cuerpoDeLaInterfaz
}
```

La declaración básica de una interfaz incluye a la palabra clave interface y el nombre de la interfaz y, ademas, puede contener el especificador de acceso public (es package si se omite) y la lista de superinterfaces de la cual extiende nuestra interfaz (a diferencia de las clases, una interfaz puede extender a cualquier número de interfaces).

El cuerpo de la clase incluye las declaraciones de los métodos terminadas en punto y coma (mismos que no contienen cuerpo), también es posible definir constantes dentro de una interfaz.

Implementación de interfaces:

```
class NombreClase implementes Interfaz{
        cuerpoDeLaClase
}
```

Una interfaz se utiliza al crear una clase que la implemente (mediante *implements*). La clase que implementa a la interfaz debe proporcionar la definición completa a todos los métodos declarados en la interfaz, así como de los métodos declarados como superinterfaces de esa interfaz. Una clase puede implementar más de una interfaz.

Excepciones

Una excepción es un evento (error o caso excepcional) que ocurre durante la ejecución de un programa y detiene el flujo normal de la secuencia de instrucciones de ese programa. Es decir, si hay un error, la aplicación no debería solo terminar y generar un mensaje de error, identificar que ocurrió una excepción, y ser capaz de manejarla de alguna forma.

Cuando se produce una condición excepcional en el transcurso de la ejecución de un programa se lanza un objeto de la clase Throwable. Esta clase tiene dos subclases Error y Exception.

Un Error indica que se ha producido un fallo no recuperable y por lo tanto no hay nada que hacer, concluyendo la ejecución del programa.

Una Excepción indicará una condición anormal que puede ser subsanada para evitar la terminación de la ejecución del programa.

Las Excepciones manejan una estructura jerárquica para su definición, empleando el manejo de clases, y todas heredan de la clase Exception. También es posible crear nuestras propias excepciones extendiendo a esa clase.

throws

Para indicar que en un método puede ocurrir un error y se va a manejar se debe poner en la firma la palabra reservada "throws" seguido del tipo de Exception que se vaya a manejar.

```
MiMetodo() throws Exception{
  ...
}
```

throw

Dentro del cuerpo de los métodos, es la sentencia que lanza una excepción, la cual deberá ser "atrapada" y manejada apropiadamente. Es la forma en la que una clase informa que se ha producido un error o caso excepcional en la ejecución del programa.

```
...
throw new Exception("Mensaje");
...
```

Manejo de Excepciones

Las excepciones lanzadas deben recogerse en un bloque try/catch o try/finally. Un método que maneje errores deberá tener la siguiente sintaxis.

```
NombreMetodo (listaArgumentos) throws TipoThrowable{
      try{
            //sentencias
      } catch(TipoThrowable nombreVariable){
            Sentencias.
      }finally{
            sentencias
      }
}
```

Bloque try

Bloque de código donde se prevé que se genere una excepción. Tiene que ir seguido, al menos, por una cláusula catch o una cláusula finally.

Su sintaxis es:

```
try{
    Sentencias;
    [throw new TipoThrowable("")]
}
```

Bloque Catch

Es el código que se ejecuta cuando se produce la excepción. No debe haber código entre un bloque try y un bloque catch. En este bloque hay que asegurarse de colocar código que no genere excepciones . Se pueden colocar sentencias catch sucesivas, cada una controlando una excepción diferente.. La cláusula catch comprueba los argumentos en el mismo orden en que aparezcan en el programa. En este código debe manejarse el error ya sea, mandando un mensaje de error u otra operación mas complicada.

Su sintaxis es :

```
}catch(TipoThrowable1 nombreVariable1){
    Sentencias java
}catch(TipoThrowable2 nombreVariable2){
    Sentencias java
}...
```

Bloque finally

Es el bloque de código que se ejecuta siempre, haya o no excepción.

Su sintaxis es :

```
}finally{
    Sentencias java
}
```

Paquetes.

En Java, las clases estan organizadas en paquetes, que son librerías que agrupan funciones y clases. Los nombres de los paquetes son palabras separadas por puntos que describen la ruta de localización de una clase, ya que los nombres de los paquetes corresponden a los directorios en los que estan estructuradas las clases.

```
paqueteRaiz.subPaquete1.subPaquete2.subPaquete3.MiClase
```

La palabra clave *package* permite agrupar clases e interfaces en paquetes, y normalmente es la primera línea en una clase. Al momento de crear un paquete hay que tener en cuenta las siguientes condiciones:

- La palabra clave package debe ser la primera sentencia que aparezca en el archivo.
- Todas las clases que se vayan a incluir en el paquete estén en el mismo directorio.

```
//Agrega esta clase al paquete miPaquete.
 package miPaquete;
 ...
```

Para acceder a una clase es necesario importarla, o bien importar todo el paquete que incluye a la clase, esto se hace posible mediante la cláusula *import*.

Import

Dado que en un momento dado podemos utilizar clases que son ajenas al paquete donde se encuentra nuestra clase, los paquetes de clases se cargan con la palabra clave *import* especificando el nombre del paquete como una ruta y nombre de clase.

```
//importa la clase NombreClase del paquete rutaPaquete.
import rutapaquete.Nombreclase;

//Importa todas las clases del paquete rutaPaquete
import rutaPaquete.*;
```

Paquetes de Java

Aunque existe una gran cantidad de paquetes, solo daremos una breve descripción de los principales paquetes.

java.util

Contiene la 'collections framework', o conjunto de clases para manipulación de conjuntos de objetos (colas, pilas, listas, diccionarios, árboles, tablas hash, etc.), Además tiene varios conjuntos de utilidades para manipulación de fecha y hora, generación de números aleatorios y manipulación de ficheros comprimidos en formato ZIP y JAR. El formato JAR (Java Archive) es una extensión del formato ZIP que permite empaquetar clases java compiladas para su ejecución.

java.awt

El AWT (Abstract Windows Toolkit) proporciona el entorno base para todas las clases de manipulación de la interfaz gráfica del usuario. El AWT apareció en la versión 1.0 del JDK y fue parcialmente sustituido y sustancialmente mejorado en la versión 1.1 (con el conjunto de componentes conocido como swing). Actualmente se mantiene porque es la base del swing aunque muchos de sus elementos ya no se usan.

javax.swing

Pertenece al API extendido. Conjunto extenso de clases para la configuración de la interfaz gráfica de usuario. Reemplaza parcialmente al AWT. Su característica más importante es que es independiente de la plataforma.

java.Math

Proporciona clases para realiza cálculos aritméticos de cualquier presión. Así como funciones matemáticas generales (Trigonometría, aleatorización, etc.).

java.sql

Proporciona el API para el acceso y proceso de datos organizados en bases de datos relacionales y accediendo con mecanismos de lenguaje SQL. Esta API se denomina también JDBC (Java Data Base Conectivity).

Manejo de Bases de Datos con Java

Como se mencionó anteriormente existe un API JDBC (java.sql) que permite ejecutar instrucciones en lenguaje estándar de acceso Base de Datos(SQL). Básicamente sus funciones son:

- Realizar la conexión a la base de datos
- Enviar sentencias SQL a la base de datos
- Procesar los resultados obtenidos de la Base de Datos.

Debido a que existen varios manejadores de base de datos que usan la plataforma PC como cliente para acceder a un servidor y cada uno de estos manejadores tienen características diferentes por ser orientadas para rendimiento, interfaz y programación diferente. Por lo tanto surge una estandarización, llamada conectividad Abierta de Base de Datos (ODBC por sus siglas en Ingles[4]) cuyo propósito es hacer independiente la programación del manejador de base de datos.

JDBC tiene una serie de clases y métodos para permitir a cualquier programa Java una forma homogénea de acceso a sistemas de bases de datos. Se creó debido a que el ODBC es una interfaz escrita en lenguaje C, el cual al no ser portable, haría que las aplicaciones Java también perdiesen portabilidad, además de que el ODBC debe instalarse y configurarse manualmente mientras que JDBC son automáticamente instalables, portables y seguros.

Acceso a Base de Datos

Para que una aplicación pueda hacer operaciones en una Base de Datos, ha de tener una conexión con ella, a través de un controlador o driver, que convierte el lenguaje SQL a sentencias de Base de datos. El driver JDBC-ODBC proporciona acceso a uno o más drivers ODBC a través de JDBC, sin embargo, tiene el inconveniente de problemas de rendimiento debido a la conversión de transacciones JDBC a ODBC.

La manera para conectarse con una Base de Datos es:
Indicar el Driver que se usará en el código por medio de la clase DriveManager. Esta se especifica el Driver a utlizar y el nombre de la Base de Datos. La conexión se guardará en una instancia de la clase Connection la cual tendrá la referencia a la Base de Datos.

```
Try{
     Connection  conexión =
          DriverManager.getConnection("jdbc:odbc:NombreBD)
}catch (Exception e){
     System.out.println("No se pudo cargar el JDBC-ODBC")
}
```

[4] ODBC Open DataBase Connectivity.

Ejecución de Sentencias.

Para ejecutar sentencias SQL se crea una instancia de la clase PreparedStatement por medio de la conexión, pasándole la sentencia SQL. Esta clase tiene la particularidad de que la sentencia SQL es compilada antes de ejecutarse por lo que aumenta la rapidez de la ejecución.

Una vez que se creo la instancia de la clase PreparedStatement tenemos dos métodos para ejecutarla:

- *executeUpdate.* Ejecuta la sentencia regresando el número de registros afectados. Regresa 1 cuando se afectaron uno o mas registros, 0 cuando no se afecto ningún registro y un número negativo si ocurrió algún error. Generalmente se utiliza para sentencias Insert[5],Update[6],Delete[7].
-
- *executeQuery.* Ejecuta la sentencia regresando una instancia de la clase ResultSet, al cual contiene las filas o registros afectados. Generalmente se utiliza para la sentencia Select[8].

Ejemplo:

```
try{
    Connection conexión =
            DriverManager.getConnection("jdbc:odbc:NombreBD
    PreparedStatement stmBorra = connexion.preparedStatement(
                "DELETE FROM Tabla WHERE IdTabla = ?");

     /*Se agrega un parámetro a la sentencia SQL que en
     este caso es el valor de IdTabla.*/
      stmtBorra.setInt(1,2);
      if(stmtBorra.executeUpdate()!=1){
            System.out.println("No se borro el dato.");
      }

      PreparedStatement stmSeleccion =
      conexion.preparedStatement(
            "SELECT * FROM Tabla WHERE IdTabla =?");

      /*Se le agrega un parámetro a la sentencia SQL que en este
      caso es el valor del IdTabla a buscar.*/
      stmtSeleccion.setInt(1,23);
      ResultSet rsDatosSeleccion = stmtSeleccion.executeQuery();
}catch (Exception e){
            System.out.println("No se pudo cargar el JDBC-ODBC")
}
```

[5] Inserción de datos.
[6] Actualizar registro
[7] Borrar registro
[8] Seleccionar un registro

Manipulación de Datos

Una vez que se ejecutó la sentencia y se regresa una instancia ResultSet, se recorre este objeto para manipular los datos seleccionados.

*El método **next** mueve algo llamado cursor (apuntador) a la siguiente fila y hace que esa fila (llamada fila actual) sea con la que podamos operar. Como el cursor inicialmente se posiciona justo antes de la primera fila de un objeto **ResultSet**, primero debemos llamar al método **next** para mover el cursor a la primera fila y convertirla en la fila actual. Sucesivas invocaciones del método **next** moverán el cursor de línea en línea de arriba a abajo.*

Para recuperar un valor de los datos utilizamos una serie de métodos que tienen el prefijo "get" el cual indicará el tipo de valor a recoger. Tiene como parámetros una cadena con el nombre de la columna o un número entero para señalar el número de columna.

Ejemplo:

```
try{
      Connection  conexión =
      DriverManager.getConnection("jdbc:odbc:NombreBD
      PreparedStatement stmBorra = connexion.preparedStatement(
                        "DELETE  FROM Tabla WHERE IdTabla = ?");

      stmtBorra.setInt(1,2);
      if(stmtBorra.executeUpdate()!=1){
            System.out.println("No se borro el dato.");
      }

      PreparedStatement stmSeleccion = conexion.preparedStatement
                        ("SELECT * FROM Tabla WHERE IdTabla =?");

      stmtSeleccion.setInt(1,23);

      ResultSet rsDatosSeleccion = stmtSeleccion.executeQuery();
      while(rsDatosSeleccion.next()){
         /*Se especifica la columna en numero*/
         System.out.println(rsDatosSeleccion.getString(1);
         /*Se especifica la columna con nombre*/
         System.out.println(rsDatosSeleccion.getInt
                                          (ColumnaValorEntero");
      }
}catch (Exception e){
            System.out.println("No se pudo cargar el JDBC-ODBC")
}
```

Serialización

Un aspecto importante que Java toma en cuenta es que los objetos y su estado, formados en tiempo de ejecución, no se destruyan y puedan ser reutilizados entre máquinas virtuales diferentes en un entorno distribuido, a esto se le llama persistencia de objetos. Muchas ocasiones la persistencia de los datos se controla a través de archivos o bases de datos comerciales, dependiendo de la complejidad y los recursos que se tengan. Los archivos se recomiendan para aplicaciones pequeñas debido a que es sencillo trabajar con ellos y no son limitados a su uso por un programa.

En el caso de las bases de datos relacionales y orientadas a objetos que funcionan muy bien con programas que requieran características de bases de datos: transacciones, bloqueo de registros, índices, etc. Obviamente, involucra un costo mayor, pueden llegar a ser difíciles de manejar y exigen mayores conocimientos.

En un entorno de programación distribuida, los sockets resultan ser flexibles y fáciles de utilizar, al igual que los archivos, pero presentan los mismos problemas cuando se transmiten formatos de datos complejos. Las aplicaciones distribuidas basadas en CORBA, por ejemplo, disponen de facilidades para transmitir objetos pero es una solución costosa, y en cierta forma limitada.

La **serialización de objetos** en Java proporciona una solución intermedia para salvar objetos en archivos y/o transmitirlos a través de la red. Incluso en grandes proyectos que utilicen bases de datos comerciales o comunicaciones middleware, puede ser un formato válido para archivos auxiliares o comunicaciones variadas. Tanto RMI como el API de los JavaBeans utilizan la serialización para guardar y transmitir objetos. Por lo tanto, en toda aplicación Java que tenga que ver con persistencia o distribución de objetos, la serialización de objetos es una buena herramienta.

La serialización permite escribir y leer objetos en *streams*, sean éstos archivos o sockets. Esto proporciona una forma sencilla de guardar tanto objetos individuales como grandes estructuras de objetos en archivos, o enviarlos a través de la red.

Desde la perspectiva del programador, gran parte de dicho trabajo se realiza automáticamente. El mecanismo de serialización mantiene el control sobre los tipos de los objetos, las referencias entre ellos y muchos detalles de cómo están almacenados los datos. El API de serialización está muy estructurado, de tal modo que en muchos casos se puede manejar directamente con facilidad, mientras permite realizar acondicionamientos muy complejos en cada caso.

En la mayor parte de los casos que necesiten serializar un objeto, se necesitará añadir en la declaración de la clase la implementación de **Serializable**. Esta interfaz no tiene métodos que deban ser implementados, sin embargo, cualquier campo de datos que implemente esta interfaz también debe ser serializable. Todos los tipos básicos de datos que se utilizan como: int, String, Array, Vector o Hashtable son serializables.

El envío de objetos al canal de comunicaciones es realizado por la clase **ObjectOutputStream**, que implementa la interfaz genérica **ObjectOutput**. Esta interfaz es una extensión , a su vez, de la interfaz **DataOutput**, que le añade la capacidad de escribir objetos, así como tipos básicos como cadenas y números. El objeto **ObjectOutputStream** se crea por encima de cualquier otro **OutputStream**,

como por ejemplo un archivo o un socket. Ahora ya se puede utilizar el método *writeObject()* para enviar datos al canal abierto hacia el archivo. Finalmente se vuelca todo el contenido y se cierra el canal.

Para leer objetos de un canal de entrada se utiliza la clase **ObjectInputStream**, para implementar la interfaz **ObjectInput**. Del mismo modo que con ObjectOutputStream, un ObjectInputStream se crea a partir de un **InputStream**, utilizando el método *readObject()* para leer un objeto de ese canal, el cual regresa un objeto que debe ser moldeado. Se debe saber el tipo real del objeto mediante el método *getClass()* o utilizando las capacidades de reflexión que permite Java2.

A través de la interfaz Serializable la mayoría del trabajo que interviene en la serialización de objetos se realiza automáticamente, sin embargo el programador tiene varias opciones para poder configurar esa serialización.

Notificación de Cambios entre Objetos.

La notificación de cambios de estado entre objetos es útil cuando se requiere una dependencia entre objetos. Es decir, cuando un objeto cambie de estado se necesita que otro objeto este enterado de este cambio. En un patrón cercano al Java Event Model. Java permite hacerlo dinámicamente sin la generación y comunicación de eventos, si bien el mecanismo utilizado lo permite, por medio de dos Clases: Observable y Observer.

Las clases cuyos instancias se desea monitorizar derivan de la clase *Observable* la cual implementa funciones que permiten:

1. la gestión de subscripciones(añadir objetos que observen estos cambios)
2. la notificación de cambios.

Los objetos que derivan de esta clase son los encargados de determinar cuando se han producido cambios en su estado y cuando debe comunicarlos a sus observadores.Así mismo debe existir una clase que herede de Observable para implementar un método protegido: setChanged(), el cual deberá ser invocado antes de notificar a los observadores.

Por ejemplo una clase que quiera notificar a otros objetos de cambios en su estado debe tener implementados los siguientes métodos:

- addObserver (Observer o). Añade un objeto Observador.
- notifyObservers(Object o). Avisa a los observadores de su cambio de estado pasándole un objeto para su manipulación.

Los objetos que observan los cambios, que deben implementar el interfaz *Observer*, son quienes se deben subscribirse al mecanismo de notificación(añadirse como objetos observadores) y se responsabilizan de actualizar su visión del objeto observado en función del nivel de cambio detectado. Implementa el método de la clase Observer update(Observable ob, Object cambio). Este método es el que se encarga de recibir las actualizaciones y llamar a métodos que las manejen..

Descripción de aplicación distribuida

Comprender las aplicaciones distribuidas requiere del estudio de la arquitectura Cliente-Servidor, ya que es ésta la que proporciona una forma eficiente de utilizar todos los recursos de la organización, de tal forma que, la seguridad y fiabilidad proporcionada por los entornos centralizados se traspasa a la red de área local. De igual forma, se puede considerar como un modelo para el desarrollo de sistemas de información, en el que las transacciones se dividen en procesos independientes que cooperan entre sí para el intercambio de información, recursos o servicios.

En la arquitectura cliente servidor, la funcionalidad de la aplicación de divide en dos papeles bien definidos: el cliente y el servidor. Los sistemas distribuidos conforman el último nivel de la computación cliente-servidor, al ofrecer la funcionalidad en forma de objetos, en vez de diferenciar entre las distintas partes de la aplicación. Esta perspectiva de objetos es muy similar al empleado en el paradigma de orientación a objetos, por lo que los sistemas distribuidos aportan a los sistemas las características propias de los lenguajes orientados a objetos, ala vez que implementan las propias de los sistemas descentralizados.

Las nuevas tecnologías de distribución de funciones y datos en una red, permiten desarrollar aplicaciones distribuidas de una manera transparente. Si las funciones de la aplicación están diseñadas adecuadamente, éstas se pueden mover de una máquina a otra sin modificaciones y sin necesidad de alterar los programas que las invocan.

Dentro de esta arquitectura se llama "Cliente" al sistema que inicia una comunicación y "Servidor" al sistema que responde a las comunicaciones. En este modelo las aplicaciones se dividen de forma que el "Servidor" contiene los componentes comunes a los usuarios (acceso a Bases de Datos, procesamiento de comandos, etc.), mientras que en el "Cliente" reside sólo lo que es particular a cada usuario (configuraciones personales, interfaz, etc.).

Los Clientes son aquellos programas con los que el usuario interactúa de forma gráfica y realizan funciones como :

- *Manejo de interfaces del usuario*
- *Captura y validación de datos de entrada*
- *Generación de consultas e informes.*

Por su parte, los Servidores son aquellos que proporcionan un servicio al Cliente, devolviéndole los resultados solicitados. Generalmente se encargan de manejar los interbloqueos[9], recuperación ante fallas, seguridad, etc., razón por la que la plataforma de hardware sobre la que trabaja un servidor es más poderosa que la de un cliente.

[9] Solicitudes de los usuarios a un mismo registro o datos, de esta forma, se evita que la misma información sea alterada simultáneamente, dando lugar con esto a inconsistencia o mal funcionamiento del sistema.

Las principales funciones de un Servidor son :

- *Gestión de periféricos*
- *Control de accesos concurrentes*
- *Enlace con otras redes*
- *Ofrecimiento de respuestas a las solicitudes enviadas por los clientes*

Asi, las principales caracteristicas de esta arquitectura son :

- *El servidor presenta a todos los clientes de una interfaz única y bien definida*
- *El cliente no necesita conocer la lógica del servidor, sólo la interfaz externa*
- *El cliente no depende de la ubicación fisica del servidor*
- *Los cambios en el servidor implican pocos o ningún cambio en el cliente*

Las caracteristicas funcionales de la arquitectura cliente-servidor dependen de las funciones que asuma cada uno, teniendo cinco niveles :

1. Presentación distribuida .- *El cliente asume parte de las funciones de presentación de la aplicación mientras que el servidor se encarga de ejecutar todos los procesos y almacenar la totalidad de los datos.*
2. Presentación remota .- *La aplicación está soportada directamente por el servidor, mientras que la presentación es remota y reside en el cliente.*
3. Proceso distribuido o cooperativo .- *La lógica de los procesos se divide entre los componentes del sistema. Se deben definir los servicios e interfaces de forma que los papeles entre cliente y servidor sean intercambiables, excepto en el control de los datos que es responsabilidad del servidor.*
4. Gestión remota de datos .- *El cliente realiza tanto funciones de presentación como de control de procesos. Por su parte, el servidor sólo se encarga de almacenar los datos.*
5. Base de datos distribuidas .- *Es similar al anterior, pero la administración de los datos se divide entre el cliente y el servidor.*

El ofrecer los servicios distribuidos en forma de objetos, hace que el diseño y desarrollo se realice con interfaces bien definidas que favorecen la reutilización y modularidad de los sistemas, logrando así un diseño más flexible. Las características que ofrece un sistema distribuido se resumen en: *distribución, transparencia, integridad de datos, tolerancia a fallos, disponibilidad, capacidad de recuperación, autonomia de los objetos, concurrencia de programas, concurrencia de objetos y mejora del rendimiento.*

Para fines de este proyecto, se hará referencia a EJB, como la tecnología que define la arquitectura para el desarrollo de aplicaciones enfocadas a objetos distribuidos transaccionales, eso es, software de componentes del lado del servidor.

RMI

El sistema de Invocación Remota de Métodos (RMI) de Java permite a un objeto que se está ejecutando en una Máquina Virtual Java (VM) llamar a métodos de otro objeto que se encuentra en otra VM diferente. Las aplicaciones RMI normalmente comprenden los programas separados del cliente y del servidor. El servidor crea objetos remotos, hace accesibles las referencias a dichos objetos y espera que los clientes llamen a estos métodos u objetos remotos. El cliente obtiene la referencia remota de los objetos remotos del servidor y llama a sus métodos. RMI proporciona el

mecanismo por el que cliente y servidor se comunican, cuando se trata de una aplicación, se dice que es una ***aplicación de objetos distribuidos.***

Una de las razones por las que se utiliza esta tecnología es por la habilidad de RMI de descargar código de una clase de un objeto en caso de que la clase no esté definida en la máquina virtual del receptor. Los comportamientos del objeto pueden ser transportados entre puntos por medio de la máquina virtual.

Una aplicación distribuida construida utilizando RMI de Java está compuesta por interfaces y clases. Las interfaces definen métodos y las clases los implementan. Los objetos con métodos que pueden ser llamados por distintas máquinas virtuales son los objetos remotos. RMI trata a un objeto remoto de forma diferente que al resto, en lugar de hacer una copia de la implementación del objeto en la máquina virtual que lo recibe, RMI pasa un **stub** para un objeto remoto, el cual actúa como representación local para la referencia remota que se necesita.

RMI/IIOP

Dado que ya se analizó las características de RMI, nos encontramos con un problema: portabilidad y comodidad al programar. Al utilizar RMI utilizamos un protocolo llamado Java Remote Method Protocol(JRMP), el cual no puede conectarse con otros protocolos, limitando la comunicación entre programas escritos en Java. Por otra parte, CORBA(Common Object Request Broker Architecture), es un modelo de programación de objetos distribuidos que soporta varios lenguajes de programación.

Muchas veces se tenía que escoger entre la cómoda programación de RMI(la cual sólo puede conectarse con otros programas que manejen el mismo protocolo) y la interoperabilidad de CORBA, cuya programación es laboriosa, debido a su operación mediante interfaces llamadas Interface Definition Language(IDL).

Por lo tanto se desarrollo RMI sobre IIOP(Internet Inter-ORB Protocol) el cual combina ambas características, permitiendo la comunicación entre varios protocolos.

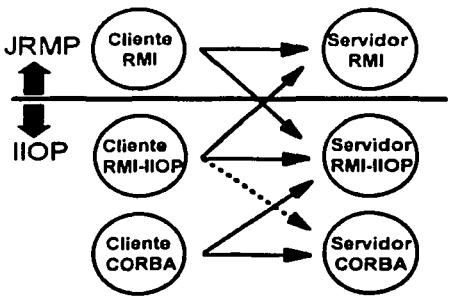


Ilustración 1: RMI-IIOP

Descripción de aplicación empresarial

La mayor importancia de las nuevas tecnologías de la información y su creciente presencia en los diversos ámbitos de la industria moderna (centros de control, grandes y pequeñas empresas, radiodifusoras, televisoras, etc.) y sus productos finales conlleva cada vez más a la presencia de programas informáticos que gobiernan muchas de sus prestaciones, o bien, como herramientas empleadas por el cliente en su beneficio.

Cuando la calidad se tornó en un concepto fundamental en la prestación de servicios o elaboración de productos, las empresas empezaron a invertir más en todos los aspectos que contribuyeran a incrementar la calidad. Esto las obligó a llevar un control intenso de sus procesos, valiéndose del elemento informático para ello. Implementando sistemas como herramientas de control, las empresas se ven en una nueva necesidad, obtener software de calidad.

La calidad del producto del software se diferencia entre la calidad del producto software y la calidad del proceso de desarrollo de éste. Sin embargo, las metas que se establezcan para la calidad del producto, determinarán los objetivos a establecer de calidad del proceso de desarrollo, ya que la calidad del primero va a depender, entre otros aspectos, de ésta. Sin un buen proceso de desarrollo es casi imposible obtener un buen producto.

También es importante destacar que la calidad de un producto de software debe ser considerada en todos sus estados de evolución (especificaciones, diseño, código, etc.). No basta con verificar la calidad del producto una vez finalizado, cuando los problemas de mala calidad ya no tienen solución o su reparación es muy costosa.

La problemática general a la que se enfrenta el software es solucionada mediante las herramientas que se ofrecen en cada una de las fases de desarrollo del mismo. Siendo algunos problemas que enfrenta :

- Aumento constante del tamaño y complejidad de los programas.
- Carácter dinámico e iterativo a lo largo de su ciclo de vida, es decir que los programas a lo largo de su vida, cambian o evolucionan de una versión a otra para mejorar las prestaciones con respecto a las anteriores.
- Dificultad de conseguir productos totalmente depurados.

En el caso del presente proyecto, la calidad en la fase de desarrollo se consigue con la tecnología EJB (Enterprise Java Beans). EJB define una arquitectura para el desarrollo y despliegue de aplicaciones basadas en objetos distribuidos transaccionales, software de componentes del lado del servidor. Las organizaciones pueden construir sus propios componentes o comprarlos a vendedores de terceras partes. Estos componentes del lado del servidor, llamados **beans enterprise**, son objetos distribuidos que están localizados en contenedores JavaBeans Enterprise y proporcionan servicios remotos para clientes distribuidos a lo largo de la red.

Descripción de EJB

Este apartado busca dar una introducción resumida a la teoría EJB, para mayor detalle, referirse a los apartados del desarrollo del caso práctico.

Como se ha mencionado previamente, EJB es una tecnología basada en componentes. Definido para una arquitectura de sistemas transaccionales de objetos distribuidos. La especificación manda un modelo de programación; es decir, convenciones y un conjunto de clases e interfaces que crean el API EJB. Este modelo proporciona a los desarrolladores de Beans y a vendedores de servicios EJB, una plataforma de desarrollo común. Asegurando la portabilidad de los sistemas.

Los EJB son las componentes del modelo de desarrollo de aplicaciones de Enterprise de Sun. Un EJB es un elemento de lógica o de persistencia, que, acompañado de la presentación basada en tecnología Servlet+JSP o aplicaciones Java, forman los elementos java programables del modelo J2EE(Java2 Enterprise Edition). Es importante notar que J2EE es una especificación, es decir un estándar y no un producto particular.

La idea de fondo tras un J2EE es que, dentro del proceso de desarrollo se programe solamente lo que es particular de la aplicación y, que los elementos comunes a la mayoría de las aplicaciones Enterprise sean provistos por el contenedor. De esta manera, elementos como transacciones, seguridad, middleware, pool de conexiones entre otras cosas, sean provistos por el contenedor y no por el sistema operativo.

Una aplicación J2EE con sus clases, beans y descriptores se instala en un servidor de aplicaciones, que es quien provee los servicios necesarios para que la aplicación funcione. La arquitectura J2EE permite dividir la presentación de la lógica y de la persistencia, lo que es muy importante para que una aplicación sea mantenible en el tiempo.

Un EJB tiene al menos una clase (la implementación) y dos interfaces (la interfaz Home y la interfaz externa). La interfaz externa puede ser remota o local. La diferencia es que la remota utiliza RMI para hacer llamadas a los métodos y la local no, sin embargo, ésta última necesita que los objetos estén en la misma máquina virtual. La implementación tiene los métodos globales para el tipo del bean.

Existen tres clases de beans : de sesión, entidad y orientados a mensajes. Los de sesión implementan la lógica de la aplicación; los de entidad representan los datos de la aplicación y; los orientados a mensajes representan la comunicación que se da entre beans mediante mensajes, permitiendo interacciones asincrónicas.

Para crear un componente EJB del lado del servidor son necesarias dos interfaces que definen los métodos de negocio del bean, además de la implementación real de la clase bean. El cliente utiliza una interfaz pública del EJB para crear, manipular y eliminar beans del servidor EJB.

Los EJB son accedidos por el contenedor, quien, a su vez, es llamado por el cliente a través de la red usando sus interfaces **remoto** y **home**. Estas interfaces exponen las capacidades del bean y proporcionan los métodos necesarios para crear, actualizar, borrar e interactuar con el bean.

Interfaces Remota y Home

Cada vez que el cliente solicita un método, el contenedor accede al bean y maneja todo el proceso. La interfaz home representa los métodos del ciclo de vida del componente (crear, encontrar, destruir) mientras que la interfaz remota representa los métodos de negocio del bean. Extienden de **javax.ejb.EJBObject** y **javax.ejb.EJBHome**, las cuales definen un conjunto estándar de métodos de utilidad y proporcionan tipos base comunes para todas las interfaces remotas y home.

Los clientes usan la interfaz home del bean para obtener referencias a la interfaz remota del bean, la cual, define los métodos del negocio como métodos accesorios o modificadores para cambiar el nombre del cliente, o métodos de negocio que realizan tareas como, en el caso del presente proyecto, dar de alta una cuenta de usuario.

```
ClienteHome _clienteHome = /*Obtener la referencia que implementa
                            la interfaz home*/.
/*Utilizar la interfaz home para creat una instancia del bean del
   Cliente*/
Cliente cliente = clienteHome.create(_strIDCliente);

/*En este momento ya es posible utilizar un método del negocio*/
cliente.metodo();
```

La interfaz remota es subclase de la interfaz javax.ejb.EJBObject que es una subclase de java.rmi.Remote; define métodos accesorios y modificadores para leer y actualizar información sobre un concepto del negocio. Esto es usual de los EJB de tipo **entidad**, que representan al objeto del negocio persistente; objetos de negocio cuyos datos se almacenan en una base de datos, siendo éstos, **datos del negocio**.

Los métodos del negocio representan las tareas y no los datos, Entonces se está hablando de un EJB de tipo **sesión**. Son agentes que realizan un servicio, como dar de alta a un usuario. Este tipo de EJB utilizan a los beans entidad para realizar sus tareas, es decir, éstos no acceden por sí mismos a la base de datos. Por cada interfaz remota hay una clase de implementación; un objeto de negocio que realmente implementa los métodos de negocio definidos en la interfaz remota.

Métodos de Ciclo de Vida

Además de una interfaz remota, todos los beans tienen una interfaz home, la cual proporciona métodos de ciclo de vida para crear, destruir o localizar beans. Éstos están separados de la interfaz remota porque los comportamientos que representa no son específicos para un solo ejemplar del bean La interfaz home extiende de la interfaz javax.ejb.EJBHome, la cual a su vez, extiende de la interfaz java.rmi.Remote.

Los métodos que conforman el ciclo de vida de un bean son :

- `create()` .- Utilizado para crear una nueva entidad. Esto resultará en un nuevo registro en la base de datos. Se pueden tener muchos tipos de create, sin embargo, el tipo de retorno debe ser el mismo tipo de la interfaz remota.

- `load()` .- Utilizado para buscar la entidad que acaba de crearse como confirmación de su creación. Esto resultará en la consulta de un registro en la base de datos.
- `remove()` .- Utilizado para destruir a un bean en particular. Esto resultará en la eliminación del registro en la base de datos.
- `store()` .- Utilizado para actualizar un bean previamente creado.
- `Activate()` .- Utilizado cuando se activa la referencia remota.
- `Passivate()` .- Utilizado cuando se quita la referencia remota.
- `finders()` .- Métodos de este tipo se utilizan para buscar ejemplares específicos del bean. Se pueden utilizar y definir tantos métodos de estos tipos como necesitemos, por ejemplo, findByPrimaryKey(), findAll(), findByNombre(), etc.

EJB Entidad

El bean entidad proporciona una interfaz orientada a objetos a los datos que normalmente serían accedidos mediante el JDBC u otro API.

Existen dos tipos de beans de entidad : Persistencia Manejada por el Contenedor (CMP) y Persistencia Manejada por el Bean (BMP).

Persistencia Manejada por el Contenedor

En los CMP, la persistencia es manejada por el contenedor. Las herramientas de los vendedores se usan para mapear los campos de entidad en la base de datos y no se escribe ningún código de acceso a la misma en la clase del bean. Son más simples para el desarrollador pero más complejos para el servidor. Los accesos a la base de datos se hacen mediante el contenedor, por tanto la herramienta debe ser capaz de manejar la seguridad de los datos. La mayoría de los vendedores EJB soportan la persistencia automática a una base de datos relacional, pero el nivel de soporte varía.

Un EJB se compone de, al menos dos interfaces y una clase de implementación del bean. La clase del bean contiene todos los métodos usuales de la interfaz que implementa, sin embargo, se encuentran vacíos debido a que la lógica de almacenamiento le corresponde al contenedor. Los métodos del negocio se programan a la necesidad del desarrollador sin afectar el funcionamiento del bean.

Los campos que se requieran para el bean entidad son campos manejados por el contenedor, debido a que éste es el responsable de sincronizar su estado con la base de datos, siendo éstos de cualquier tipo de datos primitivo o serializable[10]. Sin embargo, debe tomarse en cuenta que, cualquier campo manejado por el contenedor debe tener su correspondiente en la base de datos para su correcta creación, borrado, carga y actualización de registros.

Completando el modelo relacional, uno o más campos del EJB entidad, se convertirán en la llave primaria, correspondiente a su tabla en la base de datos. Esta llave primaria es la referencia a un único registro, el cual crea el estado del bean. Las llaves primarias compuestas deben ser representadas por una clase especial definida por el desarrollador del bean.

[10] Característica de Java que sirve para la descomposición de información a través de la red, unida al llegar a su destino.

Al crearse un EJB CMP, se llama al método create() de la interfaz home del bean. Posteriormente se puede llegar a los métodos cuando se obtiene la referencia del objeto home, la cual puede declarar cero o más métodos create(), cada uno de los cuales debe tener sus correspondientes métodos **ejbCreate()** y **ejbPostCreate()** en la clase del bean.

Al invocar al create() de la interfaz home, el contenedor delega la llamada al ejbCreate() en el bean. Estos métodos se utilizan para inicializar el estado del EJB antes de que el registro sea insertado, hecho lo cual, pueden ser accedidos sus campos. Sin embargo, antes de servir a cualquier método del negocio, puede hacerlo en el ejbPostCreate(), dicho método permite al EJB realizar cualquier proceso post-creación antes de realizar las peticiones de los clientes.

Ahora bien, una vez creado el bean y comprobada su existencia, es necesario tener métodos de búsqueda específicos[11], no obstante, no existe un lenguaje de consultas estándar definido para los finders, por eso cada vendedor implementará estos de forma diferente. En los CMP, los finders no están implementados con los métodos correspondientes en la clase del bean; los contenedores los implementan cuando el EJB se desarrolla de una forma específica del ventedor. El desarrollador lo único que hace es indicarle al contenedor cómo se manejan cada uno de los métodos.

La interfaz remota obligatoria en todos los beans definirá los métodos del negocio. Los más usuales son los métodos que asignan los campos a variables y aquellos métodos que obtienen tales variables. Tómese en cuenta que, si bien lo común es la obtención de campos para su posterior uso, los métodos del negocio podrían ser más complejos al incluir cálculos o validaciones.

Con los métodos de Retrollamada[12] permiten al contenedor notificar al EJB los eventos de su ciclo de vida. Se encuentran definidos en la interfaz javax.ejb.EntityBean, implementado por los EJB entidad. El método **setEntityContext()** proporciona al EJB una interfaz con el contenedor llamada **EntityContext**, la cual contiene los métodos para obtener información sobre el contexto bajo el cual opera el EJB en un momento dado. Se utiliza para acceder a la información de seguridad sobre quién lo llama; determinar el estado de la transacción o forzar a deshacer una transacción, etc. Esta interfaz sólo se configura una vez en la vida del EJB.

El método **unsetEntityContext()** se utiliza al final del ciclo de vida del bean, antes de que sea sacado de memoria, para quitar la referencia del EntityContext y realizar cualquier tarea de limpieza adicional.

ejbLoad() y ejbStore se invocan, en entidades CMP, cuando el estado del EJB está siendo sincronizado con la base de datos. ejbLoad(), se invoca justo después de que el contenedor haya refrescado los campos del bean. ejbStore() se invoca justo antes de que el contenedor escriba los datos.

---

[11] Finders o métodos de búsqueda específicos.
[12] Métodos que permiten ubicar al bean en el contexto de beans, es decir, la conexión con la base de datos dentro de un contenedor.

Persistencia Manejada por el Bean

En los BMP, la persistencia es manejada por el bean, es decir, el bean contiene el código de acceso a la base de datos (normalmente JDBC) y es responsable de leer y escribir en la base de datos en su propio estado. Tienen mucha ayuda debido a las advertencias sobre las actualizaciones o consultas necesarias desde la base de datos. El contenedor también puede manejar cualquier bloqueo o transacción para que la base de datos sea íntegra.

Este tipo de persistencia le da oportunidad al desarrollador de realizar operaciones más complejas o para utilizar una fuente de datos no soportada por el contenedor. La modificación respecto al CMP en las interfaces home y remota no se ve diferenciada en lo más mínimo. La diferencia radica en el código del bean, es decir, la implementación de los métodos se sobreescribe.

Los métodos **ejbLoad()** y **ejbStore()** son utilizados de manera distinta. En estos métodos se contiene código para leer los datos de la base de datos y para actualizarlos, respectivamente. Se llaman cuando el servidor EJB decide el momento de leer o escribir los registros. Normalmente ejbLoad() es invocado al principio de una transacción, justo antes de que el contenedor delegue un método de negocio al bean.

ejbLoad() usa la referencia **ejbContext()** hacia el EntityContext del bean para obtener la llave primaria, asegurando el registro requerido. ejbStore() es invocado al final de la transacción justo antes de que en contenedor trate de enviar los cambios a la base de datos. En ambos métodos el EJB sincroniza su estado con la base de datos mediante JDBC, usando a su vez, **getConnection()**. Tales conexiones a la base de datos se obtienen desde el contenedor usando un contexto JNDI llamado "JNDI Environment Naming Context" (ENC), el cual, proporciona el acceso a almacenes de conexiones JDBC transaccionales a través de un javax.sql.DataSource.

Para insertar y eliminar entidades se implementan los métodos ejbCreate() y ejbRemove() con la lógica de acceso a base de datos similar a los anteriores.

Respecto a los finders, el bean es el responsable de su implementación. Por cada finder definido en la interfaz home, debe haber su correspondiente implementación en el bean, los cuales regresan la llave primaria resultante al contenedor para ser convertida en referencia a beans que son devueltas al cliente.

EJB Sesión

Como ya se ha mencionado los EJB de sesión representan las tareas o acceden a la base de datos mediante los beans de entidad.

Los EJB de sesión pueden ser : sin estado (*stateless*) y con estado (*stateful*). Los EJB de sesión sin estado configuran métodos del negocio que se comportan como procedimientos, es decir, operan sólo sobre los argumentos que se les pasan; son temporales, ya que no mantienen el estado del negocio entre llamadas. Los EJB de sesión con estado encapsulan la lógica del negocio y el estado específico del cliente; mantienen el estado del negocio entre llamadas al método en memoria y no es persistente.

EJB de Sesión sin Estado

Los métodos del negocio en un EJB de sesión sin estado actúan como los procedimientos tradicionales en un monitor de procesamiento de transacciones legales. Cada llamada es independiente de las anteriores. Los EJB de sesión sin estado son más sencillos de manejar por el contenedor, por ello tienden a procesar las peticiones muy rápido y utilizan menor recursos, aunque no tienen referencia alguna entre una llamada a método y la siguiente.

El método ejbCreate() es invocado al principio de su vida y es invocado una sola vez. Es conveniente para inicializar recursos de conexión y variables que serán utilizadas por el EJB sin estado durante su vida. Los métodos ejbCreate() y ejbRemove() son invocados sólo una vez por el contenedor; cuando el bean se crea por primera vez y cuando es destruido. Una invocación a create en el home le proporciona al cliente una referencia del tipo de bean sin estado y remove() invalida la referencia. El contenedor decide cuando los beans son creados y destruidos e invocará a los métodos en tal situación, permitiendo ser compartidos entre muchos clientes sin impactar en las referencias de ellos. Tan pronto como un bean completa una llamada de método para un cliente, automáticamente puede servir a otro cliente, por ello se requiere la conexión constante al medio de URL's que pueden localizar a los beans entidad.

De igual forma que EntityContext le dice al bean entidad en dónde debe estar, **setSessionContext()** proporciona al bean una referencia a **SessionContext.** Los métodos ejbActivate() y ejbPassivate() no se implementan en el bean de sesión sin estado.

EJB de Sesión con Estado.

Al manejar esta modalidad de Session beans, se mantiene un estado entre llamadas a métodos, es decir, como los clientes no comparten los beans[13], es posible mantener un estado de conversaciones, es decir, un estado de negocio compartido por métodos del mismo bean. El cliente se ve liberado de la responsabilidad de seguir la pista del estado de la sesión, ya que esto lo hace el bean.

Para conservar recursos, los EJB de sesión con estado pueden implementar el Passivate() cuando no están en uso por el cliente, esto es, el estado conversacional del bean es escrito en un almacenamiento secundario (generalmente en disco) y el bean creado es eliminado de la memoria. La referencia del cliente al bean no se ve afectada al aplicar el passivate(), permanece viva y utilizable mientras ocurre. Cuando el cliente invoca a un método del EJB, el contenedor activará el bean en el almacenamiento secundario.

Losa desarrolladores utilizan ejbActivate() y ejbPassivate() para cerrar recursos abiertos y para hacer otras operaciones de limpieza antes de que el estado del EJB sea escrito en almacenamiento secundario y sea eliminado de la memoria.

[13] A diferencia de los beans de sesión sin estado en los que pueden ser compartidos entre clientes.

Contenedor de EJB

Los EJB son componentes que se ejecutan en un entorno especial llamado **contenedor**, el cual contiene y maneja al bean de igual forma que un Servidor Web Java contiene un servlet. Los aspectos que controla el contenedor son *seguridad, persistencia, transacciones, concurrencia y acceso a la base de datos.*

El contenedor aísla al EJB de accesos directos por parte de aplicaciones cliente. Cuando la aplicación cliente invoca un método remoto de un EJB, el contenedor intercepta la llamada para asegurar que la persistencia, las transacciones y la seguridad son aplicadas apropiadamente a cada operación que el cliente realiza en el EJB. El contenedor maneja estos aspectos de forma automática, por eso el desarrollador no tiene que escribir este tipo de lógica dentro del propio código del EJB. El desarrollador de EJB puede enfocarse en encapsular las reglas del negocio, mientras el contenedor se ocupa de todo lo demás.

Como el contenedor maneja muchos EJB, se consume memoria y procesos, los contenedores almacenan los recursos y manejan los ciclos de vida de los EJB con cuidado. Cuando el EJB no se está utilizando por los clientes, se le saca de memoria y se trae de vuelta cuando es necesario. Como las aplicaciones cliente no tienen acceso directo a los EJB ya que el contenedor trata con el cliente y el EJB, la aplicación cliente se despreocupa completamente de las actividades de control de recursos del contenedor.Un EJB depende del contenedor para lo que necesite. Si requiere acceder a una conexión JDBC o a otro EJB, lo hace a través del contenedor; si requiere conocer a quien lo llama, obtiene una referencia a sí mismo, o accede a las propiedades a través de su contenedor.

Cada EJB implementa un subtipo de interfaz EnterpriseBean que define muchos métodos, cada uno de los cuales alerta al EJB sobre un evento diferente en su ciclo de vida y el contenedor llamará a estos métodos para notificar al bean sobre el momento de su activación, la persistencia de su estado a la base de datos, finalización de la transacción, eliminación del bean, etc.

Todo EJB obtiene un objeto **EJBContext**, que es una referencia directa a su contenedor. La interfaz EJBContext proporciona métodos para interactuar con el contenedor para que el bean pueda solicitar información sobre su entorno como la identidad de sus clientes, el estado de una transacción o para obtener referencias remotas a sí mismo.

La interfaz Java de Nombres y Directorios (JNDI) es una extensión estándar de la plataforma Java para acceder a sistemas de nombrado como LDAP, NetWare, sistemas de archivos, etc. Todo bean tiene automáticamente acceso a un sistema de nombrado especial llamado **Environment Naming Context** (ENC). El ENC está controlado por el contenedor y los bean acceden a él usando JNDI. El JNDI ENC permite al bean acceder a recursos como conexiones JDBC, otros EJB y a propiedades específicas para ese EJB.

La portabilidad es el principal valor que trae los EJB ya que asegura que un EJB desarrollado para un contenedor puede migrarse a otro si ese otro ofrece mejor rendimiento. La destreza del desarrollador de EJB puede influenciarse a través de varios contenedores , proporcionando mejores oportunidades para las organizaciones.

Al dividir las tareas de esta forma, dejando al cliente las llamadas a métodos, el contenedor puede y debe asegurarse que la seguridad, transacciones, persistencia, concurrencia y control de recursos, sean tareas simples que permitan al programador enfocarse en las reglas del negocio.

Descriptor de despliegue (Deployment Descriptor).

Como se ha mencionado, el contenedor maneja la persistencia, transacciones, concurrencia y el control de acceso automáticamente para los EJB. La forma de indicarle al contenedor cómo va a controlar estas cosas es, a través del uso de un **descriptor** de desarrollo XML. Cuando un EJB es desplegado dentro de un contenedor, éste lee el descriptor de desarrollo para encontrar las instrucciones de manejo de transacciones, persistencia (en el caso de beans de entidad), y el control de acceso.

El formato que utilizan los beans compatibles con EJB se especifica en un **Document Type Definition**, o DTD de XML. Éste describe el tipo de bean y las clases usadas para la interfaz remota, home y la clase del bean. Incluso especifica los atributos transaccionales de cada método del bean, qué roles de seguridad puede acceder a cada método, y si la persistencia de un bean de entidad es manejada automáticamente por el bean.

Los servidores de aplicaciones EJB normalmente proporcionan herramientas que pueden ser utilizadas para construir descriptores de desarrollo, lo cual simplifica el proceso. Cuando un bean va a ser desplegado, sus archivos *class* de las interfaces home y remota, así como la clase del bean y el descriptor de desarrollo deben empaquetarse en un archivo JAR. El descriptor de desarrollo debe almacenarse en el JAR con un nombre especial : **META-INF/ejb-jar.xml.**

EJB como objeto distribuido

La clase de implementación real del EJB es llamada clase del bean y se convierte en un objeto distribuido. Las interfaces remota y home son tipos de interfaces **Java RMI Remote**. La interfaz java.rmi.Remote se usa con objetos distribuidos para representar el EJB en un espacio de direccionamiento diferente. Un EJB es un objeto distribuido, lo que significa que la clase bean es ejemplarizada y vive en un contenedor pero puede ser accedida por aplicaciones que viven en otros espacios de direccionamiento.

Para hacer que un ejemplar de un objeto esté disponible para otros requiere de esqueletos **(skeleton)** y talones **(stub)**. El skeleton tiene conexión de red con el stub. El stub implementa la interfaz remota por eso se parece a un objeto de negocio, aunque no contiene la lógica del negocio, contiene una conexión socket con el skeleton. Cada vez que se invoca un método del negocio sobre la interfaz remota del stub, ésta envía un mensaje de red al skeleton diciéndole qué método ha sido invocado. Cuando el skeleton recibe un mensaje de red desde el stub, identifica el método invocado y los argumentos, luego invoca al método correspondiente del ejemplar actual, el cual, ejecuta el método de negocio y devuelve el resultado al skeleton, quien lo envía de vuelta al stub.

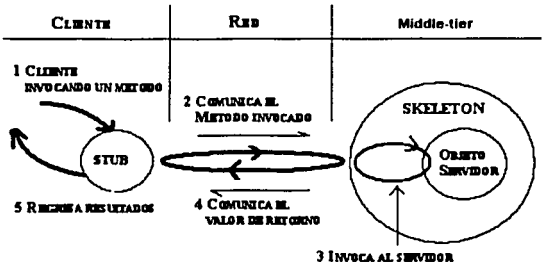


Ilustración 2: Objetos distribuidos.

En EJB, el skeleton para las interfaces remota y home están implementados por el contenedor, no por la clase bean. Esto es para asegurar que cada método invocado sobre estos tipos de referencia por una aplicación cliente son manejados primero por el contenedor y luego delegados al ejemplar del bean. El contenedor debe interceptar aquellas peticiones que vayan al bean para poder aplicar la persistencia (beans de entidad), las transacciones, y controlar los accesos automáticamente.

Los protocolos de objetos distribuidos definen el formado de mensajes de red enviados entre espacios de direccionamiento. La mayoría de los servidores EJB soportan "Java Remote Method Protocol" (JRMP) o el "Internet Inter-ORB Protocol" (IIOP) de CORBA. El programador de beans y aplicaciones sólo ve la clase del bean y su interfaz remota, los detalles de la comunicación de red están ocultos.

Ventajas y desventajas de EJB

Ventajas

- Recuperación de Errores. Al momento de definir los servicios y funcionalidades de un sistema es necesario contemplar la recuperación de errores tales como error al llevar a cabo un procedimiento, error al conectarse de la base de datos, etc. Mediante un contenedor de EJBS se ofrecen estos servicios y a través de un "Enterprise Java Bean" es posible desarrollar los componentes principales de la lógica del negocio.
- ·División del Trabajo. El programador se puede concentrar sus esfuerzos en la "lógica de proceso" sin preocuparse del diseño de servicios.
- ·Diversos vendedores. El uso de especificaciones para EJB's permite que existan diversos vendedores tanto de contenedores de EJB así como "Enterprise Java Bean's" los cuales resuelven algún tipo de lógica, permitiendo ejecutar cualquier EJB en cualquier contendor.

Desventajas

- Tiempo de Desarrollo. Aun cuando tiene varias facilidades esta tecnología se necesita planear cuidadosamente la lógica del negocio.
- Conocimiento Exhaustivo en Java. Se debe conocer RMI, JDBC y JNDI.

Descripción de XML

XML (eXtensible Markup Language o lenguaje extensible de marcado) es un estándar para representar datos, independientemente del sistema a utilizar, que, en resumen, proporciona un formato para describir datos estructurados.No es sólo un lenguaje, sino una forma de especificar lenguajes o "metalenguaje", de ahí parte su característica de extensible. XML representa un lenguaje para información auto-descrita, (en documentos bien escritos).

Actualmente XML es un estándar abierto del W3C[14] que agrupa una serie de tecnologías:

- *XML*. Lenguaje que define la sintaxis del XML que nos ayudará a crear nuevos lenguajes de etiquetas.
- *XLink*. Define la forma estándar de añadir enlaces dentro de un documento XML.
- *XPointer y XFragments*. Que define como poder hacer referencias a partes dentro del documento XML. Es como las URL, pero haciendo referencia a partes dentro del documento XML.
- *XSL (eXtensible StyleSheet Language)*. Define el estándar para las hojas de estilo de XML. Es la ampliación y modificación de las CSS. XSL está basado en XSLT.
- *XSLT (XSL Transformations)*. Es un lenguaje de transformación que se usa para ordenar, añadir y eliminar etiquetas y atributos.
- *XML Schemas*. Ayuda a los desarrolladores a definir estructuras precisas basadas en XML

Las características de XML son:

- Muestra el significado y las relaciones de la información contenida (en documentos y bases de datos), lo que permite gestionar y manejar datos, tanto estructurados como no estructurados.
- Facilidad de lectura, es decir, las aplicaciones pueden analizar y procesar documentos XML, y los humanos también pueden comprenderlos en caso de que haya un error en el procesamiento.
- No tienen instrucciones de formateo lo que se traduce en que los mismos datos pueden publicarse en diferentes medios.
- Es capaz de relaciones multiempresa, dado que permite simplificar las transacciones comerciales sobre la web y facilitar las relaciones entre clientes y proveedores; en realidad, fue diseñado para esto.
- Permite interactuar con otras aplicaciones, incluso automáticamente.
- Es extensible y por lo tanto adaptable tanto a necesidades futuras, en términos generales como a necesidades específicas de una empresa o de una relación comercial en concreto.
- Utilización independiente del mecanismo de acceso a datos, transacciones e interacciones.
- Permite que sean personalizables a cada usuario

[14] World Wide Web Consortium (W3C).Agrupación de empresas, univiersidades, institutos y personas significativas en el ambito web; NetScape, Microsoft, Oracle, SUN, IBM,entre otras

Entre sus ventajas, podemos listar que XML:

- Es un lenguaje independiente de la plataforma sobre la que se trabaje.
- Es UNICODE, lo que hace que pueda ser utilizado en múltiples lenguajes.
- Hace Independiente a la información con respecto a la representación, los datos solo dependen de los datos en si, no de su formato.
- Cuando se incluye un cambio en el documento, no supone un problema para su interpretación, ya que siempre hay que que leer el DTD.
- Sigue un estándar.
- Para acceder a la información, puede utilizar una representación a *alto nivel* y no *a bajo nivel* como con el HTML.

XML define dos tipos de documentos: **válidos y bien formados**. Éste es uno de los aspectos más importantes de este lenguaje, lo que hace necesario entender la diferencia.
Se dice que un documento XML está ***bien formado*** (well-formed) si cada etiqueta abierta tiene su correspondiente etiqueta cerrada y no tiene etiquetas anidadas fuera de orden, es decir que, es sintácticamente correcto respecto a la especificación, sin estar sujetos a unos elementos fijados en un DTD (descrito más adelante)..
Un documento XML es *válido* (valid) si su estructura se apega a alguna DTD. Todo documento que es válido necesariamente está bien formado pero no ocurre en caso contrario.

Con la extensibilidad de XML es posible crear etiquetas necesarias para un tipo de documento en particular. Un esquema de lenguaje describe la estructura de un conjunto de documentos del mismo tipo y puede usarse para limitar los contenidos. El esquema más utilizado es el **Document Type Definition (DTD)**. Desafortunadamente las DTD se definen con una sintaxis diferente a XML y sólo soporta 10 tipos de datos. Sin embargo, la evolución ha dado origen a esquemas XML, las cuales son otra manera de especificar la estructura de estos documentos.

Definición del tipo de Documento (DTD)

Los **DTD** (Document Type Definition o definición de tipo de documento) son aquellos queespecifican los elementos, atributos y entidades que pueden aparecer dentro del documento XML, así como las reglas y limitaciones a las que deben de estar sujetos dichos elementos, atributos y entidades. Cuando se procesa cualquier infomación formateada mediante XML, lo primero es comprobar si está bien formada(parsers no validadores), y luego, si incluye o referencia a un DTD, para verificar si sigue sus reglas gramaticales (parsers no validadores)..

El DTD puede estar definido de dos formas:
Dentro del documento XML. En este caso el DTD irá dentro del prologo del documento:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE nombreDTD [--definición del DTD--]>
```

En un documento externo. Al ser un documento externo, puede ser compartido:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE nombreDTD SYSTEM "fichero.dtd">
```

Podemos tener una mezcla de los dos. De tal manera que tengamos parte de la declaración en el propio documento y el resto en un DTD externo.

XML en EJB

En el caso de EJB, XML se utiliza para dar una descripción de los beans, el tipo de transacción que se manejará por el contenedor, la seguridad, aprovechando la característica de XML de ser una forma fácil de manejar un mensaje altamente escturcturado en un documento con un grado de estructuración relativamente bajo. Estos documentos son conocidos como descriptores de despliegue (deployment descriptors).

La principal ventaja obtenida de su uso es el cambio transparente al momento de implementar el código. Si se cambia el código de los EJB's, la manera en que el contenedor los administra no cambia ya que los metadatos obtenidos del XML tan sólo son un esqueleto de las características de los beans.

La evolución que esta teniendo XML según varios expertos tales como Adam Bosworth, Arquitecto Lider de Desarrollo Avanzado de BEA(IBM), nos orientan a una arquitectura basada en mensajes, teniendo aplicaciones de diferentes empresas conectadas asíncronamente las cuales proporcionen servicios. En esta visión los archivos XML serán convertidos en objetos metadatos tratádos por medio de un lenguaje como recipientes de información a manera de estructuras de datos.

LENGUAJE DE MODELADO UNIFICADO(UML)

Dado que hay múltiples formas de representar un sistema es necesario establecer una notación que se utilice en todas las etapas de desarrollo y que sea conocida por todos, teniendo por opción el **Lenguaje de Modelado Unificado (UML)** que se define a como un *"lenguaje que permite especificar, visualizar y construir los artefactos de los sistemas de software"*.

La notación UML se deriva y unifica las tres metodologías de análisis y diseño Orientado a Objetos(OO) más extendidas:

- Metodología de *Grady Booch* para la descripción de conjuntos de objetos y sus relaciones.
- Técnica de modelado orientada a objetos de *James Rumbaugh* (OMT: Object-Modeling Technique).
- Aproximación de *Ivar Jacobson* (OOSE: Object- Oriented Software Engineering)

Al utilizar UML se pretende:

- Proporcionar a los usuarios un lenguaje de modelado visual expresivo y utilizable para el desarrollo e intercambio de modelos significativos.
- Proporcionar mecanismos de extensión y especialización.
- Ser independiente del proceso de desarrollo y de los lenguajes de programación.
- Proporcionar una base formal para entender el lenguaje de modelado.
- Fomentar el crecimiento del mercado de las herramientas OO.

Es importante notar que UML es un lenguaje para construir modelos y no una metodología por si mismo, ya que no guía al desarrollador a través del análisis y diseño ni le indica a qué proceso de desarrollo apegarse. Su utilidad radica en la necesidad de contar con una notación estándar a las personas involucradas con el desarrollo orientado a objetos.

UML se basa en artefactos, los cuales son pedazos de información que son producidos y/o utilizados en el desarrollo del software. Existen dos tipos de artefactos:

- Artefactos dinámicos: describen el comportamiento del sistema
- Artefactos estáticos: describen las características del sistema.

Dinámicos	**Estáticos**
Comportamiento del sistema.	*Estructura del sistema.*
Casos de uso y diagramas de casos de uso. Diagramas de interacción y contratos Diagramas de estados. Diagramas de actividad.	Diagrama de clases. Diagrama de componentes. Diagrama de distribución

Diagramas Dinámicos

Diagrama Casos de uso.

Se identifica a los **actores** involucrados con el sistema que son entidades, cosas o personas que representan un rol dentro del sistema y que tienen interacción con el mismo. En este diagrama se explica el conjunto actividades de un sistema, subsistema o función en particular, así como los actores y la relación entre casos de uso y actores por medio de líneas guiadas. Su objetivo es ofrecer una especie de esquema conceptual para conocer a los actores del sistema y la forma en que lo utilizan. Sus elementos principales son: los actores (figura abstracta) los casos de uso (óvalo) y las relaciones (líneas guiadas)

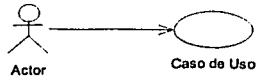


Ilustración 3: Casos de uso

Los elementos principales de este artefacto son:

Actor: No son parte del sistema, sino que representan roles que un usuario puede jugar por lo que un actor puede ser un humano, una máquina u otro sistema.. Intercambian información con el sistema, aún cuando puede ser tan sólo un recipiente pasivo de información.
Caso de Uso: Es una función o flujo de eventos completo con significado para el usuario. Modela el diálogo entre los actores y el sistema. Un caso de uso es iniciado por un actor, dando cierta funcionalidad en el sistema. Al unir todos los casos, se tienen todas las formas de uso del sistema.
Relación: Manera en que interactúan los actores y los casos de uso. y pueden estar marcadas con estereotipos que describan el tipo de relación de que se trata.

Los estereotipos son mecanismos para extender el significado de los elementos de UML, por lo que se emplean en prácticamente todos los artefactos generables. En los casos de uso se utilizan para especificar con mayor claridad la relación entre los casos de uso. La intención de los estereotipos es no solo utilizar los existentes sino extenderlos a nuestras necesidades particulares haciendo más comprensible la documentación. Normalemente se marcan con "<< >>" (<<extends>>, <<includes>>).
Ente los estereotipos principales se encuentran:
Extends. Se extiende un caso de uso y se le agregan pasos o actividades adicionales, aún cuando el caso de uso base esta completo .
Includes. El caso de uso necesita pasar por otros casos de uso para su realización, es decir, su ejecución incluye a otros casos de uso.

Los diagramas de casos de uso cuentan, además, con una documentación textual que apoya a su comprensión, y que incluye el nombre de la función a que se refiere el caso de uso, una breve descripción de la misma, las condiciones que se tienen que dar para que se lleve a cabo (precondiciones), el flujo principal de acciones, los flujos alternos u opcionales que pudieran existir, y las condiciones posteriores a la aplicación de la función (postcondiciones).

Diagramas de Interacción.

Ayudan a definir cómo se comunican e interactúan las partes involucradas en el sistema. El empleo de los diagramas de interacción define a las funciones del sistema en función a métodos y las clases que los realizan. Además muestra la forma en la que las clases se comunican entre sí y los mensajes que se envían. De igual forma que para los diagramas de casos de uso, existe una documentación particular para estas funciones que ayuda a la lectura de los diagramas, denominados *Contratos*.

Un contrato incluye el nombre del método, su responsabilidad, el tipo al que pertenece la función, el método o función asociado, notas (de ser necesarias), las excepciones o errores previstos, la salida de la función, precondiciones y postcondiciones a la ejecución del método.

Dentro de los diagramas que muestran la interacción pueden ser diagramas de secuencia o de colaboración, dependiendo de si se centran en la consecución de los pasos de una operación en particular(secuencia) o en la forma en que las clases trabajan en conjunto (colaboración).

Diagramas de Secuencia

Artefacto de UML que se utiliza para mostrar las interacciones entre los objetos, enfocándose en el orden o secuencia de pasos que integran a una función.
Los elementos de este artefacto son:

Clases/Objeto/Actor. Define las clases alineadas en la parte superior del diagrama de izquierda a derecha en el orden en que van formando parte de la función. Los métodos se indican sobre las interacciones organizadas de arriba debajo en la misma secuencia en la que van siendo utilizados. Las interacciones van desde la clase o actor que solicita el método hasta la clase que lo posee. Es posible indicar llamadas reflexivas para indicar llamadas a métodos propios de la clase que lo solicita.
Linea de Vida. Línea punteada vertical que sale de la parte inferior de la clase indicando el tiempo que las instancias de la clase existen y se puede trabajar con ellas.
Mensajes. Medio de transporte de información desde un objeto a otro con la expectativa de que realice una actividad. Se implanta como la invocación a un método y/o el envío de un mensaje entre procesos o hilos de ejecución diferentes.
Foco de Control. Indica el tiempo en que la clase tiene el control de la función.

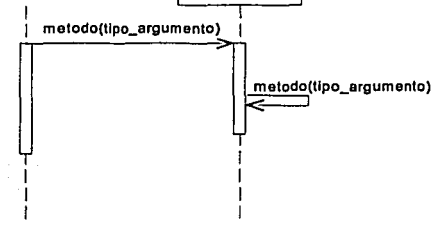


Ilustración 4: Diagrama de secuencia.

Diagrama de Colaboración.

Diagrama que, de forma similar al diagrama de secuencia, muestra la interacción entre objetos pero, a diferencia del anterior, se enfoca en la parte estructural de la interacción o la relación entre las clases. La organización de los elementos sobre el área del diagrama es de forma libre, ya que no muestra la secuencia de acciones sino la colaboración que existe entre clases y objetos. El manejo de las interacciones es por mensajes (flechas) situados sobre las ligas (lineas no guiadas), que representa la interacción entre dos objetos. Puede haber tantos mensajes como llamadas a métodos hacen y reciben las clases y son numerados en el orden en que se van dando. Otra característica importante de este diagrama es la posibilidad de representan múltiples instancias de una clase (multiobjetos), cuando el diseño lo dicta, mediante el símbolo de conjunto de objetos de la clase (rectángulos sobrepuestos).
Sus elementos son:

Objeto/Clase/Actor. Utiliza la misma notación que los diagramas de secuencia pero estos pueden ser ubicados libremente.
Liga. Solo se coloca una liga entre dos clases u objetos representando que ellos se comunican con una línea no guiada que los conecta. En este caso es posible representar tambien la interacción entre varios métodos o elementos de una misma clase u objeto, lo que se conoce como retrollamada y se grafica como una liga cuyo origen y destino estan en la misma clase.
Mensaje. No hay limite respecto al número de mensajes a colocar, cada uno representaun una invocación a un método de la clase hacia la que se dirige el mensaje.
Multiobjeto. Representación de múltiples instancias de una clase (objetos), agrupados por la clase a la que instancian, permite mostrar la relación de una clase con multiples objetos de un mismo tipo.

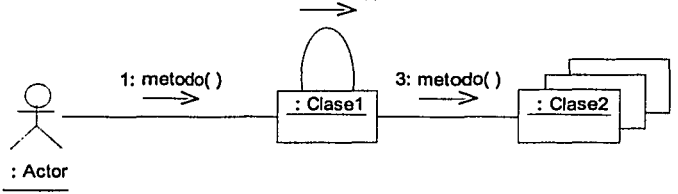


Ilustración 5: Diagrama de colaboración.

Diagramas de Estados

Indica cómo una clase o componente va adquiriendo determinadas etapas o fases a lo largo de su ciclo de vida, y qué condiciones llevan a la transición de uno a otro. Muestra el ciclo de vida del objeto, los eventos o condiciones que causan la transición de un estado a otro y las acciones resultantes de los cambios de estado. Este diagrama, al estar enfocado a un solo concepto (máquina de estados finitos), nos permite el estudio detallado del mismo y su funcionalidad como objeto, se puede definir como el diagrama del autómata que un objeto realiza.

Sus principales elementos son:
Estados: Posible condición en la que un objeto puede encontrarse dentro de su ciclo de vida, representado por un rectángulo con las esquinas redondeadas que incluye el nombre del estado y, de forma opcional, las acciones que se realizan al entrar al estado (entry), mientras se permanece en él (do) y a la salida del estado (exit). Además, se definen dos estados especiales, presentes en el diagrama de estados:

- *Estado inicial.* Estado de entrada cuando un objeto es creado, indica el inicio del ciclo o autómata. Es obligatorio contar con uno y solo uno. Se representa por un círculo sólido.
- *Estado Final* Indica el final del ciclo o autómata. Es opcional y puede existir más de uno. Se representa por medio de dos círculos concéntricos.

Transición de estado. Cambio del estado original a un estado sucesor como respuesta a un evento o al cumplimiento de una condición, se representa con una línea en la que es posible indicar la condición necesaria para el cambio de estado o el evento que lo origina. Además, la realización de una transición puede incluir acciones, aparte del cambio de estados.
Eventos. Es una ocurrencia que sucede en un instante de tiempo dado, y que deriva en una transición de estados.
Condición de guarda. Es una expresión Booleana sobre el valor de los atributos de un objeto que se debe cumplir para que se dé la transición de estado. Se representa como una expresión entre corchetes.
Acción. Es una actividad que se debe dar ya sea al cambiar de estados o mientras nos encontramos en alguno en particular, por lo que estas operaciones que puede estar relacionadas con una transición o con un estado. En todo caso se representan indicando el momento en el que se lleva a cabo la acción (entry, do, exit) y pueden combinarse con las condiciones de guarda para especificar situaciones especiales que dan paso a una acción.

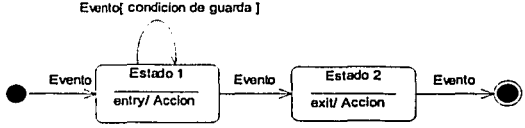


Ilustración 6: Diagrama de Colaboración.

Diagramas de Actividad

Modela el flujo de trabajo de un proceso de negocio así como la secuencia de actividades que incluye. Su enfoque es hacia las funciones y productos generados (objetos, reportes, etc) a lo largo del desarrollo de una función o conjunto de funciones del sistema. Tiene gran parecido con los diagramas de flujo, ya que representa la secuencia de pasos que definen a una actividad de principio a fin, pero incluye nuevos elementos que aumetan su capacidad de representación.

Sus principales elementos son:
Estados y Actividades. Una actividad representa el desarrollo de una tarea o paso dentro del flujo de trabajo. Dentro de los estados se identifican:

- *Estado Inicial.* Muestra el principio de un flujo de trabajo. Es representado por un circulo relleno.

- *Estado Final.* Muestra el término de un flujo de trabajo. Se representa por dos círculos concéntricos.

Transición de Estados. Representa que un objeto en el estado origen desarrollará ciertas acciones específicas y entrará en el estado destino cuando ocurra un evento o cuando se cumplan ciertas condiciones. Una transición es una relación entre dos actividades, dos estados, una actividad y un estado o una actividad y una sincronización, y se representa por una línea guiada.

Sincronizaciones. Indica que dos actividades deben unirse o bifurcarse para continuar, y se representan como barras horizontales o verticales a las que van unidas las transiciones. Definen visualmente flujos de trabajo alternos.

Decisiones. Representa un lugar específico dentro del flujo de trabajo donde este puede ramificarse dependiendo de una condición de guarda. Se dibujan como rombos de los cuales pueden salir, a diferencia de los diagramas de flujo tradicionales (si/no), más de dos transiciones, dependientes del cumplimiento de una expresión booleana (llamada condición de guarda).

Carriles. Divisiones verticales nombradas en su parte superior en donde se pueden agrupar las actividades de acuerdo a quien las realiza, representando cada carril una unidad organizacional o rol dentro de un modelo de negocio. Esta dicvisión es muy útil para determinar las responsabilidades de cada tarea dentro de la actividad.

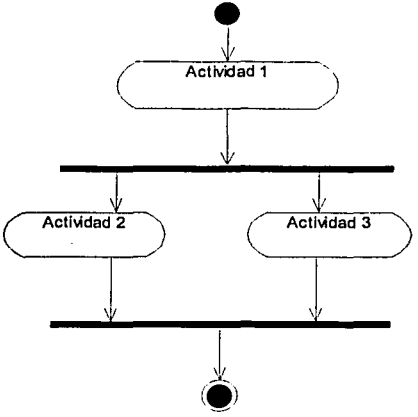


Ilustración 7: Diagrama de Actividad.

Diagramas Estáticos

Diagrama de clases.

Es el artefacto principal en el desarrollo orientado a objetos. Muestra las clases en las que se implementará el sistema, sus relaciones atributos y operaciones.
Este artefacto pasa por varias etapas a lo largo del proceso de desarrollo. La primera concepción de este diagrama se refiere al *modelo conceptual* del sistema, en donde se identifican los conceptos involucrados con el sistema y que son candidatos a clases, en esta etapa solo es necesario conocer a los conceptos por lo que se grafican las clases con relaciones simples, dando por resultado un panorama inicial de la estructura del sistema. Posteriormente se determina cuales clases se implementarán en el sistema, construyendo, a partir del modelo conceptual, un *diagrama de clases de diseño* en el que se hacen más específicas las relaciones y se agregan métodos y atributos a las clases, lo que da por resultado un esquema completo de la arquitectura de clases del sistema. Independientemente de lo anterior, este diagrama es el más recurrido por los desarrolladores orientados a objetos, ya que es el estandar más utilizado de representación de clases y sus relaciones.

Sus principales elementos son:
Clase. Descripción de un conjunto de objetos que comparten los mismos atributos, operaciones y relaciones, algunas surgen de los conceptos identificados durante el análisis y otros más, de las interfaces. Su construcción incluye elementos separados en compartimientos del rectángulo que representa a la clase y que organiza por separado el nombre de la clase, sus atributos y sus métodos.

- *Atributos.* Característica o dato de una clase a la cual se le da un nombre. La visibilidad (control de acceso).
- *Métodos.* Servicio que se le puede solicitar a un objeto para un comportamiento determinado. Se puede mostrar opcionalmente la firma completa del método (nombre completo de la operación), que incluye el tipo de dato que regresa, el nombre del método y los parámetros que requiere. De igual forma que para los atributos, la visibilidad se grafica con un ícono que antecede al método.

El encapsulamiento, o visibilidad que tienen las clases hacia los atributos y operaciones de una clase con la cual están relacionadas se grafica con un ícono que antecede al atributo.Existen tres tipos:

Público. Son visibles a cualquier clase que tengan una relación con la clase que las contiene. Se modelan con un signo de más ("+"). Corresponde a *public* en Java.
Privada. Sólo son visibles internamente en la clase que los contiene. Se modelan con un signo de menos ("-"). Corresponde a *private* en Java.
Protegidas. Solo las clases hijas pueden ver los atributos. Se modelan con un símbolo de número ("#").Corresponde tanta a *protected* como a *package* en Java.

Relación. Define la manera en que las clases interactúan. En este punto también se puede incluir estereotipos definidos por nosotros, siguiendo misma sintaxis "<<nombre_estereotipo>>". Un estereotipo define a la relación y facilita la lectura de la misma. Hay varios tipos predefinidos por UML entre los que se encuentran la ***asociación***, la ***agregación***, la ***composición***, la ***herencia*** y la ***dependencia***.

Tipos de relaciones.

Asociación: Dos clases pueden verse o solicitarse servicios.

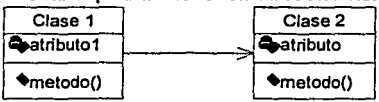


Ilustración 8 : Asociación.

Agregación Una clase es parte de otra clase que la contiene.

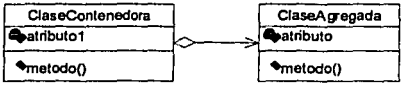


Ilustración 9: Agregación.

Composición: Una clase esta compuesta por otra que no puede existir sin la primera.

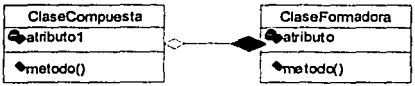


Ilustración 10: Composición.

Herencia: Una clase hereda o es un tipo especial de otra clase ya sea que se tomen características comunes a varias clases para crear una superclase de la cual heredan (*generalización*) o que se cree una que aumente o modifique las características de la que hereda (*especialización*).

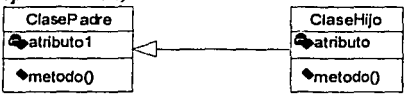


Ilustración 11: Herencia.

Dependencia: Relación poco duradera en donde la comunicación solo es posible en momentos específicos.

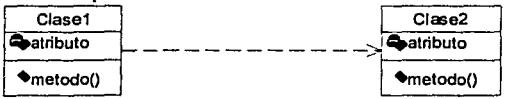


Ilustración 12: Dependencia.

Navegabilidad y cardinalidad

Además, se indica la navegabilidad (punta de la flecha, si no la hay se dice que es bidireccional) de las relaciones, es decir desde donde y hacia donde viaja la relación, así como la cardinalidad, que es la indicación de el número de clases que intervienen de cada lado de la relación, mediante un números determinados, una "n"o asterisco para indicar muchos o una, o rangos en los que puede caer ese valor.

Navegabilidad

La navegalilidad indica que una clase tiene acceso a los miembros de otra, en el sentido que la flecha marca. Así, se define el flujo de mensajes e invocaciones existente entre dos clases reacionadas:

Ilustración 13: Navegabilidad.

Cardinalidad

La cardinalidad indica la cantidad de instancias de cada clase involucradas en una relación. Puede expresarse en términos de rangos (de uno a tres 1...3; de cero a muchos 0...*):

Ilustración 14: Cardinalidad.

En términos absolutos (uno "1"; muchos " * "):

Ilustración 15:Cardinalidad.

O en términos de conjuntos de valores posibles (1, 2 ó 3 "[1,2,3]", 0 ó 1 [0,1]):

Ilustración 16: Cardinalidad.

Diagrama de Componentes.

Las clases generadas se deben organizar y estructurar en paquetes lógicos, por lo que es necesario crear un diagrama que ejemplifique la forma en la que esos paquetes o componentes existen y se relacionan para dotar de funcionalidad al sistema.

Los paquetes son los grupos de clases con características comunes o que tienen relaciones muy sólidas por lo que es más lógico tenerlas juntas. Por lo anterior es necesario definir la organización más conveniente de las clases y así crear los componentes que puedan ser reutilizados de ser necesario. Una vez identificados se establecen las relaciones existentes entre todos los paquetes del sistema (líneas simples) con lo que se tiene un panorama de la estructura de organización de sistema a través de los módulos o componentes que lo integran.

Los elementos de este artefacto son:
Componentes. Parte lógica o física reemplazable de un sistema de software que tiene una interfaz que proporciona acceso a sus servicios.
Interfaces. Agrupamiento nombrado de operaciones que caracteriza el comportamiento de un elemento. Es el punto del componente que es exhibido para acceder al mismo.
Clases: Las clases que se encuentren en un componente deben ofrecer servicios muy relacionados entre si, por lo que un componente debe ofrecer pocos servicios y muy especializados hacia el exterior.

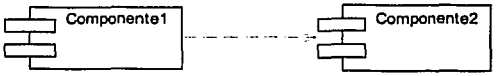


Ilustración 17: Componentes.

Diagrama de Distribución.

Artefacto donde se muestra la forma en que las aplicaciones se distribuyen dentro de los sistemas que las contienen y la forma en que se accede a ellas, esto es muy útil sobre todo hablando de sistemas distribuidos, como el del presente proyecto. Los equipos o dispositivos son representados como cubos, que en su interior contienen componentes (del diagrama de componentes y que están interrelacionados por medio de ligas. La revisión de este diagrama nos permite ver la forma en que los componentes se distribuyen en determinados dispositivos, tales como bases de datos, sistemas cliente o sistemas servidores, y por lo tanto complementa a todos los demás diagramas con la visión más general del sistema. Sus elementos son:
Nodo. Equipo, máquina, herramienta o componente de hardware que forma parte del sistema.También puede identificar a varios componentes en un sistema distribuido o cliente-servidor.
Liga. Muestra la conexión entre dos dispositivos o componentes.

Ilustración 18: Diagrama de distribución.

Podemos definir a un juego, en términos generales, como una actividad o competencia gobernada por un conjunto de reglas, en la que se participa con un fin lúdico, así como para el desarrollo de capacidades físicas y/o mentales.Un juego es, en resumen, una forma de representación con objetivos y estructura.[15] Así, pueden identificarse tres puntos fundamentales dentro de esta definición y que no sólo atañen a los juegos tradicionales, sino también a los videojuegos.

Estructura (reglas).

Cada juego cuenta con una serie de reglas que determinan las acciones que puede realizar un jugador, así como las consecuencias de dichas acciones. Existen juegos que permiten una mayor libertad en cuanto a la aplicación de reglas que otros. Tomemos por ejemplo el caso del ajedrez o las damas chinas. Para cada pieza existe un tipo de movimiento, y bajo ninguna circunstancia un jugador puede pasar por alto estas reglas al momento de realizar un movimiento. En el caso de los juegos de rol[16] (aquellos donde el jugador asume un papel que debe representar), las reglas y su forma de aplicarse dependen de una entidad reguladora (Maestro del Juego o Story Teller), del contexto y en algunos casos el azar.

El conjunto de reglas conforma la estructura del juego, misma que define el contexto en el que el juego se desarrolla, y que en todo momento deben ser seguidas por los jugadores.

Objetivo.

Todas las acciones que los jugadores realizan están encaminadas a la consecución de un objetivo, el cual, también se define como parte de la estructura del juego. En la mayoría de los casos, este objetivo es el de obtener más puntos o ganar la partida. Sin embargo no en todos los casos la consecución de los objetivos deben producir necesariamente ganadores y perdedores.

Representación.

Este aspecto nos indica que los jugadores son participantes activos, es decir, crean el juego con sus acciones. De la misma forma, todo lo que acontece en un juego sólo es parte de este, es decir, el jugador *pretende* y todas sus acciones rara vez se traducen en consecuencias trascendentales dentro de lo que puede considerarse como el mundo real.

Algo que todos los juegos tienen en común y que la gente frecuentemente olvida es que los juegos son divertidos, independientemente de que puedan ser utilizados como simulaciones o como técnicas de aprendizaje. En muchas ocasiones a los juegos se ven como algo contrario al trabajo y su utilización por parte de las personas es independiente de su edad. En cuanto a este respecto se puede decir que conforme la persona crece y madura, debe buscar nuevas formas de entretenimiento (o nuevos juegos) que se adecuen más a su cambiante forma de pensar y donde puedan sentir que sus habilidades pueden ser más desarrolladas. Los gustos, aptitudes y habilidades de las personas, así como su definición de lo que es "divertido" determinarán el tipo de juegos que jugarán.

---

[15] Maroney Kevin; My entire waking life.

[16] El más representativo de estos juegos es Calabozos y Dragones.

Videojuegos.

Hasta este punto no se ha mencionado nada sobre los videojuegos, sin embargo, los videojuegos son sólo una extensión de los juegos tradicionales, sólo que éstos utilizan a la computadora como medio por el cual se proporciona al jugador todos los elementos (estructura, objetivos y representación) para que el juego se desarrolle. Los videojuegos pueden, por este medio, representar cualquier situación imaginable para permitir que el jugador tenga una experiencia lúdica.

Desarrollo e Industria de los Videojuegos, un panorama general.

En sus inicios, el desarrollo de videojuegos era una tarea hasta cierto punto "sencilla", ya que una sola persona podía encargarse de la programación de todo un juego, el cual en muchas ocasiones se hacía con propósitos técnicos o personales. En estos casos, el público al que iba dirigido este juego era el propio programador y en algunos casos un grupo selecto de personas que compartían intereses con el programador. Conforme se popularizó el uso de los videojuegos gracias al advenimiento de las consolas caseras y la PC, se dio el génesis de la industria del videojuego. Así, el número de personas involucradas en el desarrollo de los juegos aumentó de forma considerable, y de igual forma las estructuras utilizadas para su distribución.

Debido al refinamiento de la tecnología y el dinero invertido tanto en herramientas de desarrollo como en publicidad, la creación de videojuegos pasó de ser un proyecto individual a proyectos interdisciplinarios que requieren de la coordinación de una gran cantidad de gente que tiene el objetivo de crear un software de entretenimiento, que en muchos casos llega a costar mucho más que algunos sistemas empresariales.

Dentro del desarrollo de un videojuego se requiere de gente especializada en cada una de las distintas áreas que se involucran dentro del juego:

Diseño.
Programación.
Gráficos.
Música.
Escenas de video.
Inteligencia Artificial.
Modelos Físicos.
Pruebas, etc.

Pero ¿Se puede no ser rentable en una industria que tan sólo en los Estados Unidos genera más de 20 mil millones de dólares anuales[17]? La respuesta es sí.

Como puede verse, el desarrollo de videojuegos no es una tarea sencilla y en muchas ocasiones tampoco redituable, a menos de que se cuente con una idea realmente innovadora[18] o se tengan los derechos sobre franquicias cuyo poder de venta esté asegurado[19]. Y aun así, la industria de los videojuegos se rige por las leyes de la oferta y la demanda, por lo que, en ocasiones, los desarrolladores, en lugar de ofrecer nuevas ideas, que puedan ser riesgosas, se apegan a formulas probadas.

---

[17] Hasta marzo de 2003, de acuerdo a estudios realizados por la IDSA (www.idsa.org).
[18] The Sims © Electrinic Arts 2.
[19] Franquicias como Mario Bros, Tomb Raider, Final Fantasy, sólo por citar algunos.

Ciclo de Vida de un Videojuego.

El ciclo de vida de un videojuego es muy similar al ciclo de vida tradicional de un sistema: Definición de Requerimientos, Análisis, Diseño, Programación, Implementación, Pruebas y Mantenimiento, esto en parte porque un videojuego es un sistema informático con fines lúdicos.

Fase Conceptual.

Aquí es donde surge la idea que da vida a un juego. Esta idea debe ser discutida con una serie de personas que puedan retroalimentarla, de forma que se encuentren todas aquellas posibilidades de su implementación. Una recomendación para los diseñadores y analistas es que una vez que tengan una idea y la hayan desarrollado, la dejen descansar, para posteriormente retomarla y nuevamente revisarla, de forma que puedan constatar que realmente se trata de una idea factible.

Fase de Diseño.

Durante esta etapa se genera lo que se conoce como n "Documento de Diseño" que no es más que poner sobre papel todo aquello que va a hacer el videojuego, especificando la plataforma a la que se dirige, género, tipo de gráficos, descripción de niveles, descripción de cómo deben comportarse cada una de las partes involucradas, etc. De la misma forma deben contarse con sketches del arte, storyboards que permitan conocer el flujo del juego y una descripción de cada uno de los movimientos permitidos.

Este documento, junto con una Biblia Gráfica (especificación de cada uno de los componentes gráficos involucrados) son los documentos sobre los que debe basarse todo el desarrollo. Mientras más detallado sea este documento mejor, ya que dejará menos huecos a llenar y homogeneizará todo aquello relacionado con el juego, permitiendo un mejor desempeño del equipo de trabajo.

Así mismo deben establecerse todas aquellas consideraciones financieras y legales, ya que este documento es el que se presenta a las casas productoras para poder obtener un financiamiento. Para evitar futuros errores en cuanto al desarrollo, durante esta fase deben establecerse prioridades acerca de lo que se debe incluir, de forma que puedan atacarse estas primero y si queda tiempo añadir todos esos "detalles adicionales".

Fase de Producción.

Se refiere a la programación, creación del arte digital, música, herramientas y todo aquello relacionado con la creación del producto como se presentará. Durante esta fase se cuenta con fechas de entrega que deben ser respetadas en cualquier momento si se quiere ganar la confianza de la casa productora y tener mayor libertad en cuanto a las gestiones internas.

Cabe hacer notar que una buena comunicación entre las partes involucradas, así como un excelente documento de diseño dan como resultado un mejor flujo del trabajo, así como la detección de problemas potenciales, y su rápida solución. Como sucede en cualquier proyecto de software, se debe contar con un grupo talentoso y verificar que se logren todos los objetivos en los tiempos planeados (en la medida de lo posible), ya que esto ayudará a lograr la confianza de aquella persona y/o compañía que han depositado no sólo sus confianza, sino su dinero en el producto que se está desarrollando.

Fase de Aseguramiento de Calidad.

Este es un proceso constante que se lleva de forma conjunta con la fase de producción. Cada uno de los productos que se obtienen en las fechas de entrega es probado arduamente por grupos de gente con la finalidad de encontrar errores potenciales que hagan del juego un mal producto. Con la información recopilada por este equipo, se pueden encontrar los problemas de forma rápida y poder realizar las correcciones pertinentes antes de que el producto pueda comenzar a ser fabricado para su venta.

Existe un límite para probar el software y deben darse prioridad a aquellos errores que limiten la capacidad del jugador de divertirse, ya que aunque se desee eliminar todos los errores del sistema, llegará un momento en el que este proceso será más costoso y contraproducente que benéfico; desafortunadamente no existe software libre de bugs.

Postmortems.

Este término es utilizado por el sitio gamasutra.com y la Game Developers Magazine como una serie de artículos en los que se analiza el trabajo realizado durante el desarrollo de un juego; así se puede conocer tanto lo que salió bien, como los problemas que se presentan a lo largo del desarrollo de su producto. Esto se hace con la intención de dar a conocer lo que sucede de forma interna durante el desarrollo de un juego, permitiendo a los desarrolladores aprender de las experiencias ajenas.

En este caso, se hace una recopilación de los factores positivos como negativos que se presentan durante estas fases:

Lo que salió bien.

- Capacidades del Equipo. Siempre se cuenta con gente conocedora que gracias a sus habilidades y comunicación, permiten que el proyecto cumpla con todos sus objetivos dentro de los tiempos planeados.
- La existencia de un documento de diseño consistente permiten al diseñador concentrarse en otras tareas en lugar de resolver dudas que ya están documentadas en el documento de diseño, ofreciendo una guia para todas las personas involucradas en el proyecto.

Lo que salió mal.

- Siempre existen problemas para lograr alguna fecha de entrega, ya que surgen imprevistos que exigen el desarrollo de nuevas herramientas o modificación de las existentes para poder darles solución, esto implica tiempo del equipo de desarrollo.
- Debido a la larga cadena burocrática existente y que el dinero proviene de muchas fuentes que exigen resultados, un proyecto terminado puede pasar meses en la bodega antes de que se le pueda dar luz verde para su fabricación. Por esta razón, es recomendable siempre quedar en buenos términos con aquellas personas encargadas de dar el visto bueno al producto y mantener abiertos los canales de comunicación para poder dar solución rápida a pequeños imprevistos administrativos.
- Sin importar que tan hábiles sean los miembros del equipo, siempre existe cierta aprehensión de éstos a su trabajo, sea un dibujo, un algoritmo o un fragmento de código. Cuando otra persona encuentra una mejor forma de hacer las cosas, se debe estar dispuesto a aceptar ese cambio, ya que es para lograr un mejor producto.
- Los cambios de último momento son igual de perjudiciales para el proyecto, ya que por mínimos que puedan parecer siempre tienen consecuencias imprevistas; antes de aplicar este tipo de cambios debe analizarse su impacto.

Consideraciones de tolerancia a fallos y reutilización.

Cualquier juego y cualquier software en general, requiere un tolerancia nula a fallos, es decir, que en ningún momento debe presentarse una falla que interrumpa el servicio que se ofrece a los jugadores, es decir jugar.

Desafortunadamente esto, como sucede con todo software, no es posible debido a que siempre existirán ciertas fallas (no criticas) que no se hayan refinado durante los procesos de revisión del software. Y que por diversos motivos no pueden ser corregidas más que con futuras iteraciones del producto. En el caso de los videojuegos, este es un factor que se agrava todavía más debido a que muchas veces el código realmente no es reutilizable (por más que se utilicen metodologías orientadas a objetos) y no se tienen iteraciones como sucede en el caso de software como un procesador de palabras.

Reutilización.

La programación de videojuegos así como la de cualquier otra aplicación consume una gran cantidad de tiempo, el cual puede influir de forma negativa en la cantidad de esfuerzo que debe ser aplicado. Sin embargo, gracias al advenimiento de la tecnología Orientada a Objetos (OO), la cual se ha publicitado mayormente con la reutilización que pueden darse a ciertos componentes del software, la industria del videojuego puede verse ayudada a minimizar los esfuerzos requeridos para realizar un juego.

Sin embargo ¿es realmente posible la reutilización dentro de este campo?

El principio de la reutilización del software se basa en la existencia de módulos pre-fabricados que puedan ser utilizados en la construcción de un sistema en la misma forma en que se construye una computadora. Se elige el más conveniente y simplemente lo integra a su sistema. Fácil ¿cierto?

La respuesta es no, es más complejo y para ejemplo basta un botón. Con el advenimiento de la tecnología OO muchas empresas realizaron grandes inversiones en este campo tomando como justificación económica el ahorro que existiría en cuanto a la reutilización del software. Sin embargo tiempo después esta reutilización ha sido casi nula y muy pocos componentes se reutilizan, los costos de los proyectos siguen por las nubes y los proyectos fracasan.

Ahora, antes de continuar se definirá un concepto importante: *Reutilización del software es software que ha sido diseñado para ser reutilizable.*[20] Partiendo de esto se puede visualizar la razón por la que muchos proyectos, que supuestamente se basan en la reutilización del software, fracasan. Tal es que, en vez de crear componentes que puedan ser reutilizables, crean componentes específicos para la aplicación y cuando surge otra similar, intentan ocupar los componentes previamente escritos, pero como estos no cumplen con las características necesarias para la reutilización, son descartados y fabricados desde cero.

Existe una buena razón para que las empresas (sobre todo las que desarrollan juegos) no puedan darse el lujo de diseñar componentes reutilizables: La fecha de entrega.

---

[20] Will Tracz's; Confessions of a Used Program Salesman: Institutionalizing Software Reuse.

Se ha demostrado que el desarrollo de un componente reutilizable implica de 3 a 5 veces más esfuerzo para desarrollar. Un claro ejemplo de esto es el Software Engineering Laboratory de la NASA, la cual ha obtenido un porcentaje del 75% en cuanto a la reutilización de componentes de software, con lo que se reduce a una décima parte la cantidad de esfuerzo para desarrollar una aplicación (dentro de un campo muy específico). Sin embargo esto no ha venido sin sus costos, ya que tomó más de 35,000 horas de análisis y 40,000 horas para el diseño de estos componentes[21].

Pero no todas las compañías pueden darse el lujo de dedicar tanto tiempo a la creación de componentes reutilizables ya que, como se mencionó, cuentan con una fecha de entrega. Por esta razón, lo más importante es entregar el producto, lo cual implicará un pago, de lo contrario el pago no existe y se mancilla la imagen de la compañía o el departamento, lujo que las compañías no pueden darse.

Sin embargo no todo está perdido, ya que se cuentan con compañías que se encargan de ofrecer algunos módulos generales que pueden ser utilizados indistintamente de la aplicación que se fabrique. Un ejemplo de esto es el lenguaje de programación Java, el cual incluye en su API una gran cantidad de componentes que pueden ser utilizados por los programadores, reduciendo así una gran cantidad de esfuerzo de programación.

A pesar de que puedan existir estas fábricas de objetos reutilizables es muy difícil que gracias a estos las compañías puedan reducir sus costos basados únicamente en la reutilización de componentes. A menos que se puedan dar el lujo de diseñar componentes específicamente con este propósito.

Un claro ejemplo de este problema es que en la mayor parte de las aplicaciones comerciales se cuenta con un objeto "Cliente". Entonces si se cuenta con un objeto de este tipo ¿por qué no hacer un componente reutilizable que pueda ser ocupado por cualquier aplicación? Como lo menciona el ejemplo de la NASA, los componentes reutilizables pertenecen a un campo de acción específico, cosa que no sucede con el Cliente del ejemplo, ya que su comportamiento es muy distinto si se encuentra comprando un boleto de avión , pagando facturas o realizando transacciones bancarias.

Juegos de Rol.

Comparando a los videojuegos con las películas, existe una variedad de éstos que se adecua a los gustos de los consumidores. Dentro de estos géneros (Deportes, Conducción, Aventura, Estrategia, Puzzles o Rompecabezas, First Person Shooters, etc), el que nos interesa para objetos de este trabajo es el de RPG:

Un RPG o Juego de Rol es un juego en el que una persona toma el papel de otra y dentro del juego se desarrolla como el personaje que ha seleccionado. Este es un concepto similar al de participar en una obra de teatro, sólo que con la diferencia de que el fin de esta "obra" no está definido.

Como se explicó anteriormente, dentro de un RPG las reglas del juego, así como los objetivos están controlados por una entidad que es la que funge como director y al mismo tiempo escritor de la historia. Así, los RPG en su forma original

[21] Adolph Steven; Whatever Happened to Reuse?

(no como videojuegos), ofrecen una serie de posibilidades infinitas cada vez que es jugado, cosa que desafortunadamente no sucede dentro de los videojuegos.

En lo que atañe a los videojuegos, un RPG sigue las reglas básicas de un RPG tradicional, es decir, el jugador toma el rol de un personaje y lo desarrolla a través de una historia, sólo que en este caso, la historia es finita y repetible, ya que los desarrolladores han programado una historia sobre la que el jugador debe desarrollar el personaje y por lo tanto no se puede cambiar. Una vez terminado el juego, el jugador puede volver a comenzar la historia, pero, para llegar nuevamente al final, tendrá que volver a pasar todas aquellas situaciones que ya ha vivido, dejando muy poco espacio para la novedad dentro del juego. Por esta razón, las secuelas de este tipo de juegos son tan populares.

Sin embargo, existen paradigmas dentro de la programación de los videojuegos y sobre todo los RPG que están enfocadas a ofrecer al jugador una experiencia totalmente distinta cada vez que el juego es jugado. Uno de estos paradigmas es el de las directivas procedurales.

Directivas Procedurales.

Como se ha explicado, uno de los más grandes retos para el desarrollo de casi cualquier tipo de juego, es encontrar la forma de generar de forma espontánea, creativa, pero a la vez coherente, todo lo que integra al juego. Esto es, que se generen, en tiempo real, las situaciones que conformen la historia, asegurando su carácter único y, a la vez, reaccionando a todos los eventos que se generen dentro del juego (humanos y artificiales).

Se define una entidad (el sistema servidor) que se encarga de integrar las aventuras afectando a los objetos y personajes y permitiendo que ellos afecten el ambiente, administrando recursos y situaciones, observando que se apliquen las reglas y lineamientos y, sobre todo, generando de forma espontánea, eventos que conformen la historia de acuerdo a factores desarrollados minuto a minuto.

En este sentido el sistema vuelve a tomar un rol decisivo dentro del juego, y lo comparte con los jugadores. No todo puede depender de ellos ya que no habría propuestas por parte del sistema y sería extremadamente complejo incluir otros tipos de factores. La idea es hacer un juego con la menor cantidad de límites en cuanto a creatividad. El sistema fungirá como un cronista que recabe la historia que se esta generando en ese momento pero, a la vez, manejará los elementos que son ajenos a los jugadores así como las relaciones entre ellos y los mencionados elementos.

Dichas facultades encuentran respuesta en la simulación procedural. La inteligencia artificial (IA) da las herramientas con las cuales administrar una serie de entradas y salidas, acciones y reacciones, con cierto grado de predeterminación. Las directivas procedurales, basadas en sistemas expertos, y a su vez, compuestas de IA, hacen posible la espontaneidad. Es como crear un super sistema experto que utilice a sus subsistemas especializados para tomar decisiones congruentes pero imprevisibles, recibiendo entradas múltiples y eligiendo opciones que lo acerquen a un narrador de historias[22].

[22] Suhayda George: Video Game Play and Design: Procedural Directions.

Videojuegos en línea.

Debido a la naturaleza del ser humano de utilizar cualquier medio a su alcance para obtener formas de divertirse, Internet no ha sido la excepción y gracias a su popularidad se han extendido la cantidad de juegos que aprovechan las ventajas de estar "en-línea".

Un juego en línea se caracteriza porque permite a dos personas que están separadas físicamente el poder interactuar mediante el dispositivo utilizado para jugar (ya sea una PC, un teléfono móvil o una consola).

Este tipo de entretenimiento se ha visto influenciada recientemente por la gran cantidad de juegos que soportan la característica de ser jugados en línea, ofreciendo a los jugadores la posibilidad ya no de enfrentarse a un personaje artificial creado por los desarrolladores, sino en contra de otra persona, la cual se comporta de una forma mucho más inteligente, añadiendo un cierto factor competitivo a estos juegos.

Como cualquier aplicación empresarial en línea, un videojuego en línea requiere de una tolerancia cero a los fallos críticos que puedan presentarse. Es por esta razón que cada vez más compañías están aportando la infraestructura necesaria para ofrecer no sólo al jugador una experiencia aceptable al extender su interacción con otras personas, sino a los desarrolladores una serie de herramientas que les permitan tener una base sólida sobre la que puedan realizar todas sus pruebas.

La motivación de utilizar un juego de rol en línea para ejemplificar el uso de la tecnología J2EE es que este tipo de juegos simulan la interacción que podría tener cualquier sistema distribuido, es decir una serie de clientes accesando al mismo conjunto de datos para modificarlos constantemente y de la misma forma comunicarse entre ellos (ya sea mediante texto o acciones).

De igual forma, la tecnología seleccionada ofrece el soporte necesario para la creación de este tipo de aplicaciones, ofreciendo todos los componentes necesarios para que los desarrolladores puedan concentrarse tan sólo en hacer que las cosas funciones sin detenerse en como es que funcionan a un nivel más bajo que lo que sería la capa de negocio.

A continuación se dará un panorama general de lo que representa el mercado de los juegos en línea:

Tipos de Juegos.

Dentro de los juegos que se pueden encontrar en línea, estos pueden dividirse en seis grandes categorías:

Juegos Multijugadores Masivos (conocidos también como Juegos de Rol Masivos, Multijugadores en línea MMORPG). Son los más exitosos en lo que a juegos en línea se refiere. Este tipo de juegos permiten a más de 2000 usuarios el poder interactuar dentro de un mundo o mapa. Aquí, los jugadores crean representaciones de ellos mismos y los guían mientras adquieren experiencia, acumulando poder, conocimiento y riquezas. En este tipo de juegos, el mundo es estático y el aspecto que es variable es el estado del jugador.

Este tipo de juegos son, fundamentalmente, aplicaciones cliente servidor. Aquí el "mundo" está disponible las 24 horas del día los 7 días de la semana, permitiendo que el jugador pueda acceder a este en cualquier momento. Esta característica requiere que se cuente con uno o varios servidores bastantes potentes, así como políticas de recuperación ante fallas que permitan que el jugador no note cuando algún problema exista, a menos de que este sea demasiado grave.

El juego presentado como caso práctico puede ser englobado dentro de este grupo, ya que se cuenta con un mundo (mapa) sobre el que los usuarios interactúan. Así mismo con cada sesión del jugador, este aumenta sus habilidades sin afectar el estado del mundo.

Juegos de PC. Actualmente, la PC es la plataforma predilecta en cuanto al juego en línea se refiere. A diferencia de los MMORPGs, este tipo de juegos se basa en sesiones, donde un total de hasta 64 jugadores pueden interactuar en un mapa. Al terminar una sesión, los datos del jugador no se utilizan para acumularse y obtener experiencia (quizás la puntuación más alta o estadísticas similares).

Este tipo de juegos pueden adquirirse vía CD o mediante descargas de la red. El rango de géneros que se pueden encontrar en este tipo de juegos va desde los RPGs tradicionales, hasta los juegos de deportes, pasando por los juegos de estrategia, juegos de cartas, etc.

Juegos de Consolas. Actualmente, todas las consolas en el mercado cuentan con el hardware necesario para soportar los juegos en línea y se prevé que en el futuro esta situación no cambiará; cualquiera que desee entrar al mercado de las consolas deberá ofrecer una forma de conectarse y herramientas de desarrollo para juegos en línea. Al igual que con los juegos para la PC, se pueden encontrar gran variedad de juegos.

Juegos Inalámbricos. Son aquellos que permiten a los usuarios de teléfonos celulares, PDA[23] y cualquier dispositivo móvil el poder interactuar mediante este. Con la continua mejora tanto de los dispositivos como de la infraestructura que les da soporte, este tipo de juegos comenzará a popularizarse. Sin embargo, para que esto suceda, se debe además de ofrecer servicios de conectividad más baratos y de la misma forma cada compañía debe ofrecer contenido exclusivo que permitan al usuario el seleccionar a su proveedor.

Mercado.

Debido a que cada vez más personas adquieren una computadora personal con acceso a Internet, el número de clientes potenciales aumenta día con día. Sin embargo, de toda esta cantidad de gente con acceso a Internet se pueden identificar a dos grupos que son los que hacen uso de este medio para jugar:

Jugadores "hardcore". Son aquellos que cuentan con suscripciones a MMORPGs (principalmente)y cuentan con el dinero y tiempo para poder realizar los avances exigidos por el juego.

[23] Asistentes Personales Digitales

Jugadores de Juegos masivos. Este tipo de jugadores son aquellos que acceden a los juegos que están a su disposición principalmente de forma gratuita (de ahí e término masivos). Este es el mercado más grande, ya que son juegos con tiempos de descarga mínimos, interfaces sencillas y sobre todo, el usuario no debe invertir grandes cantidades de dinero (y tiempo) para poder jugar.

Cabe hacer notar que estos dos segmentos no son mutuamente excluyentes y una persona que pertenece a un segmento, puede pertenecer al otro.

La medida en la que este mercado crezca está íntimamente relacionada con los modelos de negocio que se sigan. Aquí es importante notar que si bien se eliminan muchas de las complicaciones de los canales de distribución tradicionales, se agregan otros cuantos que afectan la forma en la que se divide el dinero. En el caso de los juegos en línea, el dinero debe distribuirse generalmente entre los siguientes participantes:

- *Fabricante.*
- *Distribuidor.*
- *Proveedor del Servicio de Conexión.*
- *Proveedor del Servicio de Host.*

En muchas ocasiones, algún participante puede tener múltiples roles, dando como resultado que su participación en el mercado aumenta, pero también aumentan sus gastos y el riesgo de recuperar la inversión.

Desarrollo.

El desarrollo de juegos en línea es uno de los más contrastantes, ya que dentro de este tipo de productos podemos encontrar, tanto a proyectos sencillos cuyo tamaño no es más de un MegaByte, hasta juegos tan complejos que deben ser distribuidos en CDs o DVDs y que requieren de una gran infraestructura que les permita sostenerse. Sin embargo, comparte una misma característica y es que estos juegos son jugados por largos períodos de tiempo, ya sea por una interacción prolongada o múltiples interacciones de corta duración.

En el caso de un juego como el que se presenta, el convertirlo en un producto comercial requeriría no sólo de un esfuerzo de análisis y diseño, sino que además se debe de contar con grupos de soporte técnico, aseguramiento de calidad, etc. Así mismo se requiere del hardware (servidores, bases de datos, etc) para poder ofrecer un servicio de calidad a los usuarios que acceden a este, más si se tiene en cuenta que para juegos MMORPGs se cobra al usuario una renta por el uso del servicio, por lo que este espera que éste nunca falle.

A continuación se presentan algunas consideraciones que deben tenerse en cuenta al momento de diseñar este tipo de juegos:[24]

Patrones de Interacción. En este tipo de juegos, el aspecto social debe ser tan importante como la jugabilidad misma. Por esta razón deben implementarse los mecanismos adecuados que permitan al jugador el beneficiarse de la interacción con los demás jugadores y de la misma forma, penalizar aquel comportamiento que no esté enfocado a la consecución de los objetivos del juego o el hostigamiento de los demás jugadores.

[24] IGDA, IGDA Online Games Whitepaper 2003; www.igda.org

Duración de Sesiones de Juego. Ordinariamente, la duración de una sesión es de 3 a 6 horas en promedio. Debe tenerse en cuenta que pueden existir jugadores que no dediquen más de 10 horas semanales a esta actividad, mientras que por el contrario, pueden existir personas que dediquen más de la mitad de su día a hacer crecer a su personaje.

Duración del Juego. Generalmente, debido a que el Personaje creado por el usuario puede ser visto como su alter-ego, los jugadores tienden a continuar su "crecimiento" durante largos períodos de tiempo (inclusive años); por esta razón, se dice que este tipo de juegos cuenta con un tiempo de vida relativamente largo.

Mecánica de Juego. Como para cualquier tipo de juego en línea, se debe tener presente el concepto de latencia (tiempo en que el usuario realiza un comando y puede ver el resultado) para poder ofrecer una mejor experiencia de juego. Deben revisarse las llamadas realizadas del cliente al servidor y viceversa para poder determinar aquellas que son más utilizadas y poder hacerlas lo más óptimas posible.

Grandes aspiraciones: juegos de rol masivos.

La motivación de esta sección es describir porqué el presente proyecto, puede ser clasificado dentro de los MMORPG, acrónimo anglosajón para Juego de Rol en Línea Multiusuario Masivo (Multiplayer Masive On-Line Role Playing Game).

Se definen los puntos principales de comprensión de tal concepto:

Juego de rol. Este concepto define al tipo de juego en el que el jugador se identifica y asume el papel de su personaje. El jugador existe e interactúa con el universo ficticio del juego a través de su personaje y se desenvuelve de acuerdo a sus motivaciones y forma de ser particular.

En línea. Este concepto indica que el jugador debe conectarse a un servidor principal para poder desarrollarse dentro de una aventura y relacionarse con otros jugadores.

Multiusuario. Este concepto indica que más de un jugador puede participar del juego al mismo tiempo.

Masivo. Como extensión al anterior, se espera una gran población de jugadores dentro de un mismo universo, afectando todos el mismo ambiente y con posibilidad de entablar relación con todo jugador activo que forme parte de esta masa humana simulada.

El proyecto cumple muy bien con estas características por lo que la clasificación es correcta, sin embargo existen otros factores que han sido adoptados y que alimentan al concepto:

a) *Manejo de masividad.* Es decir, que no se defina el número de usuarios activos en el servidor, más que por las limitantes de los sistemas que lo soportan. Esto, además, puede implicar la particularidad de que, el universo, al ser compartido, tiene una existencia continua, y cuando un usuario se desconecta, los demás jugadores siguen transformándolo.

b) *Progreso de los personajes*. Esto indica que los jugadores deberán depurar las habilidades de sus personajes mediante la práctica, es decir, jugando. Con esto, el personaje será más sabio, fuerte, hábil o experto de acuerdo a las decisiones tomadas en el juego y siempre en función de el tiempo invertido.

c) *Factor ONLINE*. Esto es, no puede ser jugado sin conexión a Internet. El programa cliente realiza como función principal las tareas de interpretación, despliegue de gráficos (renderizado) y comunicación, de los datos enviados desde y hacia el servidor, y este ultimo es quien realmente hace posible la aventura. En este punto podemos dotar al cliente de funciones extra, pero siempre se hace patente la necesidad de relación con el servidor.

d) *Almacenamiento de los datos en el servidor*. Esto es para evitar la alteración de los datos, obtener alguna ventaja por parte de los jugadores, y reforzar a la vez el punto anterior. Aunque para este punto se puede tratar de definir nuevos esquemas de almacenamiento en los cuales se comparta esa responsabilidad, permitiendo la portabilidad del sistema. Por ejemplo que el usuario pueda modificar su inventario antes de entrar al juego y que pueda almacenarlo y transportarlo.

e) *Sistema para almacenar lo obtenido dentro del juego*. De alguna manera debemos hacer posible al jugador el contar con un sistema que guarde lo que en sus aventuras haya obtenido (dinero, items, armas) En esto se hacen distinciones muy especificas entre lo que el jugador puede considerar suyo y lo que forma parte del universo o de la aventura. Si es suyo podrá almacenarlo, si no lo es, deberá dejarlo para no obstaculizar el desarrollo de los eventos para los demás jugadores. Este almacenamiento sufre los mismos problemas que el almacenamiento del perfil del usuario, pero, como se explica anteriormente, es la búsqueda de una opción más flexible y a la vez segura.

f) *Que favorezca las relaciones entre jugadores*. En otras palabras, que permita la creación de una comunidad dentro del propio juego con otras personas.

Al cumplir con estos factores, se puede considerar el caso práctico como un MMORPG. Sin embargo no es intención central de este trabajo el desarrollo del juego, como tal, por lo que se omiten partes que pueden cumplir con el objetivo de aportar algo significativo al género, enfocándonos a su realización como ejemplo a las tecnologías aplicadas. De hecho son esos factores los que dan a cada juego su individualidad y los refuerzan como propuestas interesantes, pero hay que recordar que el motivo del presente proyecto es demostrar la utilización de la standar J2EE.

Java como lenguaje de programación de Juegos.

La historia de Java, así como sus características son discutidas en otras partes de este documento, por lo que nos centraremos en la utilización de Java como lenguaje de programación para juegos (especialmente en línea). Para lograr esto vamos a estudiar las tres partes fundamentales de Java que pueden ser aplicadas al desarrollo de este tipo de proyectos:

J2ME (Java 2 Micro Edition)[25]. Esta versión de Java es la que se utiliza en dispositivos con recursos limitados, como es el caso de teléfonos celulares o PDAs. Al hablar de dispositivos con recursos limitados hablamos de las siguientes características:

- *Limitada velocidad del procesador (no medida en GHz).*
- *Memoria limitada, con un mínimo de 128 KB.*
- *Limitadas capacidades de conexión (protocolos, velocidad).*
- *Interfaces de Entrada y Salida Limitadas.*

Esta tecnología se encuentra en dispositivos dispares en términos de escala, formas de utilización y capacidades. Para hacer frente a este obstáculo, J2ME introduce la noción de perfiles de configuración. Una configuración define las características de bajo nivel de la plataforma, mientras que el perfil extiende la configuración haciendo referencia a las características particulares del dispositivo, así como su forma de uso. Generalmente un juego se desarrolla para un perfil específico, así, este juego podrá ser utilizado en todos aquellos dispositivos que comparten dicho perfil.

J2SE (Java 2 Standard Edition). Esta es la versión estándar de Java y tiene como objetivo el desarrollo de aplicaciones de escritorio y es la base para la utilización de la edición empresarial (J2EE). Como lenguaje de programación, Java es elegante y ofrece funciones como recolección de basura (limpieza de memoria), así como un conjunto de librerías que ayudan a resolver problemas comunes a cualquier sistema (incluidos el desarrollo de juegos). Esta versión incluye además un conjunto de librerías que permiten el manejo de Gráficos en 2D y 3D.

J2EE (Java 2 Enterprise Edition). Es una extensión de la anterior y está enfocada al desarrollo de aplicaciones empresariales. El uso de J2EE generalmente está aplicado a la programación de componentes del lado del servidor. Esta extensión de Java ofrece las capacidades necesarias para poder ofrecer conexión cliente-servidor en un modelo multicapas.

Actualmente muchos servidores de aplicación son compatibles con J2EE, es decir, ofrecen soporte para Ejes, JSPs (Java Server Pages), JMS (Java Message Service) y Servlets y al igual que con J2SE permiten al programador concentrarse en la resolución del problema particular ofreciendo un entorno en el que pueden desplegar sus aplicaciones.

---

[25] Una extensa descripción sobre J2ME está fuera de los límites del proyecto, pero se coloca para dar una referencia de las tecnologías JAVA que se utilizan para la programación de videojuegos en línea.

Generalmente se tiene la idea de que Java no es un lenguaje óptimo para la programación de juegos debido a que es más lento en comparación con C o C++, lenguajes tradicionalmente utilizados para estos fines. Sin embargo, mientras las tecnologías de compilación, velocidades de procesador y mayor manejo de memoria, Java será utilizado con mayor frecuencia para la programación de juegos, ya que cuenta con una gran cantidad de librerías que, como se ha mencionado anteriormente, ayudan al desarrollador a concentrarse en la resolución de su problema, en lugar de hacerlo en situaciones que las librerías ya resuelven.

Lo que los usuarios no saben de los videojuegos y deberían saber.

- *No existe el software libre de bugs.* Todo software tiene errores. Algunos son aceptables, ya que no interfieren en el desempeño del juego. Se deben eliminar aquellos errores críticos que impidan que el software cumpla con su función.
- *Un juego no es como cualquier software.* Requiere de una gran inversión y gran cantidad de trabajo de un gran número de personas.
- *Cuando un juego se retraza no es por la incapacidad de los programadores.* Mientras más tiempo se le dedique a un juego este será mejor y podrá satisfacer mejor las necesidades de los jugadores.
- *Sólo se obtiene el producto por el que se pagó.* Si uno paga por un juego malo y barato, sin antes haberse informado, es culpa del consumidor y no del programador.
- *Si se quiere algo innovador, se tiene que votar con el dinero y el interés.* Si los consumidores no compran juegos innovadores, no deben quejarse de que todos los juegos del mercado son iguales.
- *Los Reviews y demos tienen una razón de ser.* Su utilidad es la de informar sobre las capacidades que tiene un juego para satisfacer a los consumidores. Informan al consumidor sobre las opciones en las que puede gastar su dinero.
- *Si se quieren juegos buenos y baratos no se deben comprar al dia siguiente de su salida.* Al paso del tiempo el software tiende a abaratarse y además existen líneas de descuento que ofrecen el mismo juego a un precio menor, esto sí se puede esperar uno o dos meses después de la salida del juego.
- *La piratería no es "cool" ni benéfica en el futuro (o el presente).*
- *No debe uno quejarse acerca de lo que uno quiere.* Si uno está deseoso de un juego que resulta ser malo y pagó por él, no hay que quejarse.

Ingeniería y Ciencia.

La distinción que se hace de Ingeniería y Ciencia dentro del software es la misma que se hace dentro de cualquier otro campo. Los científicos aprenden lo que es cierto, la forma de probar hipótesis y cómo extender los conocimientos en sus campos; por su parte los ingenieros aprenden lo que es cierto, lo que es útil y cómo aplicar estos conocimientos para solucionar problemas de forma eficiente. Al ser científico se puede tener el lujo de tener un campo de acción muy cerrado y especializado, mientras que por su parte un ingeniero se requiere de un conocimiento amplio (dentro de una especialización) de todos los factores que afectan a un producto en desarrollo. Generalmente un científico se prepara para seguir estudiando y desarrollar aún más su campo de conocimiento, mientras que un ingeniero se prepara para integrarse inmediatamente a un mercado laboral en la búsqueda de solución a problemas.

Un Ingeniero, según la definición del diccionario es una persona que aplica el conocimiento científico y matemático con fines prácticos. Esta definición encaja perfectamente con lo que muchos programadores intentan hacer: aplicar algoritmos matemáticos, métodos de aseguramiento de calidad y otras prácticas para desarrollar software.

Este tipo de proyectos son complejos y riesgosos, razón por la que a diferencia de otras aplicaciones de la Ingeniería, los proyectos de software deben enfocarse en la optimización del proyecto, por lo que además de trabajar con el fin de cumplir los objetivos planteados, se debe trabajar para lograr la mayor eficiencia posible. Por esta razón se requiere de gente con la experiencia necesaria para que el proyecto no quede sin terminar, experiencia con la que muchos ingenieros están más familiarizados que los científicos.

Ahora bien, dadas ambas definiciones, el desarrollo de sistemas debe plantearse la cuestión sobre si el desarrollo del software debe ser o no considerado como una Ingeniería: la respuesta es sí, pero no por esto es minimizando el esfuerzo de los científicos, ya que como Roger Pressman menciona "*la Ingeniería del Software es un área de las Ciencias de la Computación que ofrece métodos y técnicas para el desarrollo del software*"[26]. Por lo que, en lugar de abrir la brecha entre ambas disciplinas debe existir una definición de cada uno de los campos de acción de los egresados en estas carreras.

Así, por una parte, un Científico se encarga de dar el soporte a todo el conocimiento aplicado por el Ingeniero, esto con la generación de nuevo conocimiento y expansión de las áreas de acción sobre las que el Científico trabaja. Por su parte, el Ingeniero es el encargado de aplicar no sólo este conocimiento de forma práctica, sino que además es el encargado de llevar a un sistema a buen término mediante la aplicación de técnicas de Análisis y Diseño, Métricas, Técnicas para el Aseguramiento de Calidad, etc.

Más que una contraposición entre las dos disciplinas, lo que se busca es una interacción entre ambas, ya que cada una es complementaria de la otra.

[26] Pressman Roger, Ingeniería del Software, un enfoque práctico.

Educación.

Actualmente no se puede negar la influencia social, cultural y educativa de los videojuegos[27] y en definitiva, ya nadie discute que se puede aprender jugando. Éstos se han convertido en agentes socializadores, junto a los padres, la escuela y otros medios como la televisión, aún cuando los juegos no se engloban, dentro de los métodos convencionales de enseñanza, como una forma de impartición conocimiento.[28]

Por lo tanto, no solamente se pueden clasificar como juegos educativos los que fueron creados con ese fin, si no también aquellos creados para entretener, ya que éstos pueden ejercer un importante papel para el debate, tanto en las escuelas como en los hogares. Los juegos inherentemente pretenden despertar la curiosidad dentro del jugador, ya sea por medio de elementos fantásticos, retos mentales o actividades susceptibles de coordinación, conocido éste último término como movimiento psicomotor fino [29].

Hablando particularmente acerca de los videojuegos masivos como agentes educativos, al existir relaciones sociales complejas, se puede tener una mejor comprensión de fenómenos sociales, económicos y psicológicos existentes en nuestra realidad. Dichos juegos implican la interacción dentro de un mundo con un ambiente y una economía virtuales que ayudan al jugador a entender los diferentes factores comprometidos en una sociedad, tales como el cuidado de los recursos, la ley, la oferta y la demanda, etc.

Videojuegos en las escuelas.

El videojuego introducido en la escuela se trasforma, ya no es un programa para jugar sino que tiene además una intención educativa. Se utiliza para desarrollar determinadas habilidades, para motivar a los alumnos y/o para enseñar un contenido específico, ayudando a responder a diferentes planteamientos. Así, estos pueden cubrir desde áreas cognoscitivas específicas, hasta propuestas basadas en el uso de destrezas cognitivas y motoras (razonamiento lógico, resolución de problemas, viso motricidad, etc)[30].

Según el escritor Daniel Gómez Cañete, al desarrollar un juego educativo se deben tomar en cuenta, tanto la edad con la que cuentan los niños o jóvenes, como el tiempo dedicado al juego[31]. De este estudio se concluye que algunas de las principales características con las que un video educativo debe cumplir son:

- Permitir el aprendizaje de diferentes tipos de habilidades y estrategias.
- Ayudar en la dinámica de las relaciones entre los miembros de la comunidad, no sólo desde el punto de vista social sino también en el mismo aprendizaje.

---

[27] Gómez Cañete Daniel; Cuatro retos para el futuro de los Videojuegos.
[28] Game Research; Education.
[29] Movimiento Psicomotor fino. *movimiento del cuerpo, bajo dirección de la mente.*
[30] Pons Juan de Pablos; Las Tecnologías de la información y la comunicación: Un punto de vista educativo.
[31] Gómez Cañete Daniel; Cuatro retos para el futuro de los Videojuegos.

- Permitir la introducción del análisis de valores y conductas a partir de la reflexión de los contenidos de los propios juegos.

Por otra parte, las razones principales por las que no se ha fomentado este tipo de videojuegos son, según la desarrolladora de Juegos Educativos Silvia Héctor, la cual trabaja para Vermie, empresa dedicada al desarrollo de software educativo[32]:

- *Concepción errónea de eficiencia económica.*

Es común entre las instituciones educativas enfocarse en el aprendizaje de un sistema operativo y otras herramientas administrativas, conocidas como "paquetería", dejando de lado a los videojuegos como una herramienta de apoyo al estudio de las materias tradicionales con recursos computacionales.

- *Falsa sensación de autosuficiencia.*

Eventualmente, maestros o instructores de informática escolares, conscientes de la necesidad de contar con software que les permita apoyar sus cursos, deciden destinar parte de su tiempo para la elaboración del software necesario. Desafortunadamente, esta decisión suele tener como consecuencia, un retraso en la adquisición del software correcto, ya que generalmente estos proyectos no son concluidos o arrojan productos muy débiles, en comparación con software disponible comercialmente o elaborado por los profesionales correspondientes.

- *Insatisfacción por la oferta de software.*

Ocasionalmente, profesores y directivos escolares se quejan de que el software existente en el mercado no satisface sus requerimientos pedagógicos. Sin embargo, a menudo dichas quejas no son derivadas de un verdadero análisis de las opciones existentes al no contar con un conocimiento real del software disponible.

- *Naturaleza intangible del software.*

No solo en el contexto educativo, sino en todos los órdenes de la informática, el software ha "vestido la más discreta indumentaria"; seguramente por su carácter intangible. A menudo se torna "invisible" ante los ojos de la mayoría de la gente. Esto explica que frecuentemente los directivos escolares piensen en el equipo y el personal, como los únicos ingredientes necesarios para montar un laboratorio de cómputo, y por tanto, destinen recursos económicos sólo a estos rubros.

Comunicación.

A la formación de una comunidad mediante la convivencia en un videojuego se le conoce como CMC (Comunicación Mediada por Computadora)[33]. Esto hace que el contexto social en el cual se encuentre el individuo mientras ocurre el CMC juegue un papel importante dentro del juego, dando origen a nuevos procesos y actividades las cuales crean nuevos retos, modificando la relación inicial del jugador y su contexto, incluida la relación con su mundo exterior.

Anne-Marie Sheliner, psicóloga social citada por Daniel Gómez Cañete describe a los videojuegos de rol en línea como una "fuente de desarrollo social", debido a la organización de personas en clanes, es decir, grupos en oposición a otros grupos. Algunas veces, se extienden los lazos mas allá de un juego, creando amistades

[32] Silva Hector; Software Educativo: Ingrediente Olvidado.
[33] Ferenback, J.(1999). En ingles. Computer-Mediated-Communication

u otro tipo de relaciones, haciendo de una reunión para jugar un evento social. Inclusive se han formado convenciones a partir de este tipo de juegos o se han realizado bodas, tanto dentro del juego como fuera de él[34].

Cuando las comunidades forman una semántica propia, compartiendo conocimiento, resolviendo problemas, jugando, construyendo, las relaciones entre sus miembros crecen. Los juegos son grupos formales porque tienen reglas; son sistemas porque son colecciones de partes que interactúan entre si de manera compleja[35].

Los videojuegos permiten una interactividad de la cual, según Pierre Lévy, profesor del Departamento de Hipermedia de la Universidad Paris-Saint Denis, destacan las siguientes variables[36]:

- Las posibilidades de apropiación y personalización del mensaje recibido, sea cual sea su naturaleza.
- La reciprocidad de la comunicación.
- La virtualidad.
- La implicación de la imagen de los participantes en los mensajes.
- La telepresencia.

Aun cuando existan varios grados de realismo al tratar de representar ese medio físico, tales simulaciones pueden ser calificadas como un estado donde las imágenes tratan desesperadamente de producir un efecto dentro de la realidad, es decir una hiper-realidad, donde los jugadores interactúan con el videojuego en un nivel emocional. La estrategia es compartir la responsabilidad entre autor y lector (usuario y escritor a la vez), creando mundos autónomos.

Todo esto, permite la creación de una nueva cultura, la cual actúa como si trascendiese los límites de nuestra existencia, haciendo que la realidad presentada sea simplemente una simulación. Desde su origen, los videojuegos brindaron la oportunidad de satisfacer la necesidad de expresar las fantasías, al permitir ser una parte del juego que permite la realización de éstas; así, el jugador desempeña los papeles tanto de espectador como participante de forma simultánea.

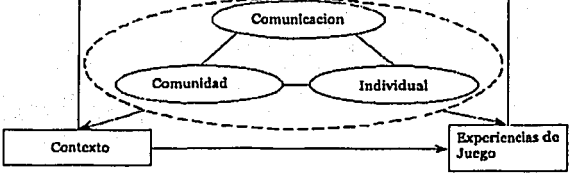


Ilustración 19: Experiencias de comunicación.

[34] Gómez Cañete Daniel; Vidas Virtuales, problemas reales.
[35] Ahuna Cindy; Online Game communities are social in nature.
[36] Gros Salvat Begoña; La dimensión socioeducativa de los juegos.

En el anterior esquema, realizado por Primo Dolzan, Virtual World: "Ultima Online" ,se representa el manejo de un mundo virtual. El jugador tiene antecedentes, los cuales le permiten tener una percepción única del juego en línea. Al estar en contacto con otros individuos, se inicia un proceso CMC, formando, a partir de esta hyper-realidad, un contexto de su propia realidad, modificando las experiencias del jugador junto con su percepción del juego.

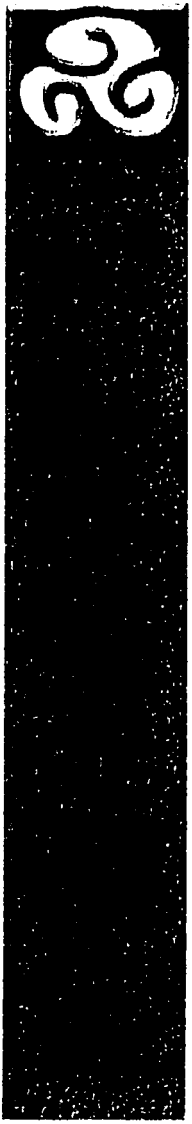
Entretenimiento.

Como seres humanos, vivimos en un mundo físico, pero nuestra dependencia de él rivaliza con la dependencia que mostramos hacia nuestras propias fabricaciones. La tecnología nos permite sobrepasar los límites del mundo físico donde "las fantasías que expresan nuestros deseos y temores en una forma de arte, de entretenimiento y liberación".Por lo tanto, podemos decir que los videojuegos ofrecieron la oportunidad de satisfacer la necesidad de expresar las fantasías, al permitir ser una parte del juego. Cuando un individuo forma parte de un entretenimiento, presenta una claridad de metas y de ambiente, desarrolla habilidades presentando una conciencia de si mimso y un balance entre la realidad y la fantasia. Inclusive los videojuegos han ayudado a otras formas de entretenimiento como el cine al desarrollar tecnología que pueda ser utilizada en esta.

Entrenamiento.

Los videojuegos aportan una gran ayuda a varias organizaciones y áreas de investigación, en las que se utilizan videojuegos para la simulación, permitiendo recrear diferentes situaciones por medio de guantes virtuales, cascos para visión virtual y sillas de juegos, con el fin de acostumbrar a un individuo a reaccionar "correctamente" a ciertos estímulos. Principalmente este tipo de entrenamiento se da en:

- *Ejército.* Se crean juegos que permiten la instrucción de los soldados que formarán parte de las fuerzas de élite. El objetivo es desarrollar la capacidad de tomar decisiones adecuadas en periodos de tiempo ínfimos en medio de situaciones críticas.
- *Medicina.* Se recrean, por medio de la simulación, varias enfermedades y padecimientos, con el fin de que el estudiante pueda desarrollar un criterio profesional adecuado.
- *Aviación.* Se utilizan simuladores para recrear situaciones difíciles a los que se deben enfrentar los pilotos, permitiendo así, la adquisición tanto de los conocimientos como de las habilidades que les permitan la toma adecuada de decisiones, en caso de que se encuentren dentro de algunas de estas situaciones, con la ventaja de que si dentro de la simulación la decisión tomada no es la correcta, no existen pérdidas humanas y/o materiales.



Capítulo 2.

Objetivo del Proyecto

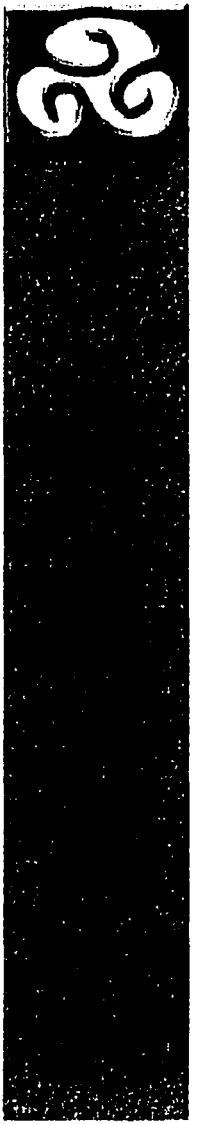
CAPÍTULO 2. OBJETIVO DEL PROYECTO

Objetivo general.

Ejemplificar y documentar el uso de la tecnología EJB(Enterprise Java Beans) para soluciones empresariales distribuidas y multicapa, mediante el desarrollo de una aplicación que abarque los aspectos más importantes de dicha tecnología, la cual sirva de referencia a la realización de proyectos similares.

Objetivos específicos.

- Presentar la documentación que sirva como guía para el aprendizaje de EJB.
- Integrar la literatura y material de consulta, necesarios para la introducción de nuevos elementos a proyectos de desarrollo de soluciones empresariales.
- Apoyar el proceso de aprendizaje con el desarrollo de una aplicación que aproveche las principales características del estándar J2EE.
- Desarrollar y aplicar una metodología de trabajo adecuada a la dependencia, a la vez que aplicable a los productos de software futuros que integren esta tecnología, en respuesta a la adopción de las nuevas herramientas de desarrollo.
- Promover el conocimiento de esta tecnología favoreciendo la difusión de la misma, que puede ser un punto de apoyo al desarrollo de software en la UNAM y otras organizaciones.



Capítulo 3.

Alcance del Proyecto

76-A

CAPÍTULO 3. ALCANCE DEL PROYECTO

El producto final del presente proyecto, en relación con los objetivos planteados inicialmentre, cumple con las siguientes características:

- Generar y organizar una guía de referencia para el desarrollo de soluciones empresariales basadas en la tecnología EJB en forma estandarizada, es decir, independiente de las plataformas de desarrollo y despliegue de componentes.
- Presentar el caso práctico desarrollado como ejemplo de aplicación de la tecnología, mismo que sirve de guía de consulta a los tópicos abarcados por el presente trabajo.
- Incluir la documentación y explicación de las características principales tanto de EJB, como de tecnologías y herramientas relacionadas.
- Proponer una metodología de desarrollo adecuada a la institución, referente al desarrollo de sistemas distribuidos y multicapa, de acuerdo con el estándar J2EE.
- Incluir esquemas y diagramas que faciliten la comprensión de los temas expuestos.

Cabe mencionar que el presente proyecto, no es un manual para desarrolladores, sino una guía de esquemas de aplicación, que requiere de conocimiento previo tanto del lenguaje como de las herramientas elegidas para el proceso de desarrollo. Además, independientemente del caso práctico seleccionado, no es finalidad de este trabajo el presentar una guía de programación de videojuegos.

Capítulo 4.

Metodología

CAPÍTULO 4. METODOLOGÍA.

INTRODUCCIÓN: ¿PORQUÉ ADOPTAR UNA METODOLOGÍA DE DESARROLLO?

Una metodología de desarrollo de proyectos es una herramienta que nos permite tener un enfoque más claro de lo que se desea obtener con un sistema, así como la forma en que habremos de lograrlo.

Para esto, es necesario seguir una serie de pasos y procedimientos de investigación, con lo que se consigue abrir las perspectivas del nuevo sistema y así lograr una visión más clara de los problemas que serán resueltos.
El desarrollo de sistemas al seguir una metodología:

- Facilita la comprensión del problema a resolver, el ámbito en que será utilizada la solución, los requisitos de los usuarios finales y de la empresa, y en general, los factores que deben tomarse en cuenta para el producto final.

- Permite estudiar las alternativas de solución propuesta, y determinar la mejor forma de crear software bien diseñado, robusto, de fácil mantenimiento y empleando las tecnologías más adecuadas.

- Implementa reglas bien definidas así como herramientas de apoyo a los desarrolladores como soporte al proceso de análisis, diseño, construcción, implementación, etc. Lo que favorece a la integración de los sistemas y al establecimiento de convenciones que favorecen la comunicación efectiva.

- Simplifica el proceso de planeación y organización del trabajo de desarrollo, es posible evaluar la factibilidad de manera más precisa y establecer calendarios de actividades enfocados a logros, tomando en cuenta verificaciones periódicas de resultados.

- Permite organizar de forma más óptima las funciones del sistema así como las responsabilidades de los componentes, favoreciendo el aprovechamiento de características como la reutilización de código, abstracción, modularidad, ocultamiento de información, etc.

- Aporta mayor predictibilidad y control de los resultados de las fases de la construcción de software así como del resultado final, lo que permite contar con formas de control para saber qué esta ocurriendo en cualquier punto del proyecto.

- Facilita la adaptación a los cambios, el mantenimiento de los sistemas, la transmisión de conocimiento, el manejo de errores y, en general, favorece a que se cuente con documentación completa y correcta de los sistemas.

DESCRIPCIÓN DE LA METODOLOGÍA ADOPTADA.

Poseer un a metodología que abarque las fases involucradas en la creación de software, a la medida de los sistemas desarrollados en la organización, permite unificar las formas de trabajo así como los productos de las etapas de desarrollo. La metodología de desarrollo descrita a continuación se generó a partir del estudio de metodologías orientadas a objetos establecidas entre los que se encuentran:

- "Diseño Orientado a objetos"[37]
- "Objectory" (Ivar Jacobson[38]
- "Técnica de modelado de objetos" [39]
- "Análisis y diseño orientado a objetos" [40]
- "RUP: Rational Unified Process" [41]

Además del análisis de técnicas de desarrollo de proyectos, desarrollo de aplicaciones multicapa y de la observación de los métodos, procesos y herramientas adoptadas de la organización. Con lo anterior se unifica una metodología adecuada al área de desarrollo de sistemas de TVUNAM, para quien es desarrollado este documento.

---

[37] Gary Booch, Object Oriented Design with Applications Benjamming Cummings 991

[38] Object-Oriented Software Enginnering A Use Case Driven Approach Addison-Wesley 1992

[39] James Rumbaugh, Object Oriented Modeling and Design Prentice Hall 1991

[40] Craig Larman, UML y Patrones Introducción al análisis y diseño orientado a objetos, Prentice may, 1999

[41] Booch, G., Jacobson, I, Rumbaugh, J. The UML Specification documents, 1997

Análisis.

La finalidad de esta fase es conocer las necesidades de la organización y comprender a fondo el problema identificado, analizando sus antecedentes, requisitos específicos y contexto, para comenzar a dar forma a los conceptos de la solución sistematizada que se propondrá. Entre sus actividades comprende:

Actividad	*Descripción*	*Productos y artefactos generados.*
Detección de necesidades.	Se identifican las necesidades que dan origen al proyecto, así como los factores, internos y externos que deben ser tomados para el desarrollo del proyecto.	*Identificación de factores.* *Identificación de necesidades.*
Definición del problema.	Se definen las áreas y temas sobre las que será planteada la solución, de acuerdo a los parámetros de las necesidades. Se definen los objetivos y metas, la visión inicial de la solución, la metodología de trabajo y se evalua la mejor alternativa de solución con base en los aspectos identificados.	*Identificación del problema.* *Factores a tomar en cuenta.*
Definición de factibilidad.	Se identifica la disponibilidad de recursos existente y se compara con los requeridos por el proyecto en términos técnicos, económicos y operativos.	*Definición de factibilidad técnica* *Definición de factibilidad económica.* *Definición de factibilidad operativa.*
Planeación del proyecto.	Se delimita el problema en función de la solución propuesta, justificando los objetivos desarrollados inicialmente. Se definen las etapas de desarrollo y el control que se aplicará. Se concretizan los resultados del análisis.	*Antecedentes.* *Alcance.* *Objetivos.* *Justificación.* *Metodología* *Plan de trabajo.*
Especificación de requerimientos	Se identifican los requisitos específicos que los usuarios finales, la organización y las herramientas establecen.	*Panorama general.* *Descripción de clientes.* *Metas del sistema.* *Listado de funciones del sistema.* *Listado de atributos del sistema.*
Descripción de procesos.	Se identifican los procesos principales del nuevo sistema en función a los requerimientos, formando un esquema inicial de funcionamiento del sistema.	*Casos de uso.* *Diagramas de casos de uso.*
Modelado conceptual del sistema.	Se identifican los actores y conceptos principales para hacer el primer modelo del funcionamiento del sistema, que posteriormente serán traducidos a clases, atributos y funciones del sistema.	*Modelo conceptual del sistema.*

Diseño.

Contando con el conocimiento generado en la etapa previa, en esta fase se comienza la cimentación de los conceptos de análisis en elementos más concretos y cercanos a la construcción. Los modelos y artefactos de esta etapa toman en cuenta el lenguaje y herramientas que se utilizarán, para definir la estructura en términos de clases, responsabilidades, componentes, capas y arquitectura en general. Entre sus actividades comprende:

Actividad		*Descripción*	*Productos y artefactos generados.*
Determinación de la arquitectura.		Se analiza la estructura requerida por el sistema para planear niveles y capas principales de la aplicación, describiendo cada uno de ellos.	*Descripción de la arquitectura. Descripción de las capas o niveles de trabajo.*
Modelo de capas, niveles de trabajo y componentes del sistema y asignación de responsabilidades.		Se modifica el modelo conceptual adaptándolo a la tecnología y lenguaje de desarrollo del proyecto. Se definen las clases, paquetes y capas y las relaciones entre ellas. Se analizan las responsabilidades y rol de cada componente dentro del sistema.	*Diagrama de clases.*
Diseño de las capas del sistema.		Se estructuran las clases identificadas en las capas que les son correspondientes, se determinan las responsabilidades de cada nivel de trabajo.	*Diagrama de componentes.*
	Capa de presentación.	Se definen los componentes que comprenderá la interfaz del usuario.	*Diagramas de colaboración*
	Capa de negocio.	Se definen los componentes que resolverán las reglas y funciones de negocio.	
	Capa de datos	Se define la arquitectura de la base de datos así como los componentes que harán posible el acceso a ella.	
Descripción de la tecnología.		Se describen los aspectos particulares de la tecnología seleccionada para la construcción e implementación del sistema.	*Manuales técnicos.*

Construcción.

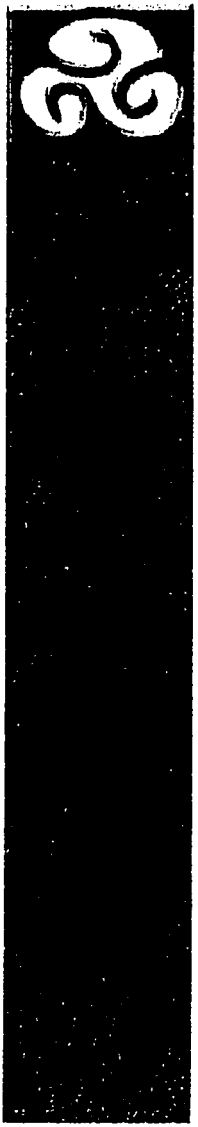
Siguiendo el diseño del sistema se traducen los esquemas y conceptos generados en código, planeando antes la forma en que se atacará la programación de los módulos identificados. Además se toman en cuenta pruebas periódicas para el control de la calidad de los sistemas. Entre sus actividades comprende:

Actividad		*Descripción*	*Productos y artefactos generados.*
Determinación de los ciclos de construcción.		Se planea el trabajo de construcción del sistema definiendo el orden y organización para desarrollar los módulos del sistema, definiendo metas.	*Plan de construcción.*
Construcción de las capas arquitectónicas.		Se genera el código de las clases diseñadas.	*Código y documentación*
	Infraestructura.	Se establece la arquitectura de clases que sirven de esqueleto al sistema, programando las funciones básicas.	
	Funciones del sistema.	Se incluyen funciones particulares y se construye cada capa aumentando el nivel de detalle.	
Administración de pruebas.		Se establece la forma en que se llevarán a cabo las pruebas tanto de las aplicaciones aisladas como de la comunicación e integración de las mismas.	*Plan de pruebas.*
Construcción de aplicaciones.		Se determinan las clases de acuerdo a los paquetes que serán instalados en diversos equipos. Se definen las clases cliente, servidor y datos para integrar los módulos que serán distribuidos.	*Aplicaciones instalables.*
Generación de manuales y documentación del lenguaje (java doc).		Se perfeccionan los manuales (técnicos y de usuario) y la documentación del código generado.	*Manual de usuario. Documentación de código.*

Implementación, despliegue y mantenimiento.

Comprende la instalación y puesta en marcha de los sistemas siguiendo la arquitectura elegida. Esta etapa evalúa planes de mantenimiento a sistemas y la distribución del software en los equipos de la organización. Entre sus actividades comprende:

Actividad	*Descripción*	*Productos y artefactos generados.*
Distribución de módulos.	Se lleva a cabo la instalación del sistema en servidores y clientes definidos por la arquitectura, así como los componentes destinados a servir a su funcionamiento.	
Puesta en marcha.	Se realizan pruebas de instalación, se entregan manuales y se da la introducción al sistema.	
Elaboración del plan de mantenimiento.	Se plantean los tiempos de mantenimiento que solicita el sistema así como los recursos que requerirá.	*Plan de mantenimiento.*



Capitulo 5.

Desarrollo del caso práctico : videojuego en línea.

Capítulo 5.

Desarrollo del Caso Práctico: VideoJuego en Línea

ANÁLISIS DEL CASO PRÁCTICO.

Detección de necesidades.

Un proyecto surge de una necesidad de la organización, y esta, a su vez, esta condicionada a elementos tanto internos como externos que deben ser tomados en cuenta desde el inicio del proceso de desarrollo ya que observarlos nos acerca a la certeza de que se cumplirán los objetivos de forma óptima para la organización.
Los factores identificados para el presente proyecto son:

- Rotación de personal en el área de desarrollo de sistemas, favorecido por los programas de becas, fuente principal de personal en el área.
- El perfil de los nuevos integrantes del equipo de desarrollo es, en su mayoría, gente joven , de los últimos semestres de carreras universitarias relacionadas con la informática.
- Incremento en la complejidad de las herramientas de desarrollo, debido a la adquisición de nuevas tecnologías.
- Crecimiento en los requerimientos del área, referentes a los conocimientos del personal, necesarios para el proceso de desarrollo de nuevos sistemas.
- Alto costo en tiempo del personal con antigüedad, al dedicarlo a la transmisión de experiencia y capacitación a nuevos integrantes del equipo, implicando uso poco efectivo de la información.
- Escasez de documentación de referencia suficiente para introducir a nuevos elementos al manejo de las tecnologías adoptadas por la organización.
- Deseo de explorar nuevas alternativas que favorezcan el aumento en rapidez y reducción en costos de la inducción de nuevos elementos humanos al área de desarrollo de sistemas.

Con lo anterior se busca establecer si algún elemento (equipo, proceso, personal, etc.) dentro de la organización no cumple de manera óptima con los objetivos o metas, o bien, que se requiere de un sistema no existente para hacer frente a una necesidad.

Existen diversos métodos para analizar las necesidades, en nuestro planteamiento se sigue el análisis creativo, que involucra la investigación de nuevas alternativas para solucionar problemas actuales. Como resultado de el análisis de necesidades se determina que:

- Se requiere contar con herramientas y documentación que favorezcan el uso eficiente de la información en términos de transmisión de experiencia, introducción a la nueva tecnología y referencia para todo el proceso de desarrollo.

Definición del problema.

Una vez detectadas las necesidades existentes en la organización se requiere definir las áreas sobre las que será planteada la solución, de acuerdo a los parámetros que proporciona la necesidad y no abarcando más allá de los que delimita. El desarrollo del proyecto no consiste solo en la solución de problemas sino también implica definir la mejor solución posible tomando como base aspectos específicos identificados, por lo que se considera importante la aplicación de la creatividad e innovación.

Se busca determinar las fronteras y el alcance de las necesidades que se desea atender y sobre las cuales existen posibilidades de definir un proyecto. Por lo anterior se hace necesario conocer a fondo la problemática a la que nos enfrentamos, para lo que se definen detalladamente los alcances y fronteras de un proyecto, con lo cual es posible establecer una guía de operación en el desarrollo del proyecto.

Posteriormente, como resultado de una fase de generación de ideas, que corresponde al trabajo que se lleva a cabo para particularizar las alternativas de solución de problemas, y luego de evaluar las alternativas obtenidas se determina la solución a desarrollar, misma que debe cubrir las necesidades observando. En esta etapa se logra:

- Determinar metas y objetivos.
- Crear una visión inicial.
- Adoptar una metodología.
- Evaluar recursos disponibles.
- Planear un equipo de trabajo.
- Preparar un plan inicial de trabajo.
- Determinar responsabilidades.

Se identifican los factores que son importantes para el desarrollo del proyecto, aislando todos aquellos aspectos que no interfieren en el mismo. Los aspectos a observar en el presente proyecto son:

- *Área objetivo*: Desarrollo de sistemas.
- *Usuario objetivo del proyecto*: Becarios de nivel universitario.
- *Utilidad requerida*: introducción a los conceptos de EJB , referencia a la tecnología EJB, referencia al desarrollo de aplicaciones distribuidas, documentación de apoyo, caso práctico de la aplicación de la tecnología.
- *Recursos disponibles*: 4 desarrolladores con experiencia previa en la tecnología EJB, 4 equipos dedicados a desarrollo, un equipo dedicado a Servidor de aplicaciones, Ambiente de desarrollo Jbuilder6.0, ambiente de despliegue Borland Enterprise Server 5.0, y una base de datos relacional SQL Server 7.0.

Definición de factibilidad.

La factibilidad se refiere a la disponibilidad que se tiene de recursos para llevar a cabo los objetivos y metas planteados en términos operativo, económico y técnico. Con esto, se logra definir las posibilidades de éxito para conseguir la solución de las necesidades. El estudio de factibilidad aplicado al presente proyecto determina que:

Factibilidad técnica.

Indica si el nuevo sistema representa una mejora tangible frente a los procesos y sistemas actuales, además de que se cuente con los requerimientos técnicos para llevarlo a cabo.

- El proyecto representará una mejora al proceso de adquisición de conocimiento, haciendo posible que un nuevo elemento pueda, de forma más autónoma, comprender los procesos que requiere para integrarse al proceso de desarrollo y cumplir con las funciones que le sean asignadas.
- Se cuenta con la tecnología y equipo suficientes para llevar a cabo el desarrollo del proyecto. Los requisitos para el desarrollo son:

- Personal: 4 desarrolladores con conocimiento en Java y J2EE
- Equipos de desarrollo: 4 computadoras personales.
- Equipo de despliegue: 1 Servidor de aplicaciones/ Administrador de bases de datos.
- Software y licencias: Desarrollo: 4 Jbuilder 6.0
  Despliegue: 1 Borland Enterprise Server 5.0
  Base de datos: 1 SQL Server 7.0

Factibilidad económica.

Es el estudio de los recursos que deberán ser invertidos para el desarrollo, y la determinación del costo que implica cada fase del desarrollo. Del análisis aplicado al presente proyecto resulta que:

- Costo en tiempo: El desarrollo del proyecto implica la inversión de tiempo de los recursos que actualmente laboran en el área de desarrollo.
- Costo de adquisición de nuevos recursos: No se requiere la adquisición de hardware o software adicional al que se cuenta actualmente.
- Costo de realización: El proyecto puede llevarse a cabo con la organización de los tiempos del personal de desarrollo, planeando las actividades y distribuyendo el esfuerzo entre las diversas actividades del área.

Factibilidad operativa.

Se refiere a el hecho de que el sistema una vez implantado será operado y utilizado, con lo que se evita invertir en desarrollos que no aporten algo a la organización. Para el presente proyecto:

- La rotación del personal en el área determina que el proyecto será utilizado constantemente en su aspecto de introducción a la tecnología, mientras, al ser también referencia para el desarrollo, se empleará ocasionalmente para resolver dudas y conflictos que surjan en medio del proceso de desarrollo.

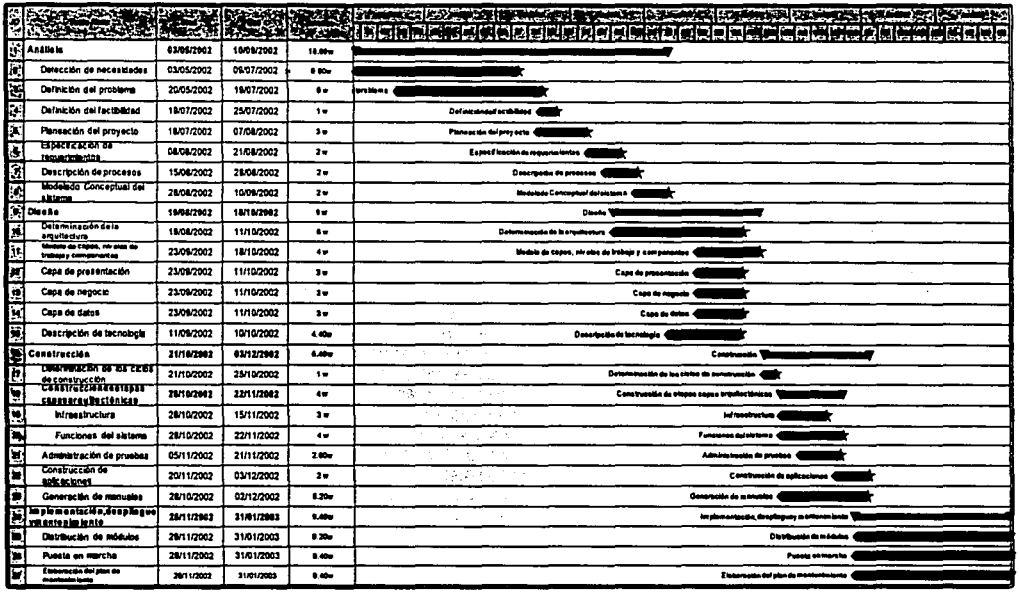
Planeación del proyecto.

En esta etapa se delimita el problema identificado, justificando el planteamiento de los objetivos desarrollados inicialmente, además, se definen las etapas o niveles de desarrollo así como las técnicas y el control que se aplicará.

El proceso de planeación busca concretizar los resultados del análisis en un plan de trabajo más detallado siguiendo el siguiente proceso, generando los siguientes documentos, que se integran como parte de este proyecto de titulación:

- Antecedentes
- Determinación de alcances y objetivos.
- Delimitación del problema.
- Justificación.
- Definición de las etapas de desarrollo.

Los resultados de esta fase se ven reflejados en toda la estructura del proyecto, para mayor detalle referirse a los apartados correspondientes a antecedentes, objetivos, definición del problema, justificación y metodología.



1	Análisis	03/05/2002	19/09/2002	18.00w
2	Detección de necesidades	03/05/2002	05/07/2002	8.80w
3	Definición del problema	20/05/2002	19/07/2002	9 w
4	Definición del factibilidad	18/07/2002	25/07/2002	1 w
5	Planeación del proyecto	18/07/2002	07/08/2002	3 w
6	Especificación de requerimientos	08/08/2002	21/08/2002	2 w
7	Descripción de procesos	15/08/2002	28/08/2002	2 w
8	Modelado Conceptual del sistema	28/08/2002	10/09/2002	2 w
9	Diseño	19/08/2002	18/10/2002	9 w
10	Determinación de la arquitectura	19/08/2002	11/10/2002	8 w
11	Modelo de capas, niveles de trabajo y componentes	23/09/2002	18/10/2002	4 w
12	Capa de presentación	23/09/2002	11/10/2002	3 w
13	Capa de negocio	23/09/2002	11/10/2002	3 w
14	Capa de datos	23/09/2002	11/10/2002	3 w
15	Descripción de tecnología	11/09/2002	10/10/2002	4.40w
16	Construcción	21/10/2002	03/12/2002	6.40w
17	Determinación de los ciclos de construcción	21/10/2002	28/10/2002	1 w
18	Construcción de etapas capas arquitectónicas	28/10/2002	22/11/2002	4 w
19	Infraestructura	28/10/2002	15/11/2002	3 w
20	Funciones del sistema	28/10/2002	22/11/2002	4 w
21	Administración de pruebas	05/11/2002	21/11/2002	2.60w
22	Construcción de aplicaciones	20/11/2002	03/12/2002	2 w
23	Generación de manuales	28/10/2002	02/12/2002	5.20w
24	Implementación, despliegue y mantenimiento	28/11/2002	31/01/2003	9.40w
25	Distribución de módulos	28/11/2002	31/01/2003	9.20w
26	Puesta en marcha	28/11/2002	31/01/2003	9.40w
27	Elaboración del plan de mantenimiento	28/11/2002	31/01/2003	9.40w

Ilustración 20: Gráfica deGantt

Especificación de requerimientos.

Un proyecto no puede ser exitoso sin el conocimiento exhaustivo de los requerimientos. Para ello se requieren habilidades que rebasan el objetivo de este documento, enfocado a referencias básicas del proceso de desarrollo. Este apartado describe los requerimientos principales identificados en este proyecto, con la finalidad de que el lector pueda tener un sustento práctico al expresar los requerimientos, por lo que la lista presentada no debe considerarse exhaustiva sino representativa.

Los requerimientos son una descripción de las necesidades identificadas en el producto final proyectado. La primera meta es identificar lo que el sistema realmente necesita y expresarlo de forma clara. Se recomienda desarrollar los siguientes artefactos:

Artefacto.	Descripción.
Panorama general.	Este proyecto tiene por objetivo crear un sistema que ejemplifique el desarrollo de aplicaciones distribuidas y que sirva de sustento bibliográfico a futuros desarrollos. Se ha elegido un videojuego en línea por las características identificadas en este tipo de proyectos.
Clientes.	TVUNAM, área de desarrollo de sistemas. Becarios, servidores sociales y trabajadores del área. Personal relacionado con la creación y mantenimiento de software en la institución.
Metas.	Desarrollar un videojuego en línea aprovechando las capacidades del desarrollo multicapa y de las tecnologías y estandares J2EE. Documentar el proceso de desarrollo y alimentarlo con referencias especificas de apoyo a la creación de proyectos.
Funciones del sistema.	Se describen las principales actividades del sistema, organizándolas para consulta y categorizandolas. *Ver apartado Funciones del sistema.*
Atributos del sistema.	Se describen las características o dimensiones del sistema (no son funciones). *Ver apartado atributos del sistema.*

Funciones básicas del sistema.

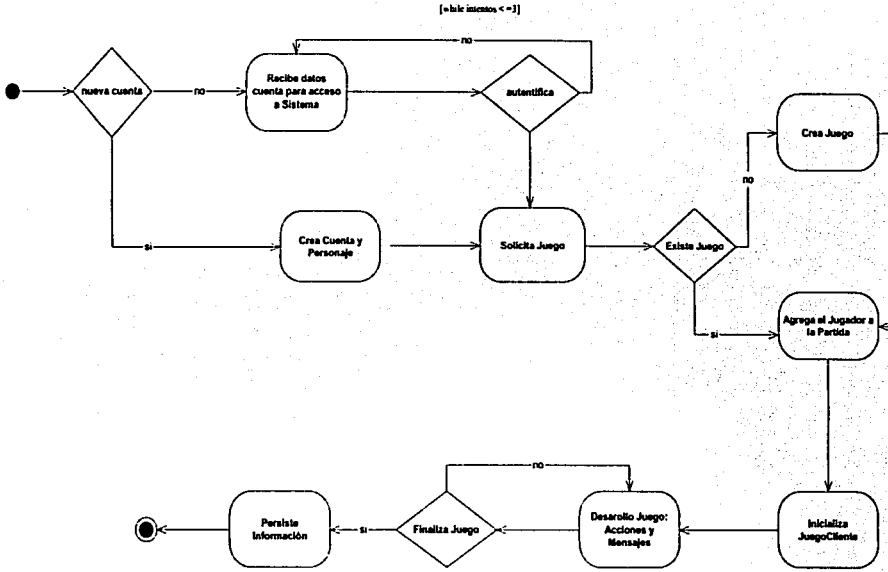


Ilustración 21 : Funciones básicas del Sistema

Funciones Básicas del Sistema.

El siguiente listado muestra las funciones básicas inmersas en el producto final. El estudio exhaustivo de las mismas se omite de este documento para facilitar la comprensión de los temas expuestos. Nuestro objetivo es entender los detalles del análisis no el funcionamiento específico de un videojuego en línea. La forma de analizar la compilación completa de funciones es observando los servicios descritos más adelante y que fueron generados a partir de estas funciones básicas.

Referencia: Clave de identificación o consecutivo para localizar de forma rápida una función.
Función: Nombre de la función.
Resumen: Descripción abreviada de las actividades de la función.
Categoría: Describe si es transparente al usuario (Oculta) o es visible por él (Evidente).

Referencia.	Función	Resumen.	Categoría
F.1	Registro de usuarios.	*Se crea una cuenta para el nuevo usuario, con los datos que él proporciona.*	Evidente
F.2	Creación de personajes.	*Se crea un personaje con los datos que el usuario aporta y se asocia a su cuenta.*	Evidente
F.3	Creación del juego.	*Se extrae la información de la base de datos y se generan todos los aspectos del juego.*	Oculta
F.4	Acceso al sistema.	*Se reciben y validan los datos de acceso ingresados por el usuario. Se inicializan los valores de la aplicación cliente.*	Evidente
F.5	Generación de acciones de juego.	*Se reciben y validad los comandos generados por el cliente.*	Evidente
F.6	Gestion de acciones.	*Se llevan a cabo las acciones efecto de los comandos del cliente, afectando al juego y su representación.*	Oculta
F.7	Generación de mensajes de comunicación.	*Se reciben y validan los mensajes de comunicación textual del cliente.*	Evidente
F.8	Gestion de mensajeria.	*Se administran los mensajes de comunicación textual de los clientes.*	Oculta
F.9	Salida del sistema.	*Se cierra la conexión con el usuario, se actualizan los valores del juego.*	Evidente
F.10	Persistencia de datos de juego.	*Se actualizan los valores que requieren persistencia en base de datos.*	Oculta

Atributos del sistema.

El siguiente cuadro muestra un listado de características y dimensiones deseables del sistema. Como en el caso de las funciones, solo se incluyen los aspectos básicos, favoreciendo la comprensión de los temas tratados en este proyecto. Los atributos del sistema pueden abarcar a todas las funciones o ser específicos de una función o grupo de funciones. Los atributos tienen un conjunto de detalles que tienden a ser discretos o simbólicos, así como valores de restricción frontera obligatorios, que son condiciones máximas para cada valor.

Atributo	Detalles y restricciones de frontera.
facilidad de uso.	(detalle) El usuario debe tener múltiples opciones para realizar acciones, todos los elementos deben poder identificarse adecuadamente, se debe incluir cuadros de ayuda e información. (detalles) Los conceptos manejados deben ser claros y requerir poco esfuerzo de comprensión para poder enfocar la atención a los factores de la tecnología, la metodología y estándares.
comunicación.	(detalle) Comunicación en línea de los clientes con el servidor siguiendo un esquema de arquitectura distribuida, se requiere una conexión a red para jugar. (detalle) Estructura multijugador, varios clientes pueden conectarse simultáneamente al juego. (restricción) El número máximo de jugadores para el prototipo inicial es 4 por juego creado.
tolerancia a fallas.	(detalle) En caso de caída del sistema debe ser posible recuperar el estado del juego. (detalle) En caso de desconexión involuntaria el jugador puede reingresar al juego con el estado que tenia anteriormente. (restricción) Se resguardan solo los datos fundamentales del juego.
tiempo de respuesta.	(detalle) Se debe contar con una rápida respuesta a las acciones, simulando el tiempo real, introduciendo mecanismos que optimicen el acceso a información. (detalle) Los tiempos de respuesta están condicionados a las características de los equipos cliente y servidor así como a el medio de comunicación. Las restricciones en tiempos de respuesta se generan a partir de los requisitos técnicos medios. (restricción) De cinco a sesenta segundos para la carga del juego. (restricción) De cero a cinco segundos para actualizaciones. (restricción) De cero a cinco segundos para respuesta a acciones.
plataformas de despliegue.	(detalle) Cualquier servidor de aplicaciones, en el servidor.
plataformas de operación.	(detalle) Cualquier sistema operativo.
metáfora de la interfaz	(detalle) Ventanas separadas para registro, creación de personajes, acceso al sistema y desarrollo del juego. (detalle) Aplicación integrada para el desarrollo del juego, una sola ventana con apartados interrelacionados: tablero, visor, comandos, mensajería. Manejo de cuadros de diálogo para errores y ayuda. (detalle) Fácil navegación entre pantallas y sus vistas. Enfoque gráfico de la aplicación. (detalle) Componentes para generar acciones de forma alterna al uso del apuntador.

Descripción de procesos.

En esta actividad se generan los casos de uso que nos ayudan en gran medida a la comprensión de los requerimientos del sistema. La notación UML incluye formalmente el concepto de casos de uso, así como los artefactos y diagramas que emplea.

Un caso de uso es un documento narrativo que describe la secuencia de eventos que un actor (agente externo al sistema) utiliza para completar un proceso. Los casos de uso requieren el conocimiento de los requerimientos y atributos del sistema, ya que los ejemplifican e incluyen tácitamente en su estructura.

Cabe resaltar que los casos de uso son descripciones de los procesos de principio a fin, y suelen abarcar muchos pasos o transacciones, no son actividades o pasos individuales del proceso.

Un diagrama de casos de uso explica gráficamente a un conjunto de casos de uso, representando los procesos y a los actores que intervienen así como la forma en que se relacionan. Los procesos (casos de uso) son óvalos, mientras los actores son figuras estilizadas, las flechas indican el flujo de la información. Estos diagramas tienen por objeto ofrecer una clase de esquema conceptual que nos permite conocer las formas básicas de uso del sistema.

A continuación se describen los procesos principales involucrados en el presente proyecto.

Actores del sistema y casos de uso principales.

Actor	Descripción
Cliente	Representa al jugador.
Servidor	Representa los servicios del juego.
AdministradorBaseDatos	Representa la persistencia de la información.

Casos de uso.

Caso de uso: Registro de usuario y creación de personajes.

Caso de uso:	*Registro de usuarios y creación de personajes.*
Actores:	*Cliente, Servidor, AdministradorBaseDatos.*
Propósito:	*Crear cuentas para jugadores y asociarles un personaje.*
Resumen:	*El jugador requiere tener una cuenta en el servidor para hacer posible el acceso al juego. El cliente indica que desea crear una nueva cuenta, se le presenta el formulario de registro en el que ingresa sus datos. El Servidor crea la cuenta y el personaje, y solicita al AdministradorBaseDatos la creación d e los nuevos registros asociados.*
Referencias:	*Funciones: F. 1, F.2* *Caso de uso: Persistencia de datos de juego.*

Curso normal de eventos.

Cliente	*Servidor*	*AdministradorBaseDatos*
1.- Este caso de uso comienza cuando el Cliente solicita la creación de una cuenta.		
2.- El cliente ingresa, valida y envía la información que se le solicita para la creación de cuenta y personaje.	3.- El servidor recibe y valida la información. 4.- El servidor genera la nueva cuenta y el personaje asociado a ella. 5.- El servidor solicita el registro de la cuenta y personaje en la case de datos.	6.- El AdministradorBaseDatos ingresa los datos en la base de datos.
8.- El Cliente recibe la información, con lo que concluye este caso de uso.	7.- El servidor envía al Cliente la información de la cuenta y personaje.	

Cursos alternos.

Acciones 2 y 3: La validación de los datos falló, ya existe la cuenta o los tipos no coinciden: Indicar el error y reiniciar el proceso de creación. Ir a Acción 1.

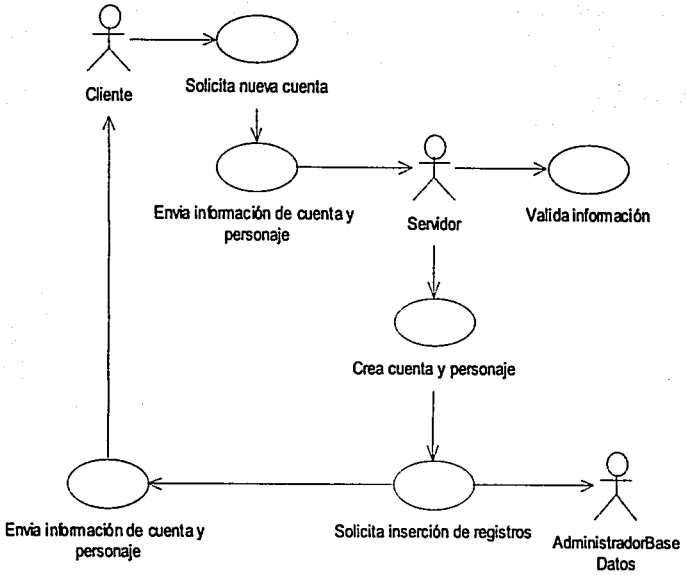


Ilustración 22 : Caso de uso Creación de cuenta y de personaje.

Caso de uso: Creación del juego.

Caso de uso:	*Creación del juego*
Actores:	*Cliente, Servidor, AdministradoBaseDatos*
Propósito:	*Construir un nuevo juego con la información leída de la base de datos.*
Resumen:	*Al recibir la primera solicitud de acceso al juego de un Cliente, el Servidor solicita al AdministradorBaseDatos la información que requiere la creación de un juego, organiza y realiza cálculos y validaciones necesarios para formar una imagen del juego. El juego queda en memoria para su uso.*
Referencias:	*Función: F.3* *Caso de uso: Acceso al sistema.*

Curso normal de eventos.

Cliente	*Servidor*	*AdministradorBaseDatos*
1.- El caso de uso comienza cuando el Cliente solicita acceso al sistema al Servidor.		
	2.-Si no existe un juego creado el Servidor solicita al AdministradorBaseDatos la información para construir un nuevo juego.	
		3.- El AdministradorBaseDatos recibe la solicitud y realiza las consultas para regresar al Servidor la información.
	4.-El servidor crea la imagen del juego, inicializando todos los valores con la información recibida. 5.- El servidor agrega al Cliente al juego. 6.- El Servidor envía al Cliente la información del juego.	
7.-El Cliente recibe la información del juego con lo que concluye el caso de uso.		

Cursos alternos.

Acción 2: Ya existe un juego creado: Utilizar los datos del juego existente, ir directamente a la acción 5 del servidor.

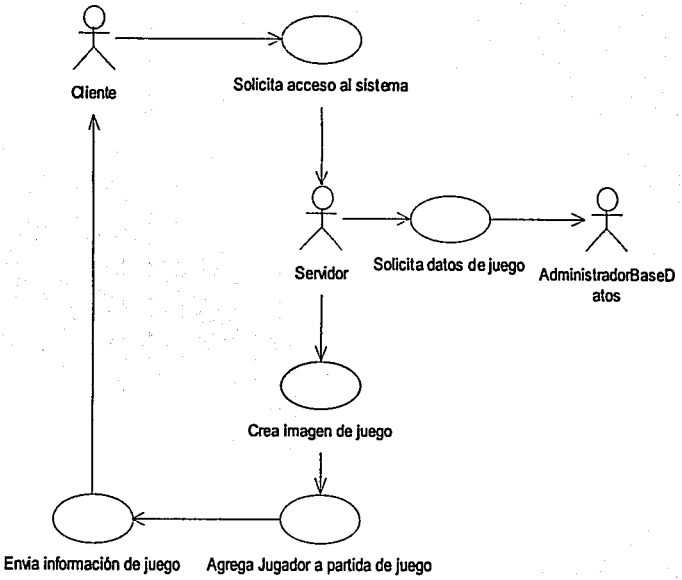


Ilustración 23 : Creación del Juego.

Caso de uso: Acceso al sistema.

Caso de uso:	*Acceso al sistema*
Actores:	*Cliente, Servidor, AdministradorBaseDatos*
Propósito:	*Autentificar al cliente como usuario válido del sistema.*
Resumen:	*Un cliente, al abrir la aplicación del juego, ingresa su nombre de usuario y contraseña, si la información corresponde a lasolicitada al AdministradorBaseDatos, se le permite el acceso al juego, registrando al personaje del Cliente en el juego y enviándole la información del mismo.*
Referencias:	*Funcion: F.4* *Caso de uso: Creación del juego.*

Curso normal de eventos.

Cliente	*Servidor*	*AdministradorBaseDatos*
1.- El caso de uso comienza cuando el Cliente ingresa nombre de usuario y contraseña como inicio del proceso de autentificación, los valida y envía al Servidor.	2.- El Servidor recibe la información, y solicita al AdministradorBaseDatos el registro correspondiente a los datos, para cotejar la identidad del Cliente.	3.- El AdministradorBaseDatos Recibe la solicitud, realiza la consulta y envía la información al Servidor.
	4.- El servidor autentifica al Cliente 5.-El servidor obtiene la información del juego activo. 6.- El servidor agrega al personaje al juego. 7.- El Servidor envía la información del juego al Cliente.	
8.- El Cliente recibe la información del juego, con lo que concluye el caso de uso.		

Cursos alternos.

Acción 3: Fallo en la autentificación del cliente, datos incorrectos, no existe usuario o contraseña no corresponde: Reiniciar el proceso de acceso. Ir a Acción 1 del cliente.
Acción 4: No existe un juego activo: Crear un nuevo juego. Ver caso de uso Creación del juego.

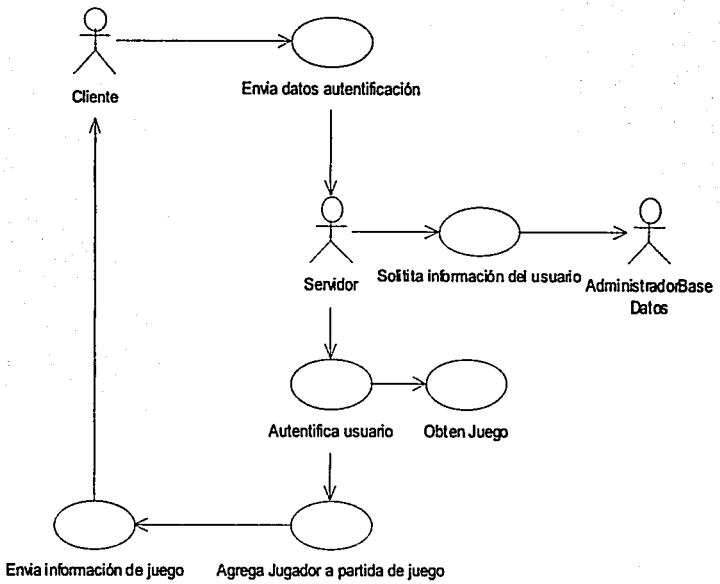


Ilustración 24 : Caso de uso Acceso al sistema.

Caso de uso: Gestión de actividades de juego.

Caso de uso:	*Generación de acciones de juego*
Actores:	*Cliente, Servidor, AdministradorBaseDatos*
Propósito:	*Permitir al jugador la realización de acción de juego.*
Resumen:	*El Cliente solicita, por medio de su interfaz la realización de una acción relacionada con un comando, mismo que es validado y estructurado en una solicitud al servidor. El Servidor recibe la solicitud del cliente y realiza las modificaciones necesarias tanto en la imagen de juego como en la base de datos para lo que envía, en caso de ser necesario, una solicitud al AdministradorBaseDatos.*
Referencias:	*Funciones F.5, F.6* *Caso de uso: Persistencia de datos de juego.*

Curso normal de eventos.

Cliente	*Servidor*	*AdministradorBaseDatos*
1.-El caso de uso comienza cuando el Cliente genera una acción. 2.- El cliente valida la estructura, origen y destino de la acción para generar un comando. 3.-El Cliente solicita al Servidor la realización de la acción sobre el juego.	4.- El Servidor recibe la solicitud del Cliente y valida la acción en términos de el estado actual del juego. 5.- El Servidor realiza las modificaciones al juego y, de ser necesario, solicita al AdministradorBaseDatos que actualice los registros de la base de datos.	6.-El AdministradorBaseDatos recibe la solicitud del Cliente y realiza las actualizaciones, confirmando la operación.
8.- El Cliente recibe la información del juego actualizada con lo que concluye el caso de uso.	7.- El Servidor envía la información modificada del juego al Cliente.	

Cursos alternos.

Acciónes 3 y 4: Fallo en la validación de la acción, estructura incorrecta del comando, faltan atributos, no se encuentra el origen u objetivo. Se notifica el error .
Acción 5: Se requiere persistir información en la base de datos. Ver caso de uso: Persistencia de datos de juego.

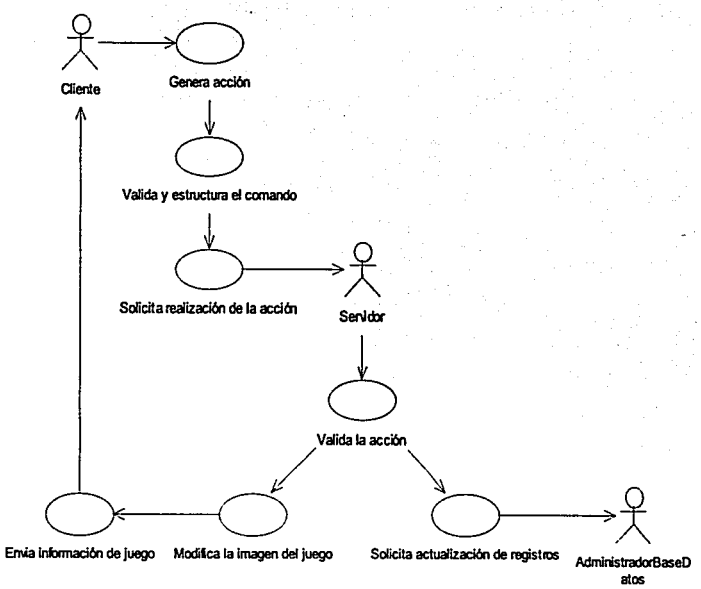


Ilustración 25 : Caso de uso Gestión de actividades del juego.

Caso de uso: Gestión de mensajes de comunicación.

Caso de uso:	*Gestión de mensajes de comunicación*
Actores:	*Cliente, Servidor*
Propósito:	*Permitir al jugador el envío y recepción de mensajes de comunicación.*
Resumen:	*El Cliente envía un mensaje de comunicación al Servidor. El Servidor recibe el mensaje del cliente, lo procesa y reenvía a los diferentes clientes conectados al Servidor. El Cliente discrimina que mensajes desplegar.*
Referencias:	*Funciones: F.7, F.8* *Caso de uso: Gestión de acciones de juego.*
Curso normal de eventos.	
Cliente	***Servidor***
1.-El caso de uso comienza cuando el Cliente genera un mensaje nuevo. 2.-El Cliente gestiona el tipo de mensaje. 3.-Si se trata de un mensaje de comunicación el Cliente envía el mensaje al Servidor. 6.-El Cliente recibe el mensaje con lo que concluye el caso de uso	4.-El Servidor recibe el mensaje y lo procesa agregándolo a la cola de mensajes. 5.-El Servidor distribuye y canaliza los mensajes a los clientes.
Cursos alternos.	
*Acción 2 :El mensaje gestionado es un comando escrito: S*e genera un comando. Ver caso de Uso Gestión de Acciones de Juego	

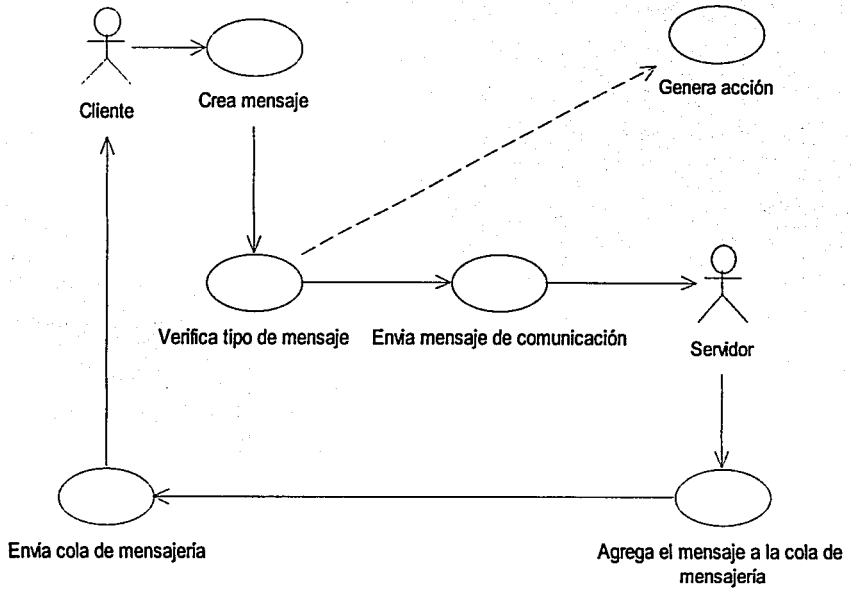


Ilustración 26 : Caso de uso Gestión de mensajes de comunicación

Caso de uso: Salida del sistema.

Caso de uso:	*Salida del sistema.*
Actores:	*Cliente, Servidor*
Propósito:	*Permitir el cierre de la conexión con el cliente, actualizando el juego.*
Resumen:	*El Cliente decide finalizar el Juego. El Servidor saca de la partida al personaje del Cliente, afectando la imagen del Juego. Se elimina la conexión con el Cliente.*
Referencias:	*Función: F.9*
Curso normal de eventos.	
Cliente	***Servidor***
1.-El caso de uso comienza cuando el Cliente decide finalizar el Juego y envia la solicitud al Servidor	
	2.-El Servidor recibe la solicitud, y afecta la imagen del Juego.
3.-El cliente se desconecta con lo que concluye el caso de uso.	
Cursos alternos.	
Acción 4: El Cliente es el único jugador activo en la partida: Finaliza y elimina el Juego.	

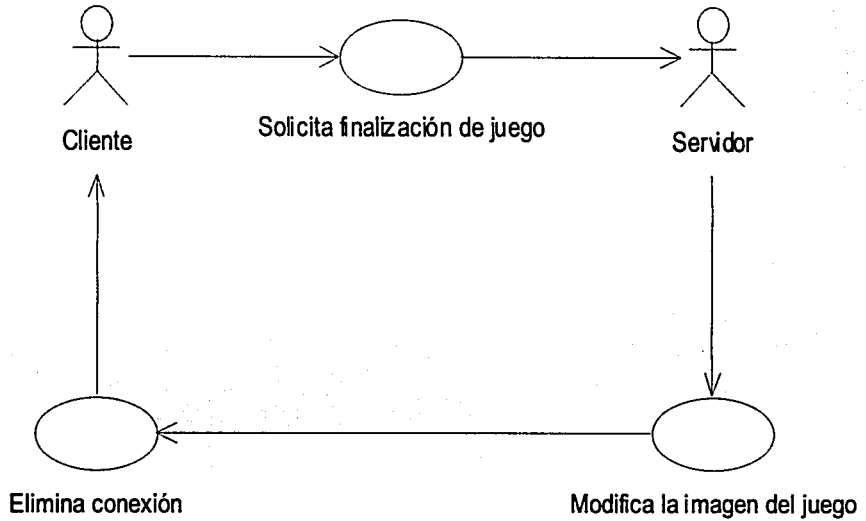


Ilustración 27 : Caso de Uso Salida del Sistema

Caso de uso: Persistencia de datos de juego.

Caso de uso:	*Persistencia de datos de juego.*
Actores:	*Servidor, AdministradorBaseDatos*
Propósito:	*Almacenar en la fuente de información los datos que no dependen enteramente del juego.*
Resumen:	*Existe información que no tiene carácter volátil, sino que es persistida pues tiene significado más allá del juego activo. El Servidor identifica modificaciones a esta información, solicita al AdministradorBaseDatos el registro, actualización o borrado de registros, a la vez que actualiza la información de la imagen del juego.*
Referencias:	*Función: F.10*
Curso normal de eventos.	
Servidor	***AdministradorBaseDatos***
1.- El caso de uso comienza cuando el Servidor decide que se requiere persistir un dato en la base de datos. 2.-El Servidor valida la información a persistir. 3.-El Servidor valida la acción de persistencia a realizar. 4.- El Servidor solicita al AdministradorBaseDatos la actualización de registros.	5.- El AdministradorBaseDatos realiza los cambios en la base de datos y confirma la operación.
6.- El servidor modifica la imagen del juego, con lo que concluye el caso de uso.	
Cursos alternos.	
Accion 2y 3: Fallo en la validación de la información. Los tipos no coinciden, falta información. Se cancela la operación (Rollback). Se informa el error. *Acción 5: No se confirma la operación.* Se cancela la operación. Se informa el error.	

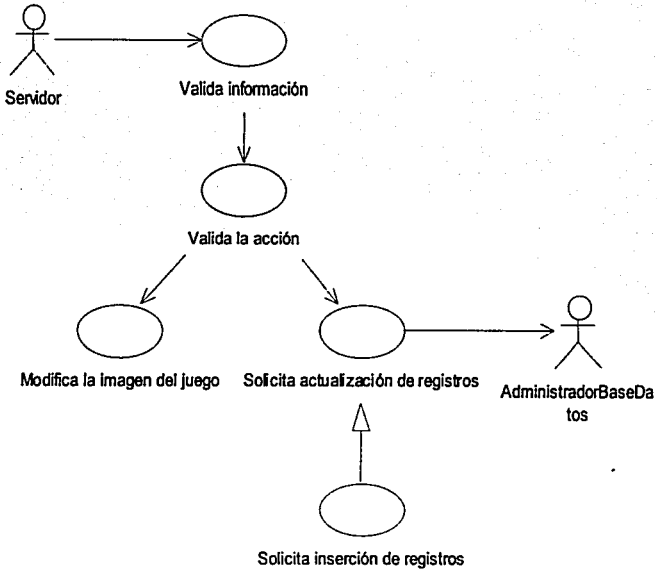


Ilustración 28 : Caso de Uso Persistencia de datos en el sistema.

Modelado Conceptual del sistema.

Un modelo conceptual explica a los desarrolladores los conceptos significativos en un dominio del problema. Los casos de uso son artefactos muy importantes para el análisis de requerimientos, pero no están realmente orientados a objetos, es en esta actividad en donde realmente son identificados los objetos o conceptos que constituyen la esencia del análisis orientado a objetos.

Como punto inicial se descompone el problema en conceptos u objetos individuales, para lo que se puede emplear una lista de conceptos por categoría (lugares, actores, transacciones, eventos, etcétera). Dichos conceptos son representados en un diagrama de modelo conceptual (especificado por UML). Posteriormente se incorporan las asociaciones necesarias y se agregan atributos fundamentales para completar el modelado.

Se presenta el modelo conceptual del caso práctico de este proyecto:

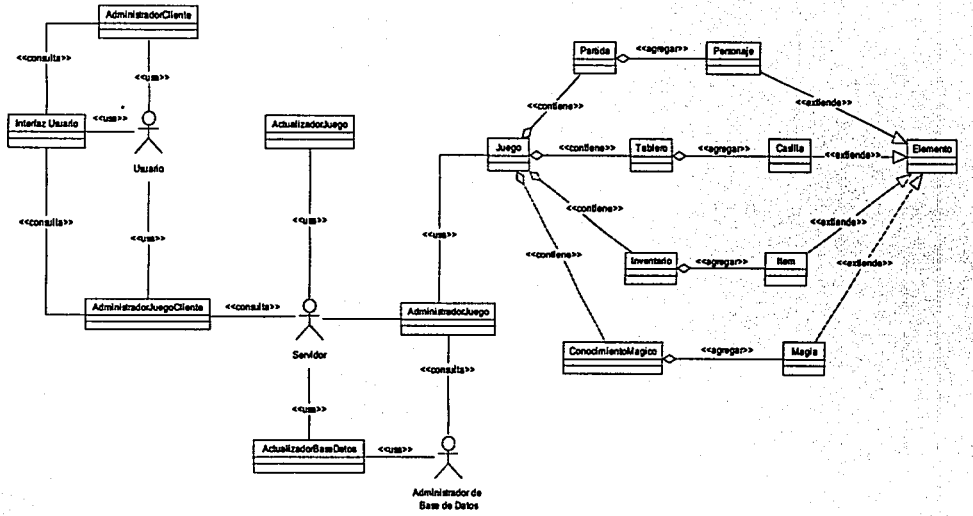


Ilustración 29 : Modelo conceptual del proyecto.

Descripción de la arquitectura del proyecto.

Se define a la arquitectura como el conjunto de reglas, definiciones, términos y modelos que se emplean para construir un producto. Es la estructura de alto nivel de los subsistemas y de los componentes así como de sus interfaces, conexiones e interacciones. Se compone de un conjunto de esquemas, subsistemas, clases asignaciones de responsabilidad y colaboración entre objetos que cumplen con las funciones del sistema.

Así como es importante el estudio y análisis detallado de la problemática, un punto que vale la pena analizar es lo referente a la arquitectura del proyecto, ya que la intención debe ser no solo que satisfaga las necesidades presentes de forma óptima, sino que , además, presente la flexibilidad suficiente para poder hacer frente a cambios futuros en los requerimientos.

Por lo anterior es conveniente encontrar la respuesta a las siguientes preguntas, tratando de construir un modelo eficiente a la vez que flexible:

¿Cual es la mejor arquitectura para el sistema?
¿Cual es la mejor herramienta para desarrollar lo que pide el cliente?
¿Cómo diseñamos la Base de Datos?
¿Cómo diseñamos las Clases Fundamentales?

Un proceso eficaz estimula la generación temprana de una arquitectura global del sistema, es decir, esta centrado en la arquitectura.

Una arquitectura bien diseñada reúne las siguientes características:

- Incluye subsistemas y componentes sustentados en arquitectura de capas.
- Ofrece bajo acoplamiento entre sus sistemas.
- Es de fácil comprensión.
- Es robusta, flexible y se incrementa de forma adecuada.
- Maneja componentes reutilizables.
- Se orienta a las funciones del negocio importantes.

Se ha seleccionado para el presente proyecto el desarrollo de un sistema de estructura distribuida, donde los servicios que los clientes requieren, se concentran en los objetos del servidor, y son accesibles por medio de llamadas remotas al mismo. Así, los clientes tiene la funcionalidad necesaria para manejar y presentar la información que reciben del servidor, mientras el servidor concentra los servicios comunes a todos los clientes, tales como consultas y actualizaciones a la información que contiene.

Ventajas de los sistemas distribuidos:

- Permiten compartir los recursos.
- Permiten concurrencia en los servicios que ofrecen.
- Proveen capacidades de escalabilidad a los sistemas.
- Observan la tolerancia de fallas
- Aportan transparencia a las operaciones.

Desventajas de los sistemas distribuidos:

- Aumentan la complejidad en el desarrollo y manejo de los sistemas.
- Requieren mecanismos adicionales para el manejo de seguridad.
- La complejidad en los procesos y la concurrencia aumentan la impredictibilidad del sistema.

Particularmente, el caso práctico que servirá de guía en la lectura de este documento es un videojuego en línea distribuido, las aplicaciones cliente son las interfaces de interacción de los jugadores y el servidor tiene la responsabilidad de mantener actualizado el estado del juego, en respuesta a las acciones realizadas, a la vez que pone ese estado a disponibilidad de consulta de cada cliente.

Todas las acciones realizadas a través del cliente implican una solicitud al servidor, misma que puede ser una consulta de información o una modificación a algún aspecto del juego. El servidor recibe cada solicitud y realiza las actualizaciones necesarias, después de lo cual, cada cliente es responsable de solicitar la versión actualizada del juego, renovando, a su vez, la información presentada al jugador.

Es evidente que este sistema distribuido es un esquema cliente-servidor, donde el servidor pone a disposición una serie de servicios que son comunes a todos los clientes, y que permiten la interacción remota de jugadores ubicados en diferentes puntos en la red.

Para el análisis de la arquitectura distribuida se deben tomar en cuenta los siguientes factores:

Identificación de recursos.

En un sistema distribuido, los recursos están organizados y extendidos en diversas computadoras, pero aún así debe contarse con un mecanismo para poderlos localizar desde cualquier punto del sistema, es aquí donde entran los sistemas de nombramiento (o servicio de nombres), mismo que tiene que debe establecerse para que los usuarios puedan descubrir y referirse a los recursos que necesitan.

Para cumplir con estas funciones el estándar J2EE integra JNDI (Java Naming and Directory Interface) que es una interfaz abstracta que provee servicios de búsqueda de nombres y directorios relacionados a recursos del sistema.
Si no se usa un sistema significativo y entendido universalmente, se tendrá como resultado que algunos recursos queden inaccesibles a los usuarios del sistema.

Comunicaciones

Para la mayoría de los sistemas distribuidos la manera más eficaz de comunicación se ve reflejada en los protocolos TCP/IP de comunicación, gracias a la disponibilidad de Internet. Sin embargo, requerimientos específicos nos llevan a usar enfoques alternativos de comunicación

Para comunicación de objetos distribuidos J2EE cuenta con la tecnología RMI-IIOP(Remote Method Invocation- Internet Inter Orb Protocol), que proporciona servicios de invocación remota de métodos. Además de que CORBA, estándar internacional de un Object Request Broker (middleware para manejar las

comunicaciones entre objetos distribuidos), sirve de complemento para proporcionar un marco de trabajo de objetos distribuidos, servicios para soportar ese marco de trabajo e interoperabilidad con otros lenguajes.

Calidad del servicio

La calidad del servicio ofrecida por un sistema se ve reflejada en su rendimiento, disponibilidad y fiabilidad. Algunos factores que afectan la calidad son la asignación de procesos, la distribución de recursos a través del sistema, la red y el hardware del sistema y la capacidad del sistema para adaptarse a cambios.

Arquitecturas de software

La arquitectura del software describe cómo la funcionalidad de la aplicación es distribuida en componentes lógicos y cómo éstos de organizan en diversos procesadores. Escoger la arquitectura correcta para una aplicación es imprescindible para alcanzar el nivel deseado del servicio.

Desarrollo de aplicaciones multicapa

Desarrollar un servicio multi-capa requiere aplicaciones cliente, lógica de negocio, acceso a bases de datos y código de infraestrcutura. La infraestructura son componentes de bajo nivel del sistema que proporciona acceso a bases de datos, y recursos del sistema y proporcionan seguridad y transacciones. Los detalles de la infraestructura son manejados por J2EE para que el desarrollador sólo necesite desarrollar las lógicas de negocio, presentación y datos.

Las capas de negocio y datos se implementan como componentes EJB, mientras la capa de presentación permite utilizar diversas tecnologías, en nuestro caso, aplicaciones Java. Para crear aplicaciones EJB, en primer lugar debe crear y desplegar los enterprise beans, para, posteriormente, construir los clientes EJB que encapsulan los datos de negocio y la funcionalidad, y combinarlos del modo adecuado. Se crean aplicaciones EJB combinando uno o más beans de sesión, uno o más beans de entidad, o ambos. Aunque en un cliente EJB puede utilizar directamente beans de sesión y de entidad de modo individual, los beans de sesión se han diseñado para asociarlos con clientes y los beans de entidad se han diseñado para almacenar datos persistentes, por lo que la mayor parte de las aplicaciones EJB contienen beans de sesión que, a su vez, acceden a los beans de entidad.

Modelo de tres capas del proyecto: adopción de estándares J2EE.

En los modelos clásicos cliente – servidor se manejan normalmente dos capas o niveles de trabajo, una encargada de hacer posibles las solicitudes del cliente (interfaz al usuario)y otra encargada de dar respuesta a dichas solicitudes (reglas del negocio, acceso a bases de datos) estableciendo un middleware que controla las comunicaciones entre ambos.

Sin embargo, los servidores de aplicaciones, como el que empleamos para el despliegue de aplicaciones en la organización, implementan un modelo de tres capas, considerando la separación física entre el cliente que solicita los servicios, las aplicaciones o programas que procesan las solicitudes del cliente y los datos sobre los que se operan., con lo que se facilita la administración de las llamadas reglas del negocio, así como el manejo de seguridad y transacciones.

Ilustración 30 : Capas del negocio.

- La capa de presentación
  - Involucra recoger las entradas del usuario y presentar los resultados de una computación de los mismos, realizada por la capa de negocio.
- La capa de negocio
  - Involucra proveer la funcionalidad específica del sistema y reglas del negocio.
- La capa de acceso a datos
  - Involucra la administración de la información en los medios de almacenamiento (típicamente bases de datos) del sistema.

La arquitectura J2EE (Java 2 Enterprise Edition), esta basada en componentes para la programación multi-capa, resolviendo el problema del costo y la complejidad del desarrollo de servicios escalables, de alta disponibilidad, seguros y eficientes, al proporcionar una arquitectura de estándar abierto a través de la Plataforma J2EE y del modelo de Aplicación J2EE. Esta plataforma permite a los desarrolladores enfocarse en la lógica de negocio mientras que J2EE maneja los detalles de bajo nivel. Así, los servicios son fácilmente mejorables y rápidamente desarrollados, permitiendo a los negocios reaccionar rápidamente a cambios en los requerimientos.

El servidor de aplicaciones, basado en J2EE, ejecuta los programas de negocio en lugar del cliente, situándose éste y los datos empresariales y otras aplicaciones. Cuando el cliente necesita hacer una petición se la hace a la capa en la que se encuentra la lógica del negocio, permitiendo desarrollar y desplegar aplicaciones rápida y fácilmente e incrementar la cantidad de usuarios del sistema sin necesidad de reprogramación.

El lenguaje Java, la máquina virtual Java y los componentes JavaBaens son la base de J2EE, sin embargo también esta compuesto de diferentes componentes:

- **Servlets**— un reemplazo eficiente, independiente de la plataforma para los scripts CGI que responden a solicitudes de clientes.
- **JavaServer Pages (JSP)**— un tipo de script del lado del sevidor, que puede generar páginas web dinámicamente.
- **Enterprise JavaBeans (EJB)**— control de sesión del lado del servidor, que encapsula la lógica de negocios y abstracción para acceder a datos persistentes.
- **Java Database Connectivity (JDBC)**— un API que describe una librería estándard Java para acceder a fuentes de datos.
- **Transaction Support**— transaciones declarativas para componentes donde las transaciones pueden expandir componentes y procesos.
- **Java Naming and Directory Interface (JNDI)**— un interface abstracto para servicos de búsqueda de uniones de nombres y directorios.
- **Remote Method Invocation (RM/IIOP)**— una tecnología que permite la comunicación entre objetos distribuidos.
- **CORBA Compatible**— CORBA complementa Java proporcionando un marco de trabajo de objetos distribuidos, servicios para soportar ese marco de trabajo e interoperabilidad con otros lenguajes.
- **Java Messanging Service (JMS).** Aplicación de mensajería empresarial.
- **eXtensible Markup Language (XML).** Permite definir configuración estructurada y envío de mensajes.

Las ventajas que proporciona la adopción de los estandares J2EE en este proyecto son:

- La separación de tareas en la plataforma multi-capa.
- El conjunto de estándares abiertos que componen la plataforma J2EE (EJB, JSP, Servlets, JDBC, JNDI, RMI).
- La característica de portabilidad de Java "Write Once, Run Anywhere"(WORA) **[Escribe una vez, ejecuta en cualquier parte]** que significa que los componentes diseñados para el sistema pueden ser escritos por terceras partes, reduciendo el coste del desarrollo.
- El sistema de aplicación Enterprise soporta alta escalabilidad usando una arquitectura de aplicación distribuida multi-capa con componentes integrados.
- Pueden reutilizarse la lógica del negocio y el control de acceso a los datos
- Mayor control de acceso a través de las transacciones
- Acceso a los sistemas en forma eficiente
- La lógica de presentación, la lógica de negocio, y la lógica de acceso a datos están separadas en diferentes componentes, lo que permite a una aplicación aprovecharse del alto rendimiento de los sistemas multi-proceso y multi-capa.
- Las experiencias con el lenguaje y con los APIs de Java pueden ayudarnos a tratar con otras áreas, como el desarrollo de EJBs, además de que J2EE proporciona soluciones JAVA en todas sus capas.
- Se hace posible dimensionar cada una de las capas de manera independiente, ajustándose a las necesidades de los clientes. Interfaz gráfica, lógica del negocio, y almacenamiento de datos en forma independiente.

Entre sus desventajas se cuentan:

- Las aplicaciones distribuidas invariablemente implican mayores requerimientos de hardware y software
- Estas arquitecturas son más complicadas y es una dificultad añadida a la hora de implementarlas, aumentando la curva de aprendizaje de los desarrolladores.
- La optimización de las velocidades de acceso es el gran problema de estas aplicaciones, así como el control de las comunicaciones y las notificaciones de los eventos asíncronos.
- Se requiere el aprendizaje de nuevas técnicas, modelos y métodos para el desarrollo e identificación de los objetos de negocios, implicando mayor complejidad en el desarrollo de los sistemas.

Descripción del modelo de capas o niveles de trabajo del proyecto.

A nivel funcional, y siguiendo lo expuesto, el caso práctico de este proyecto se ha dividido en tres capas:

Capa de presentación. Es utilizada para interactuar con el usuario, así como el ingreso y egreso de datos. Se encarga , además, del proceso de información para ser exhibida al cliente y de algunas validaciones de ingresos del cliente antes de solicitar servicios de negocio. Controla la interfaz de la aplicación con el usuario procesando solicitudes, generando contenido de respuesta, formateando y entregando el contenido de vuelta al cliente.

La capa de presentación del videojuego incluye las pantallas, diálogos, interfaces y métodos, por medio de los cuales se hace posible la interacción entre los jugadores y el juego, a través de una interfaz gráfica generada y actualizada a partir de la información que se envía y recibe.
Se ejecuta en el cliente una aplicación Java encargada de gestionar los eventos producidos por el jugador que pueden ser visualizados en los paneles desplegados en la ventana principal personalizada para ese jugador.

Capa de negocio. Se encarga del procesamiento funcional de información. Controla la lógica del negocio. Los EJBs permiten que la lógica de negocio sea persistente entre llamadas, normalmente mejorando el caché, y están diseñados para trabajar en conjunto con JDBC para transaciones con bases de datos.

En el videojuego, la capa de negocio provee a los jugadores de servicios comunes a todos, y que afectan al desarrollo del juego, como el cambio de posición o el aumento del inventario de un personaje. Además por medio de la capa de negocio es posible conocer el estado actual de juego así como de cada uno de sus elementos. En términos generales, la capa de negocios, recibe las acciones o consultas que los clientes desean realizar, las valida y realiza en el juego las actualizaciones que se deriven, entregando la información que se requiera.

Los servicios de negocio son provistos por beans de sesión localizados en el servidor de aplicaciones, y tienen por responsabilidad:

1. Mantener la información del juego de acuerdo a los eventos que generen todos los jugadores.
2. Realizar la validación de las acciones de los jugadores, que afecten al contexto general en el que se desarrolla el juego.
3. Gestionar la información entre los datos almacenados, los eventos del juego y las solicitudes de los jugadores.

Capa de acceso a datos. Se encarga del control de la persistencia de los datos. Controla el almacenamiento y recuperación de bases de datos. El API JDBC está disponible para todos los componentes Java, como todos los demás APIs, sin embargo, las transacciones de bases de datos en el esquema J2EE normalmente son controladas por beans de entidad.

Este nivel esta integrado en el caso práctico por una base de datos residente en un servidor de bases de datos SQL Server 7.0 y un conjunto de beans de entidad, que representan datos persistentes y gestionan los accesos y modificaciones en los datos.

A través de los beans de entidad residentes en el contenedor de EJBs del servidor de aplicaciones, esta capa permite realizar consultas del juego, y actualizaciones de los registros que requieren persistencia, tales como el estado de los personajes, la conformación del tablero de juego, y la definición de los elementos que se encuentran en el mismo.
El sistema esta preparado para el conocimiento sobre como administrar y manejar la información y datos requeridos, además de contemplar funciones de seguridad y transacciones que representen una protección a la información en cualquier fase del sistema distribuido.

Interacción entre las capas del sistema.

La aplicación Java instalada en el cliente recibe las solicitudes del jugador, valida la entrada y llama a los beans de entidad que conforman la capa de negocio. Los beans de sesión validan las solicitudes, ejecutan los procesos necesarios y fuerzan la transacción, comunicándose con la capa de acceso a datos. Los beans de entidad manejan los datos. La comunicación entre capas se hace de forma transparente empleando el estándar RMI-IIOP.

Los EJBs hacen el trabajo duro real con los datos de la aplicación y regula el proceso pero sin proporcionar presentación visible para el usuario. Los EJBs funcionan dentro del contexto de un "contenedor de beans", que sirve como enlace con el servidor de aplicaciones que los hospeda. Este contenedor proporciona los servicios estándar denotados en la especificación J2EE, además de servicios adicionales como control de fallos de beans de sesión con estado. De hecho, el contenedor puede manejar todos los accesos remotos, la seguridad, la concurrencia, el control de transacciones, y el acceso a bases de datos.

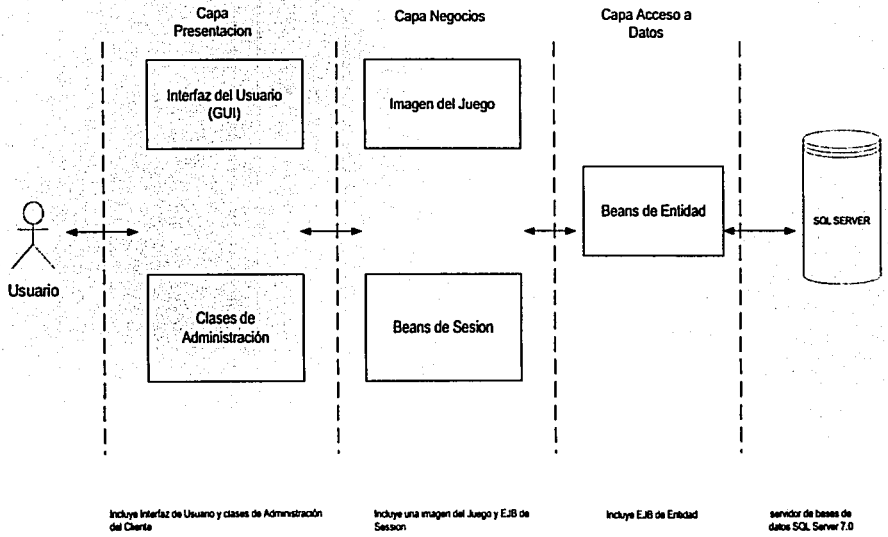


Ilustración 31 : Interacción de las capas del sistema.

Diseño de las capas del sistema.

En este apartado se detalla la estructura de las capas del sistema en función a las clases que las componen y a las responsabilidades que le son asignadas. Se incluyen las descripciones de clases generadas en la actividad de modelo de capas componentes y clases, así como la información que se anexó en etapas posteriores del diseño.

Por ser la finalidad de este documento el aprendizaje de todo el proceso de desarrollo incluyendo la tecnología se incluyen apartados con información de referencia especifica tanto del lenguaje como de la tecnología J2EE.

Las capas normales de un sistema son las que se refieren a la presentación, el negocio y el acceso a datos. No existe una regla infalible que norme sobre las capas sobre las que hay que centrarse al iniciar el proceso de desarrollo, pero si existen algunas cuestiones a considerar al organizar el trabajo:

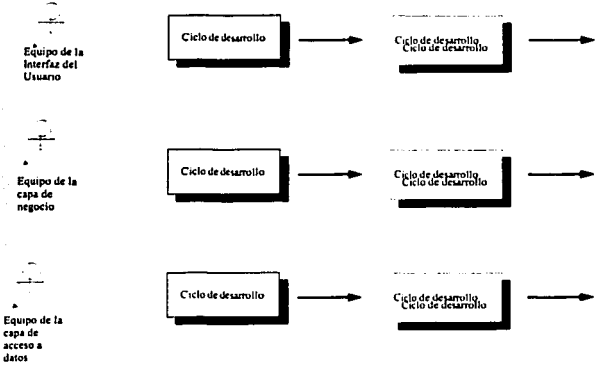


Ilustración 32 : Diseño de las capas del sistema.

- La interfaz del usuario (capa de presentación) es la concretización visible del sistema, y, tomando en cuenta que en el ser humano predomina el sentido de la vista, una interfaz atractiva proporciona satisfacción al cliente aún cuando no sea muy funcional, lo que representa una ventaja al comenzar el desarrollo con una interfaz gráficamente atractiva. Sin embargo la desventaja consiste en que el cliente, al ver la interfaz, supone que el sistema esta casi concluido, cuando apenas acabamos de iniciarlo.
- En un ciclo de desarrollo, cuando un equipo trabaja sobre las capas residentes en el servidor, se recomienda comenzar por los servicios que representan la infraestructura básica del sistema (reglas de negocio, persistencia, clases estructurales, comunicación, procesos principales), sobre las cuales es posible programar los servicios específicos.

- Se recomienda iniciar el soporte a la persistencia al inicio del desarrollo, aunque no necesariamente de forma inmediata, ya que cuando se desarrollan los servicios de persistencia y acceso a datos, se genera un esquema mínimo de bases de datos así como el marco necesario para continuar con el desarrollo paralelo de otras capas. Posteriormente, junto con los procesos de negocio, se mejoran de forma incremental los servicios de datos a lo largo del ciclo de desarrollo.
- Hay que considerar los recursos que se requieren para el desarrollo de cada capa, ya que, si se aborda de forma prematura la tarea, podría no tomarse en cuenta el buen diseño de otros niveles de trabajo. Además, el posponer demasiado las tareas suele favorecer la necesidad de ajustes regresivos o modificaciones de diseño por la interacción de metadatos, esquema de bases de datos, jerarquía de clases, transacciones, etc.

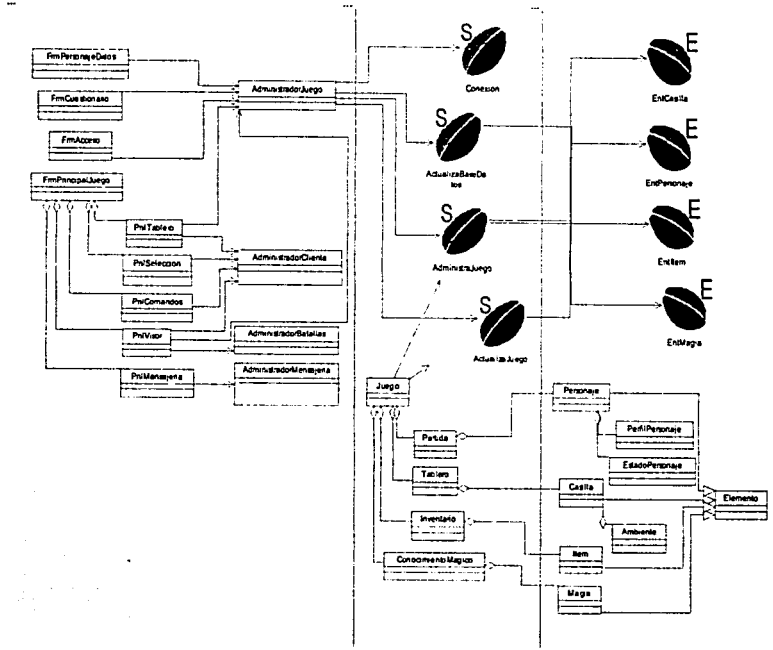


Ilustración 33: Diagrama de Clases

Descripción de la capa de presentación.

La capa de presentación es la más cercana a los usuarios del sistema, proporciona los medios necesarios para la presentación de la información y la recolección de datos, a la vez que asegura el acceso a los servicios de negocio que integran al usuario con la aplicación.

El desarrollo de esta capa se centra fundamentalmente en brindar al usuario final, la interfaz de nuestra aplicación, con base en objetos e información tomados de otros niveles (la capas de negocio y datos), para lo cual se requiere el diseño de pantallas, distribución de botones y otros componentes gráficos, análisis del flujo de información, etcétera, adecuados tanto al negocio como al usuario.

La capa de presentación normalmente reside en una aplicación localizada en la estación de trabajo del usuario final. Las responsabilidades de la capa de presentación son:

- Obtener y gestionar, solicitudes y datos del usuario.
- Enviar la información obtenida del usuario a los servicios de negocios para su procesamiento.
- Recibir los resultados del procesamiento de la capa de negocio.
- Solicitar actualizaciones siempre que sea necesario.
- Presentar resultados al usuario.

En relación con la funcionalidad de la capa de presentación del presente proyecto, se identifican dos aspectos fundamentales en el estudio de su estructura, en primer lugar, las clases encargadas de la recepción, envío y gestión de la información y eventos; y en segundo, los componentes gráficos y contenedores de información que se presentan en la interfaz del usuario para hacer posible la interacción del jugadores con el juego, a través de su aplicación cliente.

Las funciones de control para obtener información del cliente, validarla, traducirla en solicitudes a la capa de negocio, recibir la información resultado y generar las vistas de información empleando las herramientas gráficas del programa instalado en el cliente, se concentran en las clases administradoras del cliente, mismas que estan organizadas de acuerdo al tipo de servicios que ofrecen.

Las funciones y métodos para presentar gráficamente información y las herramientas para interacción con el usuario, se engloban en el estudio de componentes gráficos, clases Java que hacen uso de los APIs swing y awt para mostrar en pantalla formularios, paneles, ventanas y demás clases con orientación a la representación e ingreso de información.

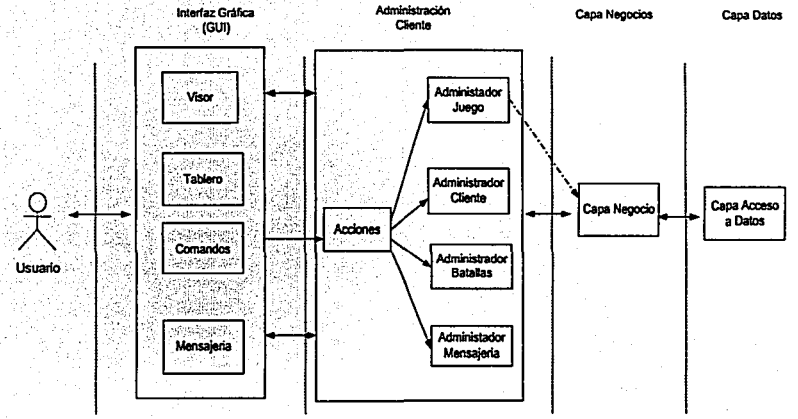


Ilustración 34 : Capas aplicadas al proyecto.

A continuación se describen las clases que integran la estructura de la capa de presentación, organizadas como se mencionó anteriormente, para después revisar los procesos principales de la misma.

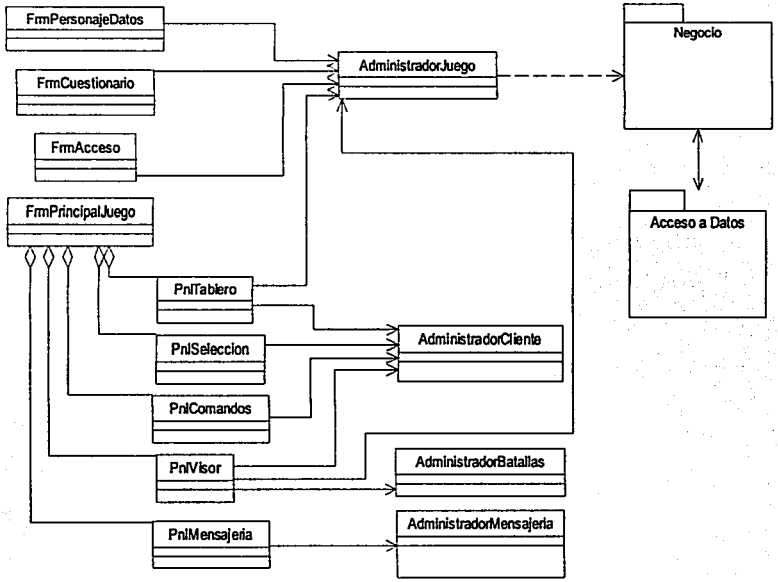


Ilustración 35 : Descrcipción de la capa de presentación.

Administración del cliente: Clases administradoras de información.

El grupo de clases identificadas bajo el concepto de administradoras de información del cliente tienen por función gestionar la información que viaja entre las capas de negocio y de presentación de forma que sean significativas para los fines del sistema en general. Este grupo de clases presenta imágenes de los servicios que ofrece la capa de negocio, pero además realiza validaciones de información significativas solo al cliente, reduciendo en algunos puntos la carga de trabajo del servidor.

Estas clases definen el comportamiento del juego del lado del cliente, que recibe información del juego, mismo que reside en la capa de negocio y es soportado por la capa de datos.

Tabla: Listado de clases administradoras de información del cliente.

Nombre	Responsabilidad	Paquete	Discusión
AdministradorJuego	Se encarga de obtener datos actualizados directamente del Juego (servidor) y de realizar los cambios producto de las acciones del cliente.	*Tonalli.cliente.administracion*	*Es el Módulo que hace referencia a la capa de Negocio.*
AdministracionCliente	Se encarga de las actualizaciones en el estado del cliente, la representación del cliente y su posición actual en el juego. Notifica cambios en selecciones hechas por el usuario.	*Tonalli.cliente.administacion*	
AdministradorBatallas	Maneja el envío y recepción de mensajes en Modo "Batalla" asi como validaciones para finalizar la batalla..	*Tonalli.cliente.administracion*	
Administrador Mensajeria	Maneja el envío y recepción de mensajes en Modo "Exploración".		
AnalizadorSintaxis	Se encarga de verificar que exista el comando escrito en el Panel de Mensajería, así como su sintaxis.	*Tonalli.cliente.administracion*	*La existencia de los comandos lo toma de una variable estática de las Clase Acciones.*
Observa	Clase que sirve de intermediario entre observadores y observados. Personaliza a la clase java.util.Observable para su uso en el juego.	*Tonalli.cliente.administracion*	
Acciones	Contiene la implementación de todas las acciones empleadas durante el desarrollo del juego.	*Tonalli.cliente.acciones*	
Principal	Clase con la que inicia la Aplicación Cliente.	*Tonalli.cliente.administracion*	*Trata de obtener por medio de la Clase AdministradorJuego una referencia a la Capa de Negocios.*

La administración de la información de los clientes, en el videojuego en línea, esta organizada de acuerdo a funciones particulares como sigue:

Clase: Principal.

Es el punto de inicio para la aplicación. Incluye un método main, que especifica la primera función a ejecutar.

Tabla: Servicios de la clase: Principal.

Nombre	*Parametros*	*Tipo Salida*	*Objetivo*	*Efecto Colateral*	*Discusion*
main	String arg[]	void	Clase con la que se inicia la Aplicación		Si el Administrador del Juego obtiene la referencia al BeanConexion, Principal crea instancia de la clase FrmAcceso. Si no muestra un diálogo con el mensaje de error.

Clase: AdministradorJuego.

Es responsable de obtener y gestionar la información del juego, que es compartida por todos los jugadores activos. Establece comunicación con la capa de negocio, es quien conoce a los beans de sesión que la componen y por lo tanto, los servicios que ofrecen. Tiene la capacidad de conocer el juego, se encarga de obtener información y de hacer efectivas las acciones de juego generadas por el cliente.

Los servicios que ofrece la capa de negocio a los clientes tienen una imagen en métodos de esta clase, haciéndolos accesibles a los componentes de la aplicación cliente.

Los métodos y atributos estáticos implementados permiten el acceso a información y procesos del juego que no requieren una instancia particular del administrador del juego, con lo que, a lo largo del juego, solo se crea un único objeto de tipo AdministradorJuego, que puede ser accedido por cualquier clase que lo requiera.

Los cambios en el Administrador del juego afectan a la interfaz gráfica del cliente así como a las condiciones de juego presentadas, por lo que se hace uso de la Clase Observa para permitir registrar observadores y notificar los cambios.

Tabla: Servicios de la clase: AdministradorJuego.

Nombre	*Parametros*	*Tipo Salida*	*Excepciones*	*Objetivo*	*Efecto Colateral*
administradorJuego	String nombreCliente	AdministradorJuego		Método constructor, se declara como una clase que debe ser observada.	Obtiene las referencias a la Capa de Negocios
aplicaEfecto	Collection persona Int[] efecto			Aplicar un efecto de una magia o de un item sobre los estados de los Personajes.	Se hace uso de una referencia a la Capa de Negocios(Bean ActualizaJuego).
asociaItem	String nombrePers String nombreItem			Asocia un Item al Inventrio de un Personaje	
asociaMagia	String nombrePers String nombreMagia			Asocia una Magia al ConocimientoMagico de un Personaje.	
buscaItem	String nombre	Casilla		Busca un item asociado al Inventario del Juego regresando la Casilla.	
colocaItem	String nombre, int coordX, int coordY			Coloca un item asociado al Inventario del Juego en una Casilla.	
colocaPersonaje	String nombre, int coordX, int coordY				
desasociaItem	String nombrePers, String nombreItem			Desasocia un Item en el Inventario de un Personaje	
desasociaMagia	String nombrePers String nombreMagia			Desasocia una Magia en el ConocimientoMagico de un Personaje	
climinaItem	String nombre			Saca a una Item del Juego	
eliminaPersonaje	String nombre			Saca a un Personaje del Juego.	
estableceObservador	Observer o			Añade un objeto observador suceptible de notificación.	
obtenBeansRemotos	String nombreCliente		CreateException, Remote Exception	Obtiene las referencias de los beans de la Capa de Negocios necesarios para su creación.	

Nombre	Parametros	Tipo Salida	Excepciones	Objetivo	Efecto Colateral
			NamingException		
obtenConexion		boolean	CreateException, RemoteException, NamingException	Obtiene la referencia al Bean de la Capa de Negocios "Conexión", necesario para verificar los datos en la forma de acceso y por lo tanto de inicializar el Juego.	Se hace uso de una referencia a la Capa de Negocios (Bean Conexión)
validaUsuario	String extLogin String extPassword	boolean		Valida que los datos ingresados en la forma de Acceso sean los correctos.	
obtenConocimiento		ConocimientoMagico		Obtiene el ConocimentoMagico asociado al Juego	Se hace uso de una referencia a la Capa de Negocios(Bean AdministraJuego)
existeCliente	String extLogin	boolean		Valida si ya esta el cliente en el Servidor.	
muevePersonaje	String nombre int destX destY			Mueve a un Personaje de Casilla Se notifica un cambioen la clase	
obtenCasilla	int coordX int coordY	Casilla		Obtiene una Casilla con una determinada coordenada.	
obtenInventario		Inventario		Obtiene el Inventario asociado al Juego.	
obtenItem	String nombre	Item		Obtiene un Item asociado al Juego	
obtenMagia	Strng nombre	Magia		Obtiene una Magia asociada al Juego	
obtenNumCasillasLado		int		Regresa número de casillas por lado del Tablero.	
obtenNumCasillas		int		Obtiene el número de Casillas manejads por el Juego	
obtenPartida		Partida		Obtiene el conjunto de Persojanes activos en el Juego.	
obtenPersonaje	String nombre	Personaje		Obtiene un personaje activo en el Juego.	
obtenTablero		Tablero		Obtiene el Tablero asociado al Juego.	
sacaItem	String nombre			Saca un Item del Juego.	
sacaPersonaje	String nombre			Saca a un personaje del Juego.	

Clase: AdministradorCliente.
Es responsable de mantener la información que identifica y personaliza al cliente.

Almacena información que es significativa solo a el cliente, los datos que le son relevantes. Esta información es útil para construir las solicitudes del cliente y para afectar la interfaz gráfica respecto a factores locales como selección de elementos y modo en que se encuentra la aplicación. Lleva registro de la interacción que tiene el jugador con su interfaz gráfica.

Los métodos y atributos estáticos implementados permiten el acceso a información y procesos del juego que no requieren una instancia particular del administrador del cliente, con lo que, a lo largo del juego, solo se crea un único objeto de tipo AdministradorCliente, que puede ser accedido por cualquier clase que lo requiera.

Los cambios en el Administrador del cliente afectan a la interfaz gráfica del cliente y a las acciones que requieren selección de elementos, por lo que se hace uso de la Clase Observa para permitir registrar observadores y notificar los cambios.

Tabla: Servicios de la clase: AdministradorCliente.

Nombre	*Parametros*	*Tipo Salida*	*Objetivo*	*Efecto Colateral*	*Discusion*
Administrador Cliente	String nombreCliente	Administra cionCliente	Método Constructor		Establece el Modo del cliente a"Exploración".
estableceCasil laSeleccion	Casilla casilla		Establece el Foco de la Aplicación Cliente en una Casilla	Notifica que ha cambiado la selección de Casilla.	
estableceItem Seleccion	Item item		Establece el Foco de la Aplicación Cliente en un Item determinado	Notifica que ha cambiado la selección del Item.	
estableceMod oCliente	String modoCliente		Establece el Modo de Juego	Notifica que ha cambiado el modo del Cliente.	
estableceObse rvaodr	Observer o		Establece que objetos observan los cambios de la Clase, por lo tanto susceptibles de notificación.		
modificaInven tarioCliente			Notifica que el Inventario del Cliente ha cambiado		
muevePersona			Notifica que la posición del Personaje ha		

Nombre	Parametros	Tipo Salida	Objetivo	Efecto Colateral	Discusion
je			cambiado.		
obtenCasillaCliente		Casilla	Se obtiene la Casilla actualizada del juego-		Invoca al método de buscaPersonaje de la Clase AdministradorJuego.
obtenCasillaSeleccion		Casilla	Se obtiene la Casilla en Foco de la Aplicación		
obtenItemSeleccion		Item	Se obtiene el Item en Foco de la Aplicación		
obtenModoCliente		String	Se obtiene el Modo de Juego de la Aplicación		
obtenNombreClielnte		String	Se obtiene el Nombre del Personaje Cliente		
obtenPersonajeCliente		Personaje	Se obtiene el Personaje Cliente		Invoca al método de obtenPersonaje de la Clase AdministradorJuego.
obtenPersonajeSeleccion		Personaje	Se obtiene el Personajen Foco de la Aplicación.		

Clase: Observa.

Sirve de intermediario en el proceso Observador-Observado. El comportamiento de algunas clases es dependiente de el estado o cambios en otras clases, esto define un esquema en el que las clases se observan. Cuando la clase observada cambia, notifica ese cambio a las clases observadoras, que deciden cómo darle seguimiento. Esta clase personaliza a la clase Observable para su uso en el juego.

Aprovechando la característica de polimorfismo del lenguaje, implementa dos métodos haCambiado que son los encargados de informar cambios en las clases que ocupan el esquema de observación descrito anteriormente, adaptados al tipo de modificación que se informa. En el caso de las componentes gráficas, se notifica un cambio por medio de una cadena que describe la actualización a realizar y que los observadores pueden manejar. En el caso particular de la mensajería se envía como notificación un objeto de tipo Mensaje para ser escrito en la salida del cliente.

Dentro del sistema, toda clase que deba ser observada hace uso de esta clase para obtener esa funcionalidad.

Tabla: Servicios de la clase: Observa.

Nombre	Parametros	Tipo Salida	Objetivo	Discusion
estableceObservador	Observer o		Establece un objeto que va a observar a otra clase	
haCambiado			Notifica que ha cambiado la Clase Observada	
haCambiado	String cambio		Notifica que ha cambiado la Clase Observada	

Clase: AdministradorMensajeria.

Es responsable del envío y recepción de mensajes de comunicación. Mantiene una cola de mensajes enviados, que es revisada por el panel que los despliega al cliente.

Los métodos y atributos estáticos implementados permiten el acceso a información y procesos del juego que no requieren una instancia particular del administrador de mensajería, con lo que, a lo largo del juego solo se crea un único objeto de tipo AdministradorMensajeria, que puede ser accedido por cualquier clase que lo requiera.

Hace uso de la clase para establecer observadores y notificar cambios a los componentes gráficos encargados del manejo de mensajes.

Tabla: Servicios de la clase: AdministradorMensajeria.

Nombre	Parametros	Tipo Salida	Excepciones	Objetivo	Efecto Colateral	Discusion

Clase: AnalizadorSintaxis.

Es responsable de gestionar los mensajes que requieren una sintaxis específica, es decir, los mensajes que representan acciones del cliente. El jugador tiene la posibilidad de escribir acciones específicas de forma alternativa con el uso de controles para generar acciones, esta clase se encarga de la evaluación de la solicitud textual del cliente así como de identificar a qué acción corresponde y si la estructura acción-parámetros es correcta.

Tabla: Servicios de la clase: AnalizadorSintaxis.

Nombre	*Parametros*	*Tipo Salida*	*Objetivo*	*Discusion*
validaExistenciaAcciones	String comando String argumento	int	Valida que la accion sea un método de la clase Acciones	
validaSintaxisAcciones	String argumento int accion	Boolean	Valida que la sintxis del comando sea correcto. Incluye nombre del comando y el correcto paso de argumentos.	

Clase: AdministradorBatallas.

Las batallas son una parte importante de la interacción entre jugadores, esta clase gestiona los eventos que se generan en el proceso y les da seguimiento, haciendo efectivas las acciones generadas.

Hace uso de la clase para establecer observadores y notificar cambios a los componentes gráficos involucrados en el manejo y representación de batallas.

Tabla: Servicios de la clase: AdministradorBatallas.

Nombre	*Parametros*	*Tipo Salida*	*Excepciones*	*Objetivo*	*Efecto Colateral*	*Discusion*

Clase: Acciones.

Es responsable de definir el efecto de cada actividad que es posible generar en el juego. Los componentes del cliente tienen la capacidad de llamar a esta clase cuando reciben eventos del cliente y esta a su vez determina qué métodos deben ser llamados y la forma en que se afecta el juego como resultado. Puede considerársele un catalogo de acciones.

Tabla: Servicios de la claseAcciones.

Nombre	*Parametros*	*Tipo Salida*	*Objetivo*	*Efecto Colateral*	*Discusion*
aplicar	String nombreMagia String nombrePersonaje		Aplica el efecto de una Magia sobre el estado de un Personaje.	Actualiza PnlVisor y PnlComandos.	Se mustra un dialogo con el resultado de aplicar el efecto de la magia.
atacar	String nombreAdversario				
buscar	String nombre		Busca un elemento(Personaje,Item) en el Tablero asociado al Juego.		Hace uso de la Clase Administrador Juego y AdministradorCliente. Busca la Casilla donde se encuentre el personaje o el item deseado.
calculaDireccion	int origenX int origenY int destinoX int destilnoY	int[]	Determina una ruta preferida de movimiento		
calculaDistanciaPM	int origenX int origenY int destinoX int destinoY	int	Calcula la distrancia Manhattan entre Casillas.		
calculaRuta	int origenX int origenY int destinoX int destinoY int Rango	Collection	Evalua la ruta, examinando las direcciones posibles comenzando por la más lógica		
cederTurno					
dar	String nombreItem String		Desasocia un Item del Inventario del Personaje.	Asocia un Item al Inventario del Personaje Destino. Hace uso de la Clase AdministradorJuego y	Esta clase en su método dar verifica que existan esos dos elementos y que se encuentre los personajes en la

Nombre	Parametros	Tipo Salida	Objetivo	Efecto Colateral	Descripcion
	nombrePersonaje			AdministradorCliente.	misma casilla. y se desasocia el Item en el Inventario del Cliente
defender					
encuentraCasillaSegura	String nombreEnemigo	Casilla			
evaluaRuta	int origenX int origenY int destinoX int destinoY int Rango	Collection			
huir					
mover	int destinoX int destinoY		Mueve de posición a un Personaje		Calcula la distancia Maniatan entre origen y destino y determina si no esta fuera del alcance del rango de movimiento del Personaje. Realiza otras validaciones referentes a la población y color de las casillas.
muestraAdvertencia	String mensaje		Muestra un dialogo con un mensaje de advertencia		
muestraDecision	String mensaje	boolean	Muestra un dialogo para preguntara acerca de una decisión.		
muestraInformacion	String mensaje		Muestra alguna información sobre una acción realizada.		
obtenPersonajesAfectados	int alcance	Collection	Obtiene los personajes afectos por la aplicación del efecto de una magia.		
recibeReto	String		Manda un mensaje al		

Nombre	Parametros	Tipo Salida	Objetivo	Efecto Colateral	Discusion
	nombreAdversario		Personaje Retado. Si acepta la batalla, se pasa a modo batalla. Si no puede escapar a una casilla segura.		
retar	String nombrePersonaje		Invita a un Personaje a entrar en Modo Batalla	El personaje objetivo recibirá una Dialogo de Decisión para aceptar o rechazar el reto.	
salir			Sale de la Aplicación.		
seleccionarCasilla	int coordX int coordY		Pedir el foco en una Casilla determinada.	Se actualizan los PnlTablero y PnlVisor.	
seleccionarItem	String nombre		Pedir el foco en un Item determinado.	Se actualizan los PnlSeleccion y PnlVisor.	
seleccionarPersonaje	String nombre		Pedir el foco en un Personaje determinado.	Se actualizan los PnlSeleccion y PnlVisor.	
tomar	String nombreItem		Asociar un Item al Inventario del Personaje.	Se actualizan los PnlSeleccion, PnlVisor y PnlComandos.	
usar	String nombreItem String nombrePersonaje		Aplicar el efecto de un Item en el Estado de un Personaje.	Se actualizan los PnlSeleccion, PnlVisor y PnlComandos.	Se muestra un dialogo donde se indica el resultado de usar el item.

La interfaz del usuario esta estructurada en ventanas y paneles que muestran al usuario la información del juego y que le permiten ingresar información y generar eventos. Define la forma en que se va a organizar la información y los componentes, requiere de un análisis detallado del producto que se quiere conseguir, tomando en cuenta el diseño de las ventanas, la navegación entre ellas, la distribución de los elementos gráficos que la componen, el flujo de información en la aplicación del cliente, las exigencias del sistema y los requerimientos del usuario final.

La interfaz del usuario del caso práctico incluye una ventana para autentificación de usuarios, una ventana para registros de una nuevas cuentas y registro de personajes, una ventana para visualización de información del personaje y la ventana del juego con distintas vistas. Las clases en este apartado observan los cambios en los administradores del juego y del cliente para actualizar su imagen y adaptarla al estado más reciente del juego.

Tabla: Listado de clases del FrmAcceso.

Nombre	*Responsabilidad*	*Paquete*	*Discusión*
FrmAcceso	Clase de Interfaz Grafica que sirve como medio para conectarse con el Servidor e iniciar un Juego por medio de introducir los datos de la cuenta del usuario. También cuenta con la opción de entre al FrmCuestionario para la creación de una nueva cuenta.	*tonalli.cliente.ui*	*Utiliza a la clase AdministradorJuego para validar los Datos. En dos threads como clases anónimas. crea un splash, lo muestra y despliega el FrmPrincipalJuego*
FrmCuestionario	Clase de Interfaz Gráfica que Contiene un formulario para datos del Personaje y un cuestionario para obtener la raza del Personaje a registrar de acuerdo a la personalidad que desee tener el usuario.	*tonalli.cliente.ui*	*Frame contenedor de dos paneles, el que contiene los datos de la cuenta PnlDatosCuenta y el que contiene los datos del personaje PnlDatosPersonaje. Al hacer clic en el botón de aceptar se accede a ObtenResultados para obtener el nombre de la raza, el color, la imagen del personaje y el sexo del personaje y del usuario.*
FrmPersonajeDatos	Clase de Interfaz Grafica que muestra una imagen del personaje a crear junto con los datos de éste tales como nombre, edad, raza. Se registra el usuario y se accesa al sistema.	*tonalli.cliente.ui*	*La imagen se trae del FrmCuestionario al generar los datos, el nombre de la raza, el color, la imagen del personaje y el sexo del personaje y del usuario; los cuales son desplegados en pantalla. Una vez presionado el botón de Aceptar Datos se esconde el botón de Cancelar y se accede a la capa de negocio para guardar los datos.*
FrmPrincipalJuego	Clase de Interfaz Gráfica donde el usuario puede interactuar dentro del Juego. Contiene los paneles comando de comandos, seleccion, tablero y mensajeria.	*tonalli.cliente.ui*	

El siguiente diagrama describe la navegación entre las pantallas del juego:

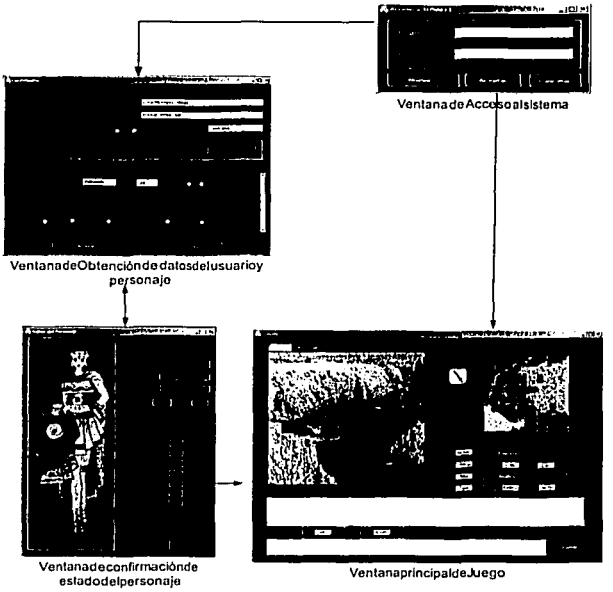


Ilustración 36 : Navegación del proyecto.

La ventana de inicio es siempre la de acceso al sistema en donde se ingresa la información para la autentificación del cliente, si la validación de usuario es exitosa se muestra la ventana principal del juego. En caso de no poseer una cuenta se proporciona la opción de crearla por medio de un botón en la ventana de acceso, esto abre una formulario que el usuario debe llenar (ventana de creación de cuentas y personajes) , después de lo cual se muestra la información del personaje y cuenta creados. Al aceptar es desplegada la ventana principal del juego.

Ventana de acceso.

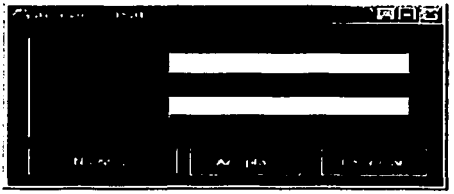


Ilustración 37 : Pantalla de acceso.

Frame: Frm Acceso.

Esta es la primera ventana que se muestra al usuario al iniciar el sistema. Su funcionalidad esta enfocada a la autentificación del jugador como usuario válido del sistema. Permite al cliente introducir su nombre de usuario y contraseña y, si no cuenta con ellos, da acceso a la creación de cuentas. Una vez obtenidos los datos del usuario se solicita al Administrador del juego que valide la información, si es exitosa la validación se llama al frame principal del juego.

Tabla:Clase FrmAcceso.

Nombre	Responsabilidad	Paquete	Discusión
FrmAcceso	Obtiene los datos de autentificación del usuario.	Tonalli.cliente.ui.	Frame que muestra los componentes en donde es posible ingresar la información de autentificación. No incluye páneles internos.

Tabla: Servicios de la clase:FrmAcceso

Nombre	*Parámetros*	*Tipo Salida*	*Objetivo*	*Discusion*
FrmAcceso		FrmAcceso	Método Constructor	Crea un Splash
carga			Despliega el FrmPrincipal	Crea una instancia de la clase FrmPrincipalJuego. Se maneja por medio de threads.
autentifica			Valida que los datos de la cuenta sean los mismos que estan registrados en el Servidor.	Tiene tres intentos para hacer esta autentificación después de lo cual se cerrará la aplicación. Este método también es llamado por medio de un KeyListener de la Tecla <ENTER>.
btn_Cancelar_actionPerformed	ActionEvent e		Sale de a Aplicación	
btn_NuevaCuenta_actionPerformed	ActionEvent e		Despliega el FrmCuestionario para la creación de Cuenta.	Crea una instancia de la clase FrmCuestionario pasándole los datos del login y el password deseado habiendo validado la inexistencia de una cuenta con el login ingresado.
limpiaCampos			Limpia los campos de la aplicación.	
noValoresNulos		boolean	Verifica antes de autentificar los datos, que los campos de la aplicación no esten vacios.	
creaSplash			Crea una ventana que se desplegará mientras se inicializa lo necesario para empezar a el Juego.	
escondeSplash			Esconde el Splash.	Se maneja por medio de threads.
muestraSplash			Despliega el Splash	Se maneja por medio de threads
inicializaJuego			Inicializa los Administradores de Juego y Cliente con el nombre del Personaje Cliente..	Se maneja por medio de threads.

Ventana de registros de cuentas y creación de personajes.

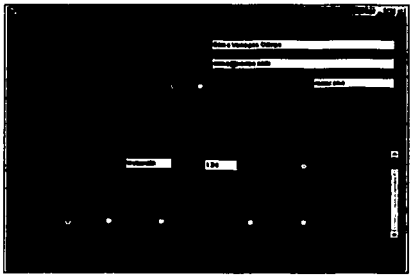


Ilustración 38 : Pantalla d eobtención de información del usuario y personaje.

Frame; FrmCuestionario.

Esta ventana es mostrada cuando el cliente solicita la creación de una cuenta y de un personaje asociado. Se muestra al usuario un formulario en donde debe llenar sus datos personales y aquellos que permitirán al sistema crear un personaje que lo represente dentro del juego. La ventana esta seccionada en los paneles PnlDatosCuenta para ingresar la información del cliente y PnlCuestionario para la del personaje.

Tabla: Listado del Frmcuestionario

Nombre	*Responsabilidad*	*Paquete*	*Discusión*
FrmCuestionario	Contiene al PnlCuestionario y Pnl DatoCuenta. Obtiene los datos del usuario tanto personales como de su personaje.	Tonalli.cliente.ui.	Frame que muestra los datos traídos del FrmAcceso para mostrar el login y el password. Contiene dos paneles y se relaciona con ObtenResultado para validar campos vacíos.
PnlCuestionario	Contiene los datos del personaje y el cuestionario que será validado por otra clase para obtener los datos iniciales.	Tonalli.cliente.ui.	Panel que contiene los grupos de botones de radio para obtener los resultados de un cuestionario que determinará los datos iniciales del personaje.
PnlDatosCuenta	Esta dedicado a los datos personales del usuario.	Tonalli.cliente.ui.	Panel que contiene el panel de fecha , el cual obtiene la fecha de nacimiento del usuario.
PnlFecha	Esta dedicado para obtener la fecha de nacimiento del usuario	Tonalli.cliente.ui.	Tiene un rango para mayores de 18 años.
EdadTemplate	Hace las validaciones para el FrmCuestionario en su campo deedad del personaje.	Tonalli.cliente.ui.	Clase que sirve para validar el campo de edad del personaje. Por tratarse de personajes fantásticos, el rango es de 000 hasta 199, se debe seguir dicho formato.
ObtenResultados	Calcula los datos iniciales del personaje(sexo, raza, estado) de un usuario que crea una nueva cuenta.	Tonalli.cliente.ui.	Obtiene el sexo del personaje y del usuario y por medio del mismo, asigna el nombre de la imagen que se desplegará en el panel de datos. Calcula la raza del personaje y dependiendo de dicho dato, asigna, estado del personaje, descripción de la raza y color.

Tabla: Servicios de la clase: FrmCuestionario

Nombre	*Parametros*	*Tipo Salida*	*Objetivo*	*Efecto Colateral*	*Discusion*
FrmCuestionario	FrmAccesoformaLogin	FrmCuestionario	Método Constructor		Recibe los datos de una instancia del FrmLogin que es la ventana padre.
regresaForma			ocultar FrmCuestionario y mostrar la forma de Acceso con sus campos vacíos.		
btn_Aceptar_actionPerformed	ActionEvente		Obtiene el Resultado de los datos ingresando definiendo la raza, color, estado del personaje.	Cerrar la aplicación cuando los intentos sobrepasen el tercero.	Valida que no exista ningún campo vacío, obtiene el resultado y crea una instancia de la clase FrmPersonajeDatos, Tiene tres intentos para llenar el cuestionario adecuadamente.

Panel: PnlDatosCuenta.

Este panel, parte del frame FrmCuestionario, permite el ingreso de la información personal del cliente para la creación de la cuenta que le permitirá el acceso al sistema. Utiliza al panel PnlFecha para facilitar la selección y validación de fechas.

Tabla: Servicios de la clase: PnlDatosCuenta

Nombre	*Parametros*	*Tipo Salida*	*Objetivo*	*Discusion*
PnlDatosCuenta		PnlDatosCuenta	Crear un Panel donde se despliegue un formulario con datos personales del usuario.	Accede a ObtenResultado para validar los campos vacíos cuando se presiona el botón de Aceptar en el frame.

Panel: PnlFecha.

Panel utilizado por el panel PnlDatosCuenta para presentar al usuario cuadros desplegables donde pueda seleccionar año mes y dia de nacimiento, tomando en consideración años bisiestos, y días por mes en el calendario.

Tabla: Servicios de la clase; PnlFecha

Nombre	*Parametros*	*Tipo Salida*	*Objetivo*	*Discusion*
PnlFecha		PnlFecha	Crear un Panel que facilite la determinación de la fecha de nacimiento del usuario por medio de ComboBoxes.	Validación de un rango de fechas como primer medida de acceso a mayores de 18 años.
PnlFecha	String strTitulo	PnlFecha	Crear un Panel que facilite la determinación de la fecha de nacimiento del usuario por medio de ComboBoxes con un titulo.	Validación de un rango de fechas como primer medida de acceso a mayores de 18 años. Coloca el título del parámetro para el panel.
getFecha		String	Obtiene la Fecha de Nacimiento del Usuario	
itemState Changed	ItemEvent e		Metodo para verificar que el numero de dias corresponda al mes y al año aunado el caso de Febrero, cuyos días cambian dependiendo de si el año es o no bisiesto.	

Panel: PnlCuestionario.

Este panel, parte del frame FrmCuestionario, permite el ingreso de la información para la creación del personaje asociado al usuario. Hace uso de la clase EdadTemplate para validar el dato relacionado con la edad del personaje.

Tabla: Servicios de la clase: PnlCuestionario

Nombre	*Parametros*	*Tipo Salida*	*Objetivo*	*Efecto Colateral*	*Discusion*
PnlCuestionario		PnlCuestionario	Crear un Panel el cual contenga varias preguntas sobre la personalidad del personaje determinando la raza y el color al que pertenecerá.	Validación de los campos vacíos además del campo de edad, que no exceda de 199 y que siga el formato que le valida EdadTemplate	Accede a ObtenResultado para calcular los resultados del cuestionario y determinar los datos iniciales del personaje, una vez presionado el botón de Aceptar en el Frame que lo contiene.

Clase: EdadTemplate

Se encarga de validar la entrada e el campo edad dentro del panel PnlCuestionario. Ejemplifica la forma de verificar cada carácter de una entrada contra el tipo de dato requerido, en este caso entero.

Tabla: Servicios de la clase: EdadTemplate

Nombre	*Parametros*	*Tipo Salida*	*Excepciones*	*Objetivo*	*Efecto Colateral*	*Discusion*
EdadTemplate		EdadTempl ate		Método constructor que establece la longitu permitida del campo a validar hasta 3 caracteres	Valida el campo de edad del personaje en PnlCuestionario	Asigna la longitud permitida como 3 caracteres.
EdadTemplate	int iLongitud	EdadTempl ate		Método constructor que establece la longitu permitida del campo a validar según el parámetro enviado.	Valida el campo de edad del personaje en PnlCuestionario	Asigna la longitud del parámetro para validación.
insertString	int offs String str AttributeSet a		BadLocationEx ception	Método que toma el contenido del campo y lo pone en un arreglo de caracteres para la validación de cada uno.		Se coloca el contenido del campo de edad del personaje en un arreglo de caracteres para la validación de cada uno de ellos.

Clase:ObtenResultados.

Empleada por el panel cuestionario. Se encarga de obtener la raza, sexo, estado inicial y edad del personaje, con base en los datos del cuestionario, junto con los demás datos de la cuenta. Calcula los datos iniciales del personaje asociado a un usuario que crea una nueva cuenta.

Tabla: Servicios de la clase:ObtenResultado

Nombre	*Parametros*	*Tipo Salida*	*Objetivo*	*Efecto Colateral*	*Discusion*
ObtenResultado	PnlDatosCuenta pnlCuenta PnlCuestionario pnlCuestionario	ObtenResultado	Constructor que necesita las referencias de los paneles del frame del Cuestionario		Método que llama a métodos que obtienen los datos del usuario y personaje por separado.
mayor	int I		Método para obtener el mayor de un arreglo de enteros	Obtener el elemento que más se selecciona en el cuestionario.	Obtiene el elemento mayor del arreglo de enteros que determinan la raza. El arreglo se forma con los datos del PnlCuestionario.
obtenRaza			Obtiene la Raza del Personaje, determinando su Estado según la raza.	Validación del mayor.	Llena el arreglo de enteros por medio de los datos del cuestionario, coloca el puntaje según el valor.
obtenRazaPuntos	int resp		Obtiene los puntajes del cuestionario determinando cuantos puntos corresponde a cada raza.		Obtiene el total de puntaje de los valores del cuestionario teniendo un rango de 10 a 50.
obtenSexo	String i	String	Obtiene el sexo del Personaje	Prepara el dato para su almacenamiento correcto en la base de datos.	Pasando el valor del botón de radio del panel cuestionario y del panel de cuenta, determina el sexo asignando una letra de M o F.
validaCuestionario		String	Verifica que no exista ningún campo vacio en el cuestionario.	En caso de encontrar un campo vacío regresa el nombre del campo. Si no regresa una cadena.	Valida que ninguno de los campos de los paneles se encuentre vacio.

Ventana de Información de personaje.

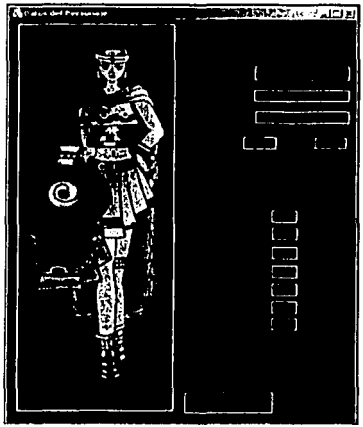


Ilustración 39 : Pantalla de información del personaje.

Frame: FrmPersonajeDatos

Ventana que muestra los datos de un personaje asociado a la cuenta ingresada. Es utilizado cuando el cliente solicita la información de su personaje o para que, al finalizar el proceso de creación verifique que el perfil e información del personaje, producto de los datos enviados, satisfacen la imagen que quiere proyectar dentro del juego. Utiliza al panel PnlPersonaje para desplegar la información junto con la imagen del personaje.

Tabla: Listado de clases de FrmPersonajeDatos

Nombre	*Responsabilidad*	*Paquete*	*Discusión*
FrmPersonajeDatos	Contiene el frame en el que se muestran los datos del personaje de cierto usuario.	Tonalli.cliente.ui.	Despliega los datos iniciales del personaje, en caso de que el usuario presione el botón de Cancelar, se regresará al Frame anterior. Al Aceptar, se desplegará el estado inicial del personaje.
PnlPersonaje	Panel que se coloca en el frame de Datos del personaje. Coloca una etiqueta con una imagen que corresponde al personaje que elige el usuario.	Tonalli.cliente.ui	Panel que inserta la imagen que se desplegará en el FrmPersonajeDatos. Obtiene la referencia del FrmCuestionario

Tabla: Servicios de la clase:FrmPersonajeDatos

Nombre	*Parametros*	*Tipo Salida*	*Objetivo*	*Discusion*
FrmPersonajeDatos	FrmCuestionario frmC	FrmPersonajeDatos	Método Constructor. Crea un FrmPersonajeDatos teniendo los datos de la cuenta y el Personaje a crear.	
FrmPersonajeDatos	Int bandera FrmAccceso frmLogin	FrmPersonajeDatos	Método Constructor. Crea un FrmPersonajeDatos teniendo los datos de la cuenta y el Personaje tomando en cuenta si ya es una cuenta registrada o no.	La bandera valida si la llamada proviene desde el FrmAcceso por medio del botón NuevaCuenta o por medio del botón Aceptar.
carga			Despliega el FrmPrincipal.	
creaSplash			Crea una ventana que de espera de carga del juego.	
escondeSplash			Esconde el Splash.	
muestraSplash			Despliega el Splash	
inicializaJuego	String login		Inicializa los Administradores de Juego y Cliente con el nombre del Personaje Cliente	
bt_Aceptar_actionPerformed	ActionEvent e		Cierra FrmPersonaje Datos, muestra el Splash y se inicializa el Juego.	
btn_Salvar_actionPerformed	ActionEvent e		Crea un Personaje y una nueva Cuenta por medio de un acceso a la capa de Negocios..	Muestra el estado inicial del personaje, esconde los botones de Cancelar y Salvar Datos.
but_Cancelar_actionPerformed	ActionEvent e		En caso de que se trate de una nueva Cuenta, regresa a la forma de cuestionario, de lo contrario, cierra la aplicación	Aparece en caso de que se trate de una nueva cuenta, en cuyo caso, regresa al frame del cuestionario para cambiar los datos.

Panel: PnlPersonaje

Muestra la información del personaje solicitado junto con la imagen asociada a él. Es mostrado dentro del frame FrmPersonajeDatos.

Tabla: Servicios de la clase:PnlPersonaje

Nombre	*Parametros*	*Tipo Salida*	*Excepciones*	*Objetivo*	*Efecto Colateral*	*Discusion*
PnlPersonaje	String nombreImage	PnlPersonaje		Método constructor que muestra una imagen representativa del Personaje.		El nombre de la imagen proviene del FrmCuestionario. En caso de existir, asigna una al azar.

Ventana principal de juego.

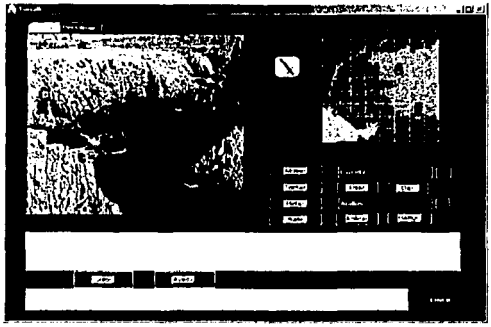


Ilustración 40 : Pantalla de Juego.

Frame: FrmPrincipalJuego

Es la ventana principal de juego, desde la cual el usuario es capaz de interactuar con el juego y de ver los sucesos que se desencadenan de la actividad conjunta de los jugadores sobre el juego. Al ingresar a esta pantalla el frame obtiene la información del juego y del cliente y la despliega en sus páneles. El frame principal esta seccionado en varias partes para lo que hace uso de los paneles: PnlTablero, PnlVisor, PnlSeleccion, PnlMensajeria y PnlComandos.

Tabla: Listado de clases del FrmPrincipalJuego

Nombre	*Responsabilidad*	*Paquete*	*Discusión*
PnlTablero	Contiene la funcionalidad para el manejo gráfico del Tablero	*Tonalli.cliente.ui.tablero*	*Se vale de la clase AdministradorJuego para obtener la información de las casillas y su ambiente al momento de crear una instancia de este clase. Muestra una reticula, cuyo fondo es la imagen de un mapa coordenado.Contiene a las clases PnlSeleccion,AdaptadorMouseTablero*
PnlVisor	Componente encargado de contener parte de la información grafica que se presenta al usuario durante el juego..	*Tonalli.cliente.ui.visor*	*Utiliza un propio MouseInputAdapter para escuchar eventos originados por el mouse dentro del componente.Se apoya en la clase AdministradorBatallas para la mensajeria necesaria en ese modo.*
PnlMensajeria	Panel que sirve de canal de comunicación, teniendo paneles donde se escriben mensaje y se muestran mensajes y comandos.	*Tonalli.cliente.ui.mensajeria*	*Tiene métodos que personalizan el color del mensaje a enviar, así como un dialogo de ayuda de sintaxis de comandos.*
PnlComandos	Clase que genera acciones con el Personaje y su Ambiente.	*Tonalli.cliente.ui.comandos*	*Contiene PnlBotonesExploracion, PnlBotonesBatalla, PnlBotonesModo.*

Tabla: Servicios de la clase:FrmPrincipalJuego

Nombre	*Parametros*	*Tipo Salida*	*Objetivo*	*Discusion*
FrmPrincipalJuego		FrmPrincipalJuego	Clase de Interfaz Grafica que sirve como medio de interacción con el Juego.	Crea instancias de la clases PnlVisor, PnlTablero,PnlMensajeria,PnlSeleccion y PnlComandos.

Ventana principal de juego: Panel del tablero.

Nombre	*Responsabilidad*	*Paquete*	*Discusión*
PnlTablero	*Contiene la funcionalidad para el manejo gráfico del Tablero.*	*Tonalli.cliente. ui.tablero*	*Se vale de la clase AdministradorJuego para obtener la información de las casillas y su ambiente al momento de crear una instancia de este clase.Asi mismo añade una instancia del AdaptadorMouseTablero.* *Se declará observador de la clases AdministradorJuego y AdministradorCliente.*
AdaptadorMouseTablero	Permite el manejo de todos los eventos del mouse comunes al Pnl Tablero.	*Tonalli.cliente. ui.tablero*	*Se declara observador de la clase AdministradorCliente*

Panel: PnlTablero.

Muestra la representación del mundo del juego, en donde se lleva a cabo a acción. El tablero es una colección de casillas, identificadas por coordenadas con diversos ambientes cada una y sobre las cuales los jugadores, representados sobre la casilla e la que se localizan, pueden moverse. Sobre el tablero es posible localizar y seleccionar casillas y personajes para realizar diversas acciones. Para determinar las dimensiones del tablero, identificar al personaje del cliente, así como para dibujar las casillas y su contenido este panel solicita a las clases AdministradorJuego y administradorCliente la información sobre el tablero del juego y del personaje del cliente. El área del tablero en el frame principal del juego es sensible a eventos generados por el ratón, y para su gestión se hace uso de la clase AdaptadorMouseTablero. Este panel observa cambios en el juego a través de la clase AdministradorJuego para actualizar la posición de los elementos sobre el tablero y observa los cambios en la clase AdministradorCliente para determinar los elementos que están seleccionados.

Tabla: Servicios de la clase:PnlTablero

Nombre	*Parametros*	*Tipo Salida*	*Objetivo*	*Efecto Colateral*	*Discusion*
PnlTablero		PnlTablero	Método constructor que crea la retícula con los datos de las casillas en la Base de Datos	Se establece como Observador de las clases AdministraJuego, AdministraCliente.	Obtiene la información de las casillas y del ambiente asociada a ellas y llama a los métodos de dibujo. Añade un AdaptadorMouseTablero para escuchar eventos del mouse.
actualizaCasillas	Graphics g Collection colCasillas		Actualiza la representación de la colección de casillas que recibe.		
dibujaCuadricula	Graphics g		Dibuja la cuadricula, junto con las etiquetas de las coordenadas.		
dibujaPersonaje	Graphics g Casilla casilla		Posiciona la representación de los personajes que asociados a la casilla		Se obtiene la posición del Personje de acuerdo al cuadrante para posteriormente conseguir el nombre de la imagen a dibujar.
dibujaTablero	Graphics g		Dibuja el área en donde se sitúan las casillas(Imagen del Mapa).		
identificaPersonaje	int coordX int coordY int cuadrante	String	stablece qué personaje fue seleccionado dentro de la casilla de acuerdo a la zona (cuadrante) en la que se ha dado click.		Se vale de la clase AdministraJuego para obtener el ambiente de la Casilla.
marcaCasillaSeleccionada	Graphics g int renglonX int columnaY		Borra las marcas anteriores y señala la Casilla por medio de dibujar los bordes de esa casilla y un rectángulo interno pequeño del color de la Casilla.	Se establece como Observador de las clases AdministraJuego, AdministraCliente.	Obtiene la información de las casillas y del ambiente asociada a ellas y llama a los métodos de dibujo. Añade un AdaptadorMouseTablero para escuchar eventos del mouse.
muestraRangoMagia	Graphics g int alcance		Dibuja la retícula de rango de alcance de la Magia		

Nombre	*Parametros*	*Tipo Salida*	*Objetivo*	*Efecto Colateral*	*Discusion*
	Color color				
muestraRangoMovimiento	Graphics g Color color		Dibuja la reticula de rango de movimiento del personaje.		
muevePersonaje	Graphics g		Actualiza las casillas al moverse un personaje.		Se obtiene la posición del Personje de acuerdo al cuadrante para posteriormente conseguir el nombre de la imagen a dibujar.
paintComponent	Graphics g		Redibuja el tablero y su contenido		
update	Observable o Objeto cambio		Metodo implementado de la interfaz Observer que identifica si el cambio que se hizo fue de selección o de movimento.		Se vale de la clase AdministraJuego para obtener el ambiente de la Casilla.

Clase: AdaptadorMouseTablero.
Permite la gestión de eventos aplicables al tablero.

Se encarga de identificar los comandos hechos por el usuario a través del ratón sobre el panel PnlTablero. Su función principal es determinar los elementos seleccionados mediante la pulsación del ratón sobre ellos, para lo cual traduce las coordenadas de la pantalla en coordenadas y cuadrantes del tablero. Extiende a la clase MouseAdapter para heredar las funciones de pulsación del mouse e implementa a la interfaz MouseMotionListener para poder utilizar las funciones de movimiento del mouse. Sobrescribe las funciones de pulsación del ratón (mousePressed) y clic con el ratón (mouseClicked).

Los eventos del ratón son traducidos empleando la clase Acciones, para modificar la interfaz, o actualizar aspectos del juego y del cliente.

Tabla: Servicios de la clase:AdaptadorMouseTablero

Nombre	*Parametros*	*Tipo Salida*	*Objetivo*	*Efecto Colateral*	*Discusion*
Adaptador Mouse Tablero	PnlTablero pnlTablero	PnlTablero	Método Constructor donde se toma como referencia el tamaño del PnlTablero para escuchar los eventos generados.		
estableceCuadrante	int cx int cy	int	Determina el cuadrante seleccionado de la casilla		
mouseClicked	MouseEvent ev		Método que identifica los eventos del mouse.	Al dar doble click se mueve el personaje a la casilla deseada. Al dar un click derecho se ve el rango de movimiento del Personaje.	Solo se puede realizar en modo "Exploracion".
mouseExited			Método de la Interfaz Mouse Listener no implementado		
mouseReleased			Método de la Interfaz Mouse Listener no implementado		
mouseEntered			Método de la Interfaz Mouse Listener no implementado		
mouseDragged			Método de la Interfaz MouseMotionListener no implementado		
mouseMoved			Método de la Interfaz Mouse MotinListener no implementado		
mousePressed	MouseEvent ev		Método que identifica los eventos del mouse	Al dar un clic selecciona una casilla.	Solo se puede realizar en modo "Exploracion".

Ventana principal de juego: Panel de selecciones.

Tabla: Listado de clases de PnlSelecciones.

Nombre	*Responsabilidad*	*Paquete*	*Discusión*
PnlSeleccion	Muestra imagenes del personaje e item seleccionado en el tablero, en el Panel Visor o en el Panel de Comandos.	*Tonalli.client e.ui.tablero*	*Se declara observador de la clase AdministradorCliente.*

Panel:PnlSeleccion.

Muestra las imágenes asociadas al personaje e item actualmente seleccionados. Observa cambios en el administrador del cliente para, cada vez que se selecciona un nuevo elemento, actualizar las imágenes presentadas al cliente.

Tabla: Servicios de la clase:PnlSeleccion

Nombre	*Parametros*	*Tipo Salida*	*Objetivo*	*Efecto Colateral*	*Discusion*
PnlSeleccion		PnlSeleccion	Método Constructor	Se declara observador de la clase AdministradorCliente.	
paintComponent	Graphics g		Llama al método dibujaElementosSeleccionados.		Invoca al método dibujaElementosSeleccinados.
update	Observable o Objeto cambio		Determina si la selección fue un item, un personaje o ambos y dibuja su imagen.		
dibujaElementosSeleccionados	Graphics g		Pinta las imagenes asociadas al Personaje y el Item seleccionados.		Obtiene de la Clase AdmnistraClliente el Item y el Personaje seleccionado.

Ventana principal de juego: Panel del visor.

Tabla: Listado de clases administradoras dePnlVisor.

Nombre	Responsabilidad	Paquete	Discusión
PnlVisor	Contiene el frame en el que se muestran los datos del personaje de cierto usuario.	*Tonalli.cliente.ui. visor*	*Obtiene de la clase AdministraCliente la casilla seleccionada como base para desplegar la información.*
PnlEscena	Componente encargado de contener al Lienzo que representa con lo que el usuario puede interactuar dada su posición en una casilla determinada.	*Tonalli.cliente.ui. visor*	
PnlBatalla	Componente encargado de contener al Lienzo que despliega la información respectiva a las batallas.	*Tonalli.cliente.ui. visor*	
PnlMana	Objeto encargado de llevar una cuenta de los puntos de Mana que un usuario ha utilizado a lo largo de un turno.	*Tonalli.cliente.ui. visor*	*Cada accion realizada por el usuario tiene un costo de Mana. Al utilizar una accion, su mana (capacidad para realizar acciones) se reduce. Esto con el objeto de limitar las acciones que los usuarios puedan realizar y que así mismo establezcan una estrategia en cuanto a su ataque/defensa..*
PnlMensajeBatal la	Panel que despliega toda la informacion de las acciones queun usuario realiza dentro de una pelea. Tiene el propósito de informarlo de las acciones realizadas por él y por sus contrincantes.	*Tonalli.cliente.ui. visor*	
CuadroInformac ion	Dibuja la información de un personaje dentro de la pelea y en el cuadro de información del personaje. Su función es la de mostrar al usuario los valores de su personaje al momento del juego.	*Tonalli.cliente.ui. visor*	

Nombre	Responsabilidad	Paquete	Discusión
Lienzo	Super clase de todo aquello que necesita ser desplegado en el visor.	*Tonalli.cliente.ui. visor*	*Contiene una colección de los Elementos contenidos, asi como de los rectángulos correspondientes a estos (para efectos de * selección)*
LienzoBatalla	Se encarga de mostrar toda la informacion pertinente al momento de una batalla- Esto es util para determinar quien sera el objetivo de la próxima acción del usuario.	*Tonalli.cliente.ui. visor*	*Despliega: Menus y Sub-Menus, Cuadros de Informacion de los Personajes, Personajes, Rectángulo, delimitador de la seleccion del personaje. Esto es util para determinar quien sera el objetivo*
LienzoBatallaMenú	Componente encargado de representar cada uno de los menus y SubMenus que se despliegan al usuario durante las batallas.	*Tonalli.cliente.ui. visor*	
LienzoEscena	Componente encargado de la representación Grafica de la casilla en la que se encuentra un personaje.	*Tonalli.cliente.ui. visor*	
AdaptadorMouse	Permite el manejo de todos los eventos del mouse comunes a los paneles del visor (escena y batalla).	*Tonalli.cliente.ui. visor*	
MouseAdapterBatalla	Permite la gestión de todos los eventos relativos al mouse cuando el jugador se encuentra en la modalidad de Batalla.	*Tonalli.cliente.ui. visor*	*Extiende a la clase AdaptadorMouse.*
MouseAdaptarEscena	Permite la gestión de todos los eventos relativos al mouse cuando el jugador se encuentra en la modalidad de Escena.	*Tonalli.cliente.ui. visor*	*Extiende a la clase AdaptadorMouse*
TipText	Despliega la descripcion de un elemento en pantalla a manera de ToolTip.	*Tonalli.cliente.ui. visor*	*Utiliza el contexto grafico de la clase que lo manda llamar para escribir la descripción del elemento.*

Panel: PnlVisor.

Muestra la información gráfica que se presenta al usuario con respecto a la casilla en la que se encuentra actualmente y a los factores y actividades que lo definen. Este panel puede considerarse la visión que tiene el personaje de si mismo y del espacio en donde se encuentra, delimitado por una casilla, en la se representan todos los elementos que conviven dentro de esa coordenada particular del tablero. Extiende a JTabbedPane para proporcionar la funcionalidad de pestañas dentro del panel en donde se organizan las diversas vistas de este componente, que son: la que representa lo que el personaje ve en una casilla (por medo del panel PnlEscena), la que muestra la información del personaje del cliente (con el panel PnlPersonaje), y la que muestra los comandos y actividades de las batallas (a través del panel PnlBatallas).

La información que grafica depende del personaje del cliente y de su casilla actual, datos que son consultados al administrador del cliente para representar el ambiente o la información a presentar.

Cada vez que se suscita un cambio en el contexto del juego, este componente recibe el cambio y se encarga de realizar las gestiones necesarias para mostrar al usuario el estado actual del juego, actualizando los componentes que contiene. Para esto, se establece como observador de las clases AdministraJuego, AdministraCliente y AdministraBatallas.

Los componentes de este panel son sensibles a eventos del ratón que se manejan a través de las clases MouseAdapterBatalla, y MouseAdapterEscena que extienden a la clase AdaptadorMouse.

Tabla: Servicios de la clase:PnlVisor

Nombre	*Parametros*	*Tipo Salida*	*Objetivo*	*Efecto Colateral*	*Descripcion*
PnlVisor		PnlVisor	Método Constructor.	Se establece como observador de las clases AdministradorJuego y AdministradorCliente.	Crea instancias de la clase PnlEscena,PnlBatalla y PnlPersonaje. Pinta los datos del Personaje.
obtenFondo Casilla		String	Dado el color de una casilla, se obtiene el fondo que corresponde para que pueda ser desplegado en el componente (escena o batallas) que así lo requiera.		Obtiene de la clase AdministradorCliente la casilla seleccionada.
obtenPnlEs cena		PnlEscena	Obtiene el PnlEscena		
update	Observable ob Object cambio		Este método se encarga de realizar las gestiones necesarias cuando existe un cambio dentro del juego, ya sea por un movimiento de casilla, o cambiar de Modo Exploración a Modo Batalla y viceversa.		

Clase: AdaptadorMouse.

Se encarga de identificar los comandos hechos por el usuario a través del ratón sobre el panel PnlVisor. Incluye las funciones comunes a los adaptadores del mouse particulares para la escena y las batallas. Incluye métodos para determinar la acción generada por un evento del ratón (como puede ser la selección de elementos), determinar su validez y gestionar las marcas de selección sobre los lienzos.

Tabla: Servicios de la clase:AdaptadorMouse

Nombre	*Parametros*	*Tipo Salida*	*Objetivo*	*Discusion*
AdaptadorMouse		Adaptador Mouse	Método Contructor.	
mousePressed	MouseEvent e		Evalua cuando se presiona el mouse sobre el tablero.	Llama a determinaAccion y establece el rectangulo que delimita al personaje seleccionado.
determinaAccion			Determina la accion a seguir en caso de que se haya presionado el mouse sobre un elemento valido.	Implica la funcionalidad del Visor en su modalidad de Escena.
realizaAccion Activado	Graphics2D g2d		Si el puntero del mouse se encuentra sobre un elemento valido se dibuja un rectangulo sobre este y se cambia el cursor por defecto (flecha) por una mano.	
realizaAccionNoActivado			Si el puntero del mouse no se encuentra sobre un elemento valido se deja de dibujar el rectangulo que lo delimita y se regresa el cursor por defecto.	Asi mismo se da mantenimiento a las variables requeridas para saber si se debe o no de dibujar el rectangulo delimitador del elemento.
validaMovimiento	Collection colElem Collection colBounds MouseEvent e	boolean	Valida que el puntero del mouse se encuentre sobre un elemento valido.	

Clase: Lienzo.

Es una super clase utilizada por todos los componentes del visor que requieren presentar imágenes sensibles a selección. Subclases de esta clase son empleadas por el visor de la escena y el visor de batallas para mostrar en pantalla un ambiente determinado. Incluye la distribución de los elementos, les asigna un espacio (rectángulo asociado al elemento) para poder seleccionarlos e incluye herramientas para identificar gráficamente la selección actual.

Nombre	Parametros	Tipo Salida	Objetivo	Discusion
Lienzo		Lienzo	Método Constructor	
Lienzo	JPanel padre	Lienzo	Método constructor que tiene una referencia a un Panel.	
estableceAlto	int alto		Define el alto del componente para efectos de dibujar la imagen de fondo.	
estableceAncho	int ancho		Define el ancho del componente para efectos de dibujar la imagen de fondo.	
establece Seleccion	Rectangle r		Establece el rectangulo que sera dibujado como parte de la seleccion del usuario.	
estableceValoresI niciales			Permite el mantenimiento de las colecciones de elementos a dibujar y los rectángulos que lo delimintan.	
update	Graphics g		Método sobreescirto sobreescribe para evitar que se deje de pintar el fondo.	
asociaElementos	Collection col, int iX,int iY, int iIncremento		Se actualiza la coleccion de elementos contenidos en la coleccion y se asocian a sus nuevos rectangulos contenedores.	
obtenTipoImagen	Elemento e	String	Regresa la extensión de la imagen a pintar.	
pintaElementos	Graphics2D g2d		Pinta los elementos asociados a una casilla.	
pintaFondo	Graphics2D g2d		Pinta el fondo de la casilla en la que se encuentra el personaje.	
pintaSeleccion	Graphics2D g2d		Pinta el rectángulo que representa al elemento seleccionado actualmente.	
verificaElementos		boolean	Verifica que los Elementos que se encuentran en una casilla determinada en un momento determinado no hayan cambiado.	

Clase: TipText.

Se utiliza para desplegar en pantalla cuadros de información con datos sobre un elemento, a manera de ToolTip Utiliza el contexto gráfico de la clase que lo manda a llamar para presentar la información. Un tool tip es una pequeña ventana rectangular que se despliega cuando el ratón pasa sobre algún área específica pero desaparece cuando el usuario mueve el ratón o hace click lejos del área diseñada. Una ventana 'tooltip' contiene algún texto que se quiere que sea desplegado, en este caso, información de un elemento.

Tabla: Servicios de la clase:TipText

Nombre	*Parametros*	*Tipo Salida*	*Objetivo*	*Discusion*
TipText	Lienzo lienzo MouseAdapterEscena adpatadorEscena	TipText	Método Constructor	
calculaPosicion	Rectangle r int ancho int alto	Point	Calcular la posición en la que debe ser colocado el ToolTip	Por defecto el tipo se coloca en la parte superior del componente seleccionado.
dibujaTip	Graphics2D g, Rectangle r{ String texto, int ancho, int alto			Dado un contexto grafico, se encarga de dibujar el tip en una posición determinada.

Panel: PnllEscena

Encargado de mostrar al usuario el ambiente y los elementos que se localizan en la casilla en la que se encuentra y con los que puede existir algún tipo de interacción. Emplea a la clase Lienzo para presentar las imágenes que componen a la escena en un solo gráfico en el que es posible identificar elementos. Este panel representa la visión que tiene el personaje del espacio en el que se encuentra, para lo que mantiene una colección de elementos (personajes e ítems) que deben presentarse en pantalla empleando el componente gráfico LienzoEscena.

Tabla: Servicios de la clase:PnlEscena

Nombre	*Parametros*	*Tipo Salida*	*Excepciones*	*Objetivo*	*Discusion*
PnlEscena		PnlEscena		Método constructor.	
PnlEscena	PnlVisor padre	PnlEscena		Método constructor que tiene una referencia al PnlVisor.	

Clase: LienzoEscena.

Empleado por el panel PnlEscena para la representación grafica de la casilla en la que se encuentra un personaje. Extiende a la clase Lienzo y cuenta con los adaptadores necesarios para manejar todos los eventos del mouse generados por el usuario, empleando la clase MouseAdapterEscena.

Se encarga de desplegar los elementos de la casilla en su propio contexto grafico. Antes de pintar se verifica que los elementos a representar no hayan cambiado, en cuyo caso se actualizan los valores tanto de la colección de elementos contenidos en PnlEscena, como de los rectangulos delimitadores de dichos elementos, y se dibujan las imágenes relacionadas con cada objeto en la colección.

Tabla: Servicios de la clase:LienzoEscena

Nombre	*Parametros*	*Tipo Salida*	*Objetivo*	*Descripcion*
LienzoEscena	PnlEscena escena	LienzoEscena	Método Constructor	
paint	Graphics g		Se encarga de desplegar los elementos de la casilla en su propio contexto grafico.	Antes de pintar se verifica que los elementos no hayan cambiado.En caso de que hayan cambiado se actualizan los valores tanto de la coleccion de elementos contenidos en la escena, asi como de los rectangulos delimitadores de dichos elementos y se dibujan.

Clase: MouseAdapterEscena.

Permite la gestión de eventos relativos al ratón cuando el panel visor del cliente se encuentra en modo Escena. Extiende a la clase AdaptadorMouse, con lo que hereda la funcionalidad para identificar y seleccionar elementos de juego. Incluye la funcionalidad para obtener información de los elementos presentados, misma que se contiene en cuadros de información (haciendo uso de la clase TipText) y se presenta al usuario cuando posiciona el puntero del ratón sobre determinado elemento.

Tabla: Servicios de la clase:MouseAdapaterEscena

Nombre	Parametros	Tipo Salida	Objetivo	Discusion
MouseAdapterEscena	Lienzo lienzo	MouseAdapterEscena	Método Constructor.	
mouseMoved	MouseEvent e		Gestion de eventos cuando el puntero del mouse se mueve por la escena.	Si el puntero del mouse se encuentra sobre un elemento, se debe dibujar un ToolTip con la descripción de dicho elemento.

Panel: PnlPersonaje.

Encargado de mostrar al usuario la información del personaje que tiene asociado, incluyendo su perfil y su estado. Emplea a la clase CuadroInformacion para organizar la información que se presenta.

Nombre	Parametros	Tipo Salida	Excepciones	Objetivo	Efecto Colateral	Discusion

Clase: CuadroInformación.

Organiza y crea una representación gráfica de la información de un personaje. Es utilizada por los páneles PnlPersonaje y PnlBatallas para que el usuario consulte los datos de su representación en el juego así como el estado de sus atributos, variando entre las dos en los datos presentados.

Tabla: Servicios de la clase:CuadroInformacion

Nombre	Parametros	Tipo Salida	Objetivo	Discusion
Cuadro Informacion	int iX, int iY int iAncho, int iAlto, Personaje p int iModo	CuadroInformacion	Método Constructor	
Cuadro Informacion	Rectangle r Personaje p		Método Constructor	
paint	Graphics2D g2d		pintar la informacion relativa a un personaje.	

Nombre	Parámetros	Tipo Salida	Objetivo	Descripción
paint	Graphics2D g2d		Método que dibuja solamente la inforamció pertinente. .	Debido a que este componente depende de otro. Este metodo tan solo manda llamar al metodo paint(Graphics2D) para que se dibuje la información pertinente
pintaContenido Cuadro			Se pinta propiamente la informacion del personaje.	En caso de que sea Modo Batalla sólo se pinta Puntos de Vida y Puntos Magia.
pintaFondo			Se pinta el fondo del cuadro en el que se dibuja la informacion del personaje	
pintaRectangulo			Se pinta el rectangulo delimitador de la informacion del personaje.	

Panel: PnlBatalla.

Encargada de mostrar en el visor del cliente la información relativa a las batallas. Organiza varios componentes gráficos como sigue:

- La clase LienzoBatalla para presentar las imágenes e información de la batalla.
- La clase LienzoBatallaMenu para crear y dar funcionalidad a menús de acciones proporcionados al jugador para generar acciones de batalla.

- El panel PnlMana para establecer y graficar el contador con que se administran los turnos.
- El panel PnlMensajeBatallas para desplegar en pantalla la información de las acciones realizadas en la batalla.
-

Tabla: Servicios de la clase:PnlBatalla

Nombre	Parámetros	Tipo Salida	Objetivo	Efecto Colateral
PnlBatalla		PnlBatalla	Método Constructor	
PnlBatalla	PnlVisor padre	PnlBatalla	Método constructor que tiene una referencia al PnlVisor.	
inicializa Menus			Realiza las funciones de mantenimiento de los componentes gráficos utilizados durante la batalla.	Se crean los Menus (LienzoBatallaMenu) y sus sub-menus. De la misma forma se agregan los personajes que tomaran parte en la batalla.

Clase: LienzoBatalla.

Empleado por el panel PnlBatallas, se encarga de organizar y presentar gráficamente la información derivada de una batalla, que incluye menús y submenús de acciones, cuadros de información de los personajes, imágenes de los personajes y rectángulos delimitadores de selección. Emplea a la clase CuadroInformacion para presentar los datos del personaje útiles para el desarrollo de una batalla.

Extiende a la clase Lienzo y cuenta con los adaptadores necesarios para manejar todos los eventos del mouse generados por el usuario, empleando la clase MouseAdapterBatalla.

Tabla: Servicios de la clase:LienzoBatalla

Nombre	*Parametros*	*Tipo Salida*	*Objetivo*	*Descripcion*
LienzoBatalla		LienzoBatalla		Se añade una instancia del MouseAdapter del padre como un objeto MouseListener.
LienzoBatalla	int iX, iY, iAncho, iAlto	LienzoBatalla		
agregaContenidoAMenu	String menu, Collection cont		Dado un menu en particular, se agregan los elemento seleccionabless que contiene.	
agregaMenu	String desc, LienzoBatalla m		Agrega un menú en el Lienzo de Batalla	
agregaPersonajes	Personaje p, Personaje e			Se instancian CuadroInformacion para que contengan información pertinente a los personajes en batalla
estableceContenido	Collection colElementos		Se actualizan los valores de los elementos en batalla,	Solo se actualizan Personajes.
estableceMenuActivo	String strMenu		Delega la responsabilidad de tratar con eventos del mouse al SubMenu seleccionado.	
obtenMenúActivo		LienzoBatalla	Se obtiene el Menu activo (Instancia de LienzoBatalla).	
obtenMouseAdapter		MouseInputAdapter	Obtener el MouseAdapter.	Se obtiene el manejador de eventos del mouse para validar los eventos generados por el cliente.
pintaMarco	Graphics2D g2d, Rectangle2D rect, Stroke stroke, Color color		Pinta el marco de algun componente dentro del juego.	
pintaContenidoMenu	Graphics2D g2d		Pinta el contenido de cada uno de los menus.	
pintaCuadro	Graphics2D g2d			

Clase: MouseAdapterBatalla.

Permite la gestión de eventos relativos al ratón cuando el panel visor del cliente se encuentra en modo Batalla. Extiende a la clase AdaptadorMouse, con lo que hereda la funcionalidad para identificar y seleccionar elementos de juego. Incluye la funcionalidad para manejar los menús y submenús así como para seleccionar objetivos para las acciones.

Tabla: Servicios de la clase:MouseAdapaterBatalla

Nombre	*Parametros*	*Tipo Salida*	*Objetivo*	*Efecto Colateral*	*Discusion*
MouseAdapterBatalla	Lienzo lienzo	MouseAdapterBatalla			
mouseMoved	MouseEvent e		Gestiona eventos si el puntero esta sobre un menú, sub-menú o un personaje.		
mousePressed	MouseEvent e		Determina lo que debe hacerse en caso de que se haya presionado el boton del mouse sobre un elemento valido.	Selecciona un elemento de la casilla afectando a la clase Administrador Cliente, PnlSeleccion y PnlComandos.	En caso de que se haya hecho clic sobre unpersonaje, se delega la responsabilidad al mismo metodo de la clase padre. De lo contrario, se invoca al metodo determinaAccion de esta clase.
determinaAccion			Determina que es lo que debe hacerse en caso de que el usuario haya hecho clic sobre un menu o sub-menu.		

Clase: LienzoBatallaMenu.

Empleado por el panel PnlBatallas, se encarga de representar cada uno de los menus y SubMenus que se despliegan al usuario durante las batallas. Extiende la funcionalidad de lienzo de batrallas. Maneja una estructura jerárquica de menús y submenús para presentar al jugador las opciones que puede seleccionar así como para seleccionar los ítems y magias con que cuenta.

Tabla: Servicios de la clase:LienzoBatallaMenu

Nombre	Parametros	Tipo Salida	Objetivo	Precondicion
LienzoBatalla Menu		LienzoBatallaMenu	Método constructor.	
LienzoBatalla Menu	LienzoBatalla jcPadre, int iX, int iY, int iAlto, int iAnchoestablecAlto	LienzoBatallaMenu	Método Constructor.	
getGraphics		Graphics	Obtiene el contexto en el que se debe dibujar este menu..	Como existe dentro de un Menu (Menu Padre), entonces debe utilizar el contexto del Menu Padre
paint	Graphics g		Pinta el menú.	
obtenMenuActivo		LienzoBatalla	Se obtiene el Sub-Menu que depende de este.	cada menu solo cuenta con un nivel de profunidad, por lo que cada Sub-Menu regresa nulo.

Panel: PnlMensajeBatallas.

Empleado por el panel PnlBatallas, se encarga de mostrar al jugador la bitácora de actividades relacionadas con la batalla en curso. Despliega toda la informacion de las acciones que un usuario realiza dentro de una pelea. Tiene el propósito de informar de las acciones que tanto el cliente como su oponente realizan, para lo cual observa al Administrador de batallas para actualizar los mensajes presentados, traduciendo las acciones en texto formateado para la lectura del cliente.

Nombre	Parametros	Tipo Salida	Objetivo	Efecto Colateral
PnlMensajeBatallas		PnlMensajeBatallas	Método Constructor	Se declara observador de la clase AdministraBatallas.
Inicializa			Dado un mensaje de batalla (acción realizada) se despliega la información respectiva a dicho cambio.	
udate	Observable ob Object cambio		Se actualiza la informacion que se despliega en el cuadro.	

Panel: PnlMana.

Empleado por el panel PnlBatallas, maneja y grafica el contador que administra los turnos de acción en los que se lleva a cabo la batalla. Objeto encargado de llevar una cuenta de los puntos de mana (capacidad para realizar acciones) que un usuario ha utilizado a lo largo de un turno. Cada accion realizada por el usuario tiene un costo de Mana. Al utilizar una accion, su mana se reduce.

Esto con el objeto de limitar las acciones que los usuarios puedan realizar y la favorecer la estrategia en cuanto a ataque/defensa.

De igual forma, para dar agilidad al juego, se penaliza con el paso del tiempo los puntos de mana disponibles por el usuario, para lo que se emplea la clase interna HazAccion (que extiende a TimerTask), encargada de afectar el valor del mana conforme pasa el tiempo por medio del manejo de threads.

Tabla: Servicios de la clase:PnlMana

Nombre	*Parametros*	*Tipo Salida*	*Excepciones*	*Objetivo*	*Efecto Colateral*	*Discusion*
PnlMana		PnlMana		Método Constructor		
estableceManaDisponible	int iMana			Se establece la cantidad de mana disponible del usuario en un momento dado.		
Finaliza				Cancela el timer utilizado durante una batalla..		Método de limpieza para asegurar el buen funcionamiento del componente
inicializaValores				Se establecen los valores iniciales del componente cuando se entra a una batalla		Se inicializa el timer y se le da un tiempo para la ejecución de magias
obtenManaDisponible		int		Se obtiene el mana disponible de un usuario en un momento dado.		
registraAccion				Registra las acciones realizadas por el usuario actualizando la cantidad disponible de Mana con que cuenta un usuario.	Se avisa al usuario de la insuficiencia de Mana y se cancela su timer	

Servicios de la clase interna HazAccion

Tabla: Servicios de la clase:HazAccion

Nombre	*Tipo Salida*	*Objetivo*	*Efecto Colateral*
HazAccion	HazAccion	Método Constructor	
Run		Verifica que el mana no es suficiente para realizar más acciones.	Despliega un mensaje de mana insuficiente.

Ventana principal de juego: Panel de mensajeria.

Tabla: Listado de clases administradoras PnlMensajeria.

Nombre	*Responsabilidad*	*Paquete*	*Discusión*
PnlMensajeria	Panel donde se escriben y se visualizan los mensajes escritos de los personajes activos en el Juego y mensajes del Servidor.	*Tonalli.cliente.ui.mensajeria*	*Por medio de la clase AdminstraMensajeria envia y recibe mensajes. Contiene un PnlEntrada para escribir mensajes y un PnlSalida para desplegar los mensajes enviados.*
DiAyudaSintaxis	Panel que muestra una ayuda de sintaxis.	*Tonalli.cliente.ui.mensajeria*	*El texto se encuentre en formato HTML.*
Mensaje	Clase que sirve como medio de comunicación.	*Tonalli.cliente.acciones*	*Clase Objeto Valor que personaliza el texto a desplegar.*
MensajeBatalla	Clase que sirve como medio de comunicación en modo Batalla.	*Tonalli.cliente.acciones*	*Subclase de Mensaje. Define el origen y destino como Personajes y el mensaje como una acción.*

Panel: PnlMensajeria.

Muestra los componentes que permiten al cliente ingresar y visualizar mensajes. Cuando el jugador ingresa un texto, esta componente crea un mensaje que puede ser evaluado y manejado por el administrador de mensajería. Esta componente simula una aplicación tipo chat en donde se envian textos de comunicación, agregando la funcionalidad al ingreso de comandos escritos que son traducidos a acciones de juego. Para mantener actualizados los mensajes presentados se vale del administrador de mensajería de donde lee la cola de mensajes actualmente enviados y filtra los que realmente deben ser mostrados en pantalla.

Tabla: Servicios de la clase:PnlMensajeria

Nombre	*Parametros*	*Tipo Salida*	*Objetivo*	*Efecto Colateral*	*Discusion*
PnlMensajeria		PnlMensajeria	Método Constructor el cual crea el Panel que servirá como canal de comunicación entre el usuario y otros personajes.		
actualizaEntrada Mensajes	Mensaje mensajeRecibido		Despliega los mensjes enviados.		
añadeTeclasRapidas			Añade teclas rápidas al componente.	CTRL.-Enter es para enviar un mensje	
asignaColor			Personaliza el color del mensaje.		
enviaMensaje			Envía mensajes.		Identifica si es un mensaje o una acción. Se actualiza el PnlSalida del PnlMensajeria.
muestraSintaxis			Despliega un Diálogo de Ayuda de Sintaxis.		
realizaAccion	Mensaje mensaje		Ejecuta un comando .		
update	Observable o Object arg				

Dialogo: DiAyudaSintaxis.

Utilizado en el panel PnlMensajeria, muestra, en formato HTML, la información de ayuda sobre la sintaxis para escribir comandos que serán enviados como mesnsajes de texto. Contiene funciones de hipertexto para lo cual implementa la interfaz HyperLinkEvent.

Tabla: Servicios de la clase:DiAyudaSintaxis

Nombre	Parametros	Tipo Salida	Objetivo	Excepción
DiAyudaSintaxis		DiAyudaSintaxis	Método Constructor que despliega una ayuda para la sintaxis de comandos.	El texto viene en formato HTML.
hyperlinkUpdate	HyperLinkEvent event		Método de la Interfaz HyperLinkListener que permite la interacción entre ligas HTML	
textoComandos			Define el documento de ayuda con extensión HTMl	

Clase: Mensaje.

Define un mensaje escrito por el jugador. Los mensajes pueden ser de dos tipos: texto o comandos. Esta clase define la estructura del mensaje convirtiéndolo en un objeto manejable por la aplicación y es capaz de informar de qué tipo de mensaje se trata.

Tabla: Servicios de la clase:Mensaje

Nombre	Parametros	Tipo Salida	Objetivo
Mensaje		Mensaje	Método Constructor Vacío
Mensaje	String mensaje String origen Color color	Mensaje	Método Constructor que define el texto del mensaje, el origen y el color.
Mensaje	String mensaje String origen	Mensaje	Método Constructor que define el texto del mensaje y el origen
Mensaje	String mensaje	Mensaje	Método Constructor que define el texto del mensaje.
estableceMensaje	String mensaje		Define el texto del mensaje.
estableceOrigen	String origen		Define el origen del Mensaje.
obtenColor		Color	Obtiene el color del Mensaje.
obtenMensaje		String	Obtiene el texto del Mensaje.
obtenOrigen		String	Obtiene el Origen del Mensaje.
esComando		boolean	Valida si se trata de un comando escrito.

Clase: MensajeBatalla.
Define un mensaje enviado en batalla. Los componentes del panel que muestra el desarrollo de las batallas generan este tipo de mensajes.

Tabla: Servicios de la clase:MensajeBatalla

Nombre	*Parametros*	*Tipo Salida*	*Objetivo*	*Discusion*
MensajeBatalla		MensajeBatalla	Método Constructor Establece el origen como el Personaje del cliente.	Define el origen y el e destino del mensaje como Personajes
MensajeBatalla	Personaje destino Elemento accion	Mensaje	Método constructor que define el personaje destino y la accion a ejecutar.	
obtenAccion		Elemento	Obtiene la accion a ejecutar.	
obtenDestino		Personaje	Obtiene el personaje destino de la acción.	

Ventana principal de juego: Panel de comandos.

Tabla: Listado dePnlComandos.

Nombre	***Responsabilidad***	***Paquete***	***Discusión***
PnlComandos	*Panel Contenedor de todos los demás paneles. Permite al usuario generar eventos.*	*Tonalli.cliente.ui. comandos*	
PnlBotonesExploracion.	Incluye acciones para el modo exploración como tomar, retar, salir..	*Tonalli.cliente.ui. comandos*	*Obtiene de la clase AdministradorCliente los objetos en selección para invocar métodos de la clase Acciones.*
PnlBotonesBatalla.	Incluye acciones para el modo batalla como atacar, defender.	*Tonalli.cliente.ui. comandos*	
PnlBotonesModo	Incluye acciones comunes a ambos modos. Especificamente a la manipulación de magias e times.	*Tonalli.cliente.ui. comandos*	*Afecta a la clase PnlTablero al invocar métodos para mostrar el rango de una magia. Obtiene de la clase AdministradorCliente los objetos en selección para invocar métodos de la clase Acciones.*

Panel: PnlComandos.

Agrupa a los botones y cuadros desplegables por medio de los cuales el cliente puede realizar acciones. Este panel varía de acuerdo al modo de juego del cliente mostrando un grupo de botones adecuado a las acciones válidas ya sea en exploración o en batalla. Para realizar la actualización, su componente PnlBotonesModo observa a el Administrador del cliente y cambia su presentación adecuándola al modo actual.

Tabla: Servicios de la clase:PnlBotonesComando

Nombre	*Parametros*	*Tipo Salida*	*Objetivo*	*Efecto Colateral*	*Discusion*
PnlBotonesComando		PnlBotonesComando	Método Constructor.	Se declará observador de la clase AdministradorCliente.	Obtiene el Inventario y el ConocimientoMagico del Personaje.
actualizaItems			Actualiza la lista de Items del Personaje a nivel gráfico.		
actualizaMagias			Actualiza la lista de Magias del Personaje a nivel gráfico.		
update	Observable ob,Object o		Determina la lista de Items o Magias a actualizar.		
aplicar_actionPerformed	ActionEvente		Ejecuta la acción Aplicar de la Clase acciones.		Obtiene el personaje seleccionado de la clase AdministradorCliente.
darItem_actionPerformed	ActionEvente		Ejecuta la acción DarTurno de la Clase acciones.		Obtiene el personaje e item seleccionado de la clase AdministradorCliente
descItem_actionPerformed	ActionEvente		Despliega un Diálogo con la descripción del Item seleccionado.		
descMagia_actionPerformed	ActionEvente		Despliega un Diálogo con la descripción de la Magia seleccionada.		
item_actionPerformed	ActionEvente		Establece que item esta seleccionado en el Combo Box.	La clase Administrador Cliente cambia el foco del Item seleccionado.	
muestraRango_actionPerformed	ActionEvente		Resalta las casillas afectadas por un rango de aplicación de magia en el PnlTablero.	Se invoca el método dibujaCuadriculadel PnlTablero.	
usar_actionPerformed	ActionEvente		Ejecuta la acción DarTurno de la Clase acciones.		Obtiene el personaje seleccionado de la clase AdministradorCliente.

Panel: PnlBotonesModo.

Empleado por el panel PnlComandos. Incluye dos componentes alternativos en su presentación, haciendo uso de la distribución CardLayout, que permite el intercambio de la vista del componente. De acuerdo al modo de juego en que se encuentre el cliente permuta entre los paneles PnlBotonesBatalla y PnlBotonesExploracion, para lo cual observa al administrados del cliente, en espera de actualizaciones en el modo.

Tabla: Servicios de la clase:PnlBotonesModo

Nombre	*Parámetros*	*Tipo Salida*	*Objetivo*	*Efecto Colateral*
PnlBotonesModo		PnlBotonesModo	Método Constructor que contiene los paneles de comandos.	Se establece como observador de la clase Administrador Cliente
update	Observable ob Object cambio		Método que define el modo en que se hallará el Panel de Comandos: Explorción o Batalla.	

Panel: PnlBotonesBatalla.

Empleado por el panle PnlBotonesModo. Incluye los botones para generar comandos de batalla. Cuando uno de los botones es pulsado, valida la acción y llama al método correspondiente de la clase Acciones, tomando del administrador del cliente los parámetros que completan la llamada al método (usualmente los elementos en foco).

Nombre	*Parámetros*	*Tipo Salida*	*Objetivo*	*Discusión*
PnlBotonesBatalla		PnlBotones Batalla	Método Constructor.	
huir_actionPerformed	ActionEvent e		Ejecuta la acción Huir de la Clase acciones.	Muestra un dialogo de decisión para identificar si huye o no.
atacar_actionPerforme d	ActionEvent e		Ejecuta la acción atacar de la Clase acciones.	
cederTurno_actionPer formed	ActionEvent e		Ejecuta la acción CederTurno de la Clase acciones.	
defender_actionPerfor med	ActionEvent e		Ejecuta la acción Defender de la Clase acciones.	

Panel PnlBotonesExploracion.

Empleado por el panel PnlBotonesModo. Incluye los botones para generar comandos de exploración. Cuando uno de los botones es pulsado, valida la acción y llama al método correspondiente de la clase Acciones, tomando del cliente los parámetros que completan la llamada al método (usualmente los elementos en foco).

Tabla: Servicios de la clase:PnlBotonesExploracion

Nombre	*Parametros*	*Tipo Salida*	*Objetivo*	*Discusion*
PnlBotonesExploracion		PnlBotonesExploracion	Método Constructor.	
mover_actionPerformed	ActionEvent e		Ejecuta la acción Mover de la Clase acciones.	Obtiene la casilla seleccionada de la clase AdministradorCliente, el cual es enviado como parámetro al método Mover de la clase Acciones.
retar_actionPerformed	ActionEvent e		Ejecuta la acción retar de la Clase acciones	
salir_actionPerformed	ActionEvent e		Ejecuta la acción Salir de la Clase acciones	
tomarItem_actionPerformed	ActionEvent e		Ejecuta la acción Tomar de la Clase acciones.	Obtiene la casilla seleccionada de la clase AdministradorCliente.

Descripción de la capa de negocio.

La capa de negocio del sistema incluye todos los servicios que son comunes a los clientes y que deben ser concentrados para su acceso remoto. Este nivel, en el presente proyecto, esta compuesto principalmente por los beans de sesión que sirven de nexo entre las solicitudes del cliente y la información contenida en la base de datos. Por ejemplo, cuando un cliente requiere información de la base de datos debe solicitarla al bean de sesión que ofrece el servicio, mismo que, a su vez, llamara al método correspondiente en el bean de entidad que administra dicha información y que tiene la capacidad de realizar solicitudes al manejador de bases de datos.

Esta capa sirve de puente entre los clientes y los servicios de datos, y es la encargada de modelar todo el conocimiento propio del negocio y es completamente independiente de la capa de servicios de usuario. Responde a peticiones del usuario (u otros servicios de negocios), aplicando procedimientos formales y reglas de negocio a los datos relevantes. Cuando la información reside en un servidor de bases de datos, garantiza los servicios de datos indispensables así como la aplicación de tareas y reglas de negocio, aislando al usuario de la interacción directa con la base de datos.

Una tarea de negocios es una operación definida por los requerimientos de la aplicación, mientras las reglas de negocio son políticas que controlan el flujo de las tareas. Ambas son encapsuladas en componentes lógicamente separados de la presentación y de las fuentes de información.
Las responsabilidades de este nivel son:

- Recibir entradas del cliente a través de la capa de presentación.
- Interactuar con la capa de acceso a datos y sus servicios para ejecutar las operaciones de negocios para los que la aplicación fue diseñada.
- Enviar el resultado procesado al la capa de presentación.

Para el caso práctico, la principal responsabilidad de esta capa es mantener actualizado el juego, en respuesta a las acciones realizadas, así como hacer posible a los jugadores que conozcan el estado actual de cada uno de los elementos que integran el juego. Lo anterior se debe a que el "negocio" del caso práctico es un videojuego en línea.

Toda la información necesaria para crear un juego se encuentra almacenada en la base de datos. La capa de negocio crea un juego residente en el servidor, con la información que recibe de la capa de acceso a datos y, al ocurrir actualizaciones importantes en su imagen del juego, solicita la persistencia de los cambios en la base de datos. Así, solo las modificaciones importantes requieren accesos a la información contenida en la base.

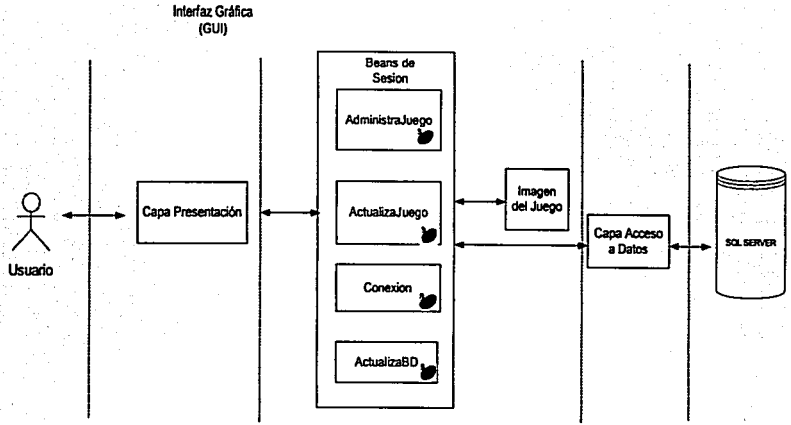


Ilustración 41 : Interacción con la capa de negocio.

Son dos los aspectos que se deben observar al estudiar la capa de negocio del caso práctico. Por un lado tenemos la imagen del juego que se crea para agilizar el acceso a la información reduciendo las consultas a la base de datos y por otro el conjunto de beans de sesión que administran y actualizan esa imagen.

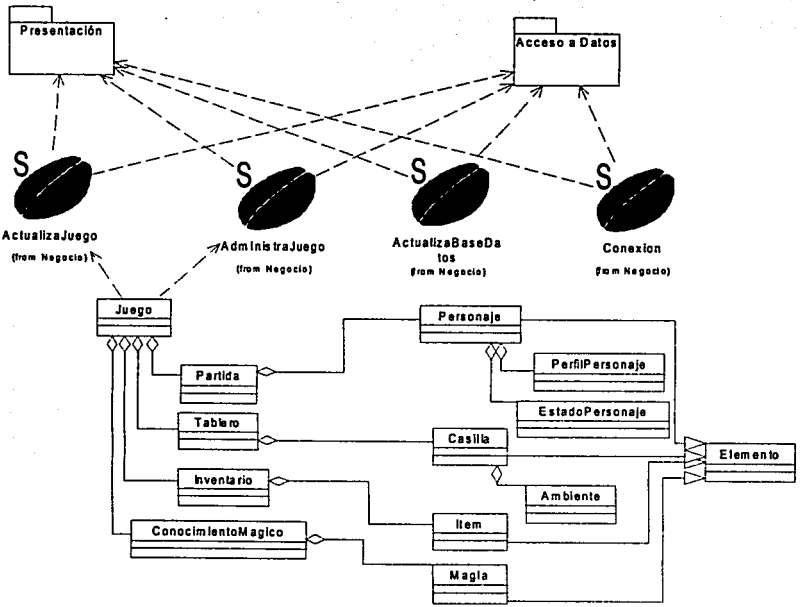


Ilustración 42 : Descripción de la capa de negocio.

Imagen del juego: Manejo de clases estáticas en el servidor.

Para propósitos generales de esta referencia es importante remarcar que el juego representa al objeto central del negocio para este caso práctico, por lo tanto, esta sección trata de la imagen del negocio manejando clases estáticas en el servidor.

La imagen del juego en la capa de negocio esta diseñada empleando una clase estática (clase Juego) residente en el servidor y accesada directamente por los beans de sesión, con lo que se logra que los cambios realizados en cualquier instancia de juego (i.e. desde cualquier cliente) afecten al juego en general.
Denominamos clase estática a aquella que esta compuesta por miembros estáticos. Si una clase incluye atributos estáticos, éstos se hacen independientes del objeto, es decir, si una instancia de una clase estática (objeto) es afectada en el valor de alguno de sus atributos estáticos, todos los objetos de esa clase serán afectados de la misma forma. Esta es la manera en que Java permite simular los valores globales en las variables.

Tomando en cuenta lo anterior, el juego (la clase Juego) esta compuesto por diversos conjuntos de elementos representados con variables estáticas, y que son afectados por diversas acciones de los jugadores. Las partes que integran a la clase Juego, a manera de atributos estáticos son:

Partida. Conjunto de personajes activos en el juego.
Inventario. Conjunto de ítems existentes en el juego.
ConocimientoMagico. Conjunto de magias válidas en el juego.
Tablero. Conjunto de casillas que representan el mundo o ambiente.

En resumen cada uno de los atributos de la clase estática representan una parte de la imagen del estado actual del juego en relación con los elementos que lo integran, mismos que pueden ser consultados y modificados de forma concurrente por todos los jugadores, manteniendo siempre la misma información independientemente de quien realice el cambio.

Nombre	*Responsabilidad*	*Paquete*	*Discusión*
Juego	Representa la imagen del juego activo, incluye toda la información que comparten los jugadores.	*Tonalli.cliente.estructura*	*Contiene atributos estáticos que definen cada aspecto sustancial del juego.*
Partida	Describe la lista de personajes reunidos para jugar. Contiene una colección de objetos Personaje.	*Tonalli.cliente.estructura*	*Se incrementa con cada acceso de clientes al sistema.*
Inventario	Describe los ítems que existen en el juego y con los que es posible interactuar. Contiene una colección de objetos Item.	*Tonalli.cliente.estructura*	*La colección de ítems refleja los almacenados (descritos) en la base de datos.*
ConocimientoMagico	Describe las magias que existen en el juego y que es posible usar. Contiene una colección de objetos Magia.	*Tonalli.cliente.estructura*	*La colección de magias refleja las almacenadas (descritas) en la base de datos*
Tablero	Incluye las casillas o zonas de juego que integran al mundo. Contiene una colección de objetos Casilla.	*Tonalli.cliente.estructura*	*La colección de casillas refleja las almacenadas (descritas) en la base de datos*

Clase: Juego.

Representa el juego integrado. Se construye inicialmente con la información almacenada en la base de datos, y se mantiene mientras no se alcance alguna de las condiciones de término del juego, modificándose de acuerdo con las acciones que generen los jugadores y el propio servidor.

Su utilidad radica en la reducción de llamadas a la capa de acceso a datos para obtener la información de la base de datos, con lo que se agiliza el proceso del juego al mantener la información en una clase accesible, de forma inmediata, a todos los componentes del servidor. Con este esquema separamos la información que solo afecta a el juego activo mientras que se inicia y se desarrolla y hasta que finaliza (por ejemplo, la posición actual de los items y personajes) de la que es significativa para la comunidad de jugadores independientemente del juego que esté jugando (por ejemplo la experiencia que ha adquirido un personaje) y que requiere persistencia más duradera, proporcionada por procesos alternos de almacenamiento en la base de datos.

El juego es, por lo tanto la imagen del mundo con sus elementos activos, información que solo es significativa mientras no se finalice el juego. En concreto el Juego esta integrado por una partida de jugadores; un inventario y un conjunto de magias (ConocimientoMágico) que pueden ser utilizados en el juego, y un tablero que representa el ambiente en el que los personajes puedes desarrollar sus acciones, los anteriores pueden verse como catálogos de información relevante para el juego. Esta clse no requiere capacidades de serialización ya que no viaja a través de la red, sin embargo sus atributos si implementan a la interfaz Serializable.

Clase: Partida.

Es la colección de Personajes reunidos para jugar. Esta clase se inicializa vacía en la creación del juego y, conforme se van registrando nuevos jugadores, sus personajes son agregados como elementos de la partida. Cada personaje incluido en la Partida del juego puede ser consultado, modificado y representado. Implementa a la interfaz Serializable lo que permite que sea enviada a través de la red como un objeto valor.

Clase: Inventario.

Define un conjunto de items u objetos utilizables en el juego. El inventario del juego define la totalidad de items disponibles, independientemente del personaje al que estén asociados o su posición en el mundo. Para poder utilizar un objeto inmerso en alguna de las acciones del cliente, el Juego debe conocerlo, y esta clase permite especificar cuales items existen en el universo creado para el juego, con lo que es posible su manejo y validación. La información para inicializar este conjunto se recibe de la base de datos donde se leen todos los registros de items existentes y se van agregando al inventario. Cada item incluido en el Inventario del juego puede ser consultado, modificado y representado.

Esta clase también es útil para asociar un grupo de items a un personaje en particular, siendo siempre ese nuevo conjunto un subconjunto del inventario del juego(i.e. todos los items asociados al personaje deben estar contenidos en el inventario del juego), esta funcionalidad se observará a detalle cuando se trate de la estructura de los personajes.

Implementa a la interfaz Serializable lo que permite que sea enviada a través de la red como un objeto valor.

Clase: ConocimientoMágico.

Define un conjunto magias practicables en el juego. Las magias se pueden definir como objetos intangibles que un jugador puede utilizar. El conocimiento mágico del juego define la totalidad de magias disponibles, independientemente del personaje al que estén asociados. Así, el ConocimientoMagico del Juego sirve a manera de catálogo donde se registran las magias conocidas y válidas La información para inicializar este conjunto se recibe de la base de datos donde se leen todos los registros de magias existentes y se van agregando al conocimiento mágico. Cada magia incluida en el ConocimientoMagico del juego puede ser consultada, modificada y representada.

Esta clase también es útil para asociar magias a un personaje en particular, siendo siempre ese nuevo conjunto un subconjunto del ConocimientoMágico del juego (i.e. todas las magias de un jugador deben estar en el "catalogo" de magias del juego), esta funcionalidad se observará a detalle cuando se trate de la estructura de los personajes.
Implementa a la interfaz Serializable lo que permite que sea enviada a través de la red como un objeto valor.

Clase Tablero.

Representa el ambiente o mundo del juego. El mundo, y por lo tanto el tablero, esta dividido en casillas, que representan una pequeña sección del mismo. Cada casilla tiene un ambiente, un color, coordenadas de localización y puede contener personajes e items. La información básica de las casillas (color, descripción, coordenadas) se extrae de la base de datos al crear el juego y posteriormente se posicionan sobre ellas a los elementos mencionados. El tablero es un mapa coordenado, donde los personajes pueden moverse para interactuar con los elementos que van encontrando. Cada casilla incluida en el Tablero del juego puede ser consultada, modificada y representada.

El manejo del tablero se trata a detalle en el apartado dedicado a su representación gráfica.

Implementa a la interfaz Serializable lo que permite que sea enviada a través de la red como un objeto valor.

Desarrollo de beans de sesión.

Los beans de sesión son aquellos componentes que actúan como agentes para el cliente, ya que son los encargados de controlar los procesos del negocio y servir de puente entre la capa de presentación (cliente) y la capa de datos (beans de entidad). Este tipo de beans se distinguen debido a que representan los verbos o acciones que se quiere realizar, a diferencia de los beans de entidad que, como se verá más adelante, son los encargados de representar las entidades sobre las que se va a trabajar.

Un bean de sesión, se caracteriza porque cuenta con un tiempo de vida relativamente corto, ya que generalmente, estos se encargan solamente de recibir una solicitud por parte de un cliente y regresar la respuesta; posteriormente, el bean se olvida del cliente que lo invocó y está listo para atender nuevas de peticiones a cualquier cliente que lo requiera. Generalmente se dice que el tiempo de vida de este tipo de beans, está determinado por el tiempo que dura la sesión de un cliente (de ahí su nombre de beans de sesión).

A diferencia de los beans de entidad, un bean de sesión no es persistente, es decir, que su estado no se almacena en ningún medio.

Los beans de sesión tienen dos subtipos: beans de sesión con estado (stateful session beans) y beans de sesión sin estado (stateless session beans). Esta división se hace con referencia a la **conversación** entre un cliente y un bean.

Esta conversación es la duración que tiene la comunicación entre un cliente y un bean. Esta comunicación debe ser entendida como todas las llamadas que realiza un cliente a los métodos del bean. Este estado conversacional puede ser tan simple como la invocación de un solo método, o tan compleja como la invocación a múltiples métodos, donde cada una de estas invocaciones modifica cierto estado que afecta los resultados de las llamadas subsecuentes (un carrito de compras).

Beans de sesión con estado.

Un bean de sesión con estado es un bean diseñado para atender a aquellos procesos del negocio cuya duración implica varias llamadas a métodos o transacciones, por lo que este bean debe almacenar el estado particular del cliente que lo invoca. Así mismo, este estado cambia con cada una de las llamadas que el cliente hace a los métodos del bean.

Así, un bean de sesión con estado está dedicado enteramente a un cliente durante todo su ciclo de vida, ya que la conversación mantenida entre el cliente y el bean abarca la invocación de múltiples métodos, donde cada método afecta el estado de dicha conversación. Así, podemos decir que existe un **estado conversacional** entre el bean y el cliente. Este estado conversacional se almacena en variables del bean y pueden ser accedidas por el cliente, ya que se garantiza que el mismo bean siempre atenderá al cliente al que fue asignado.

Sin embargo, a pesar de que este tipo de beans mantienen un estado relativamente permanente, este tipo de beans no son persistentes[42], es decir, este estado no se almacena en una Base de Datos, sino que al término del estado conversacional, el bean puede ser reutilizado para servir a un nuevo cliente y por consiguiente, contener un estado distinto.

Generalmente, este tipo de beans pueden verse como extensiones del cliente, esto tiene sentido si se piensa en el cliente como un conjunto de operaciones y estados, donde, como ya se ha mencionado. cada operación depende de la información almacenada o modificada por las operaciones previas.

Al extender al cliente mediante el uso de estos beans, se logran encapsular mejor la lógica del negocio y contar con aplicaciones cliente más ligeras y eficientes.

Beans de sesión sin estado.

Un bean de sesión sin estado es aquel que mantiene conversaciones cuya duración abarca la invocación a un solo método. Después de esta invocación este bean puede ser utilizado por otro cliente para satisfacer una nueva petición.

Este tipo de beans se utiliza para implementar servicios genéricos y reutilizables, así, todo lo que un método requiere saber para funcionar se encuentra en los parámetros que recibe. En contraste con los beans de sesión con estado, este tipo de beans no pueden recordar nada de una invocación de un método a otra.

De igual forma, los beans de sesión sin estado no están dedicados a un solo cliente en particular debido a que no mantienen un estado conversacional con el cliente. Así, para un cliente, no existe diferencia entre las múltiples instancias de un bean de sesión sin estado que existan dentro del contenedor.

Sin embargo, cabe hacer la aclaración de que a pesar de que este tipo de beans no mantenga un estado conversacional, nada impide que contengan variables de instancia o mantengan un estado interno, siempre y cuando se tenga en cuenta de que este estado no puede ser visible para el cliente, ya que nada garantiza que el mismo bean de sesión sin estado atenderá todas las peticiones del mismo cliente.

Construcción del módulo de beans de sesión.

Ya que se conocen las diferencias entre los dos tipos de beans de sesión, ahora explicaremos la forma de crear los beans de sesión.

Clases e Interfaces.

Como se ha visto en apartados anteriores de este documento, un cliente no se comunica de forma directa con un bean, sino que debe primero obtener una referencia a un objeto de tipo EJBHome para crear un objeto de tipo EJBObject que es quien realmente se encarga de transmitir las peticiones del cliente al bean que se encuentra dentro del contenedor en el servidor. Así, tenemos la siguiente estructura:

[42] Para efectos de persistencia de información existen los beans de entidad.

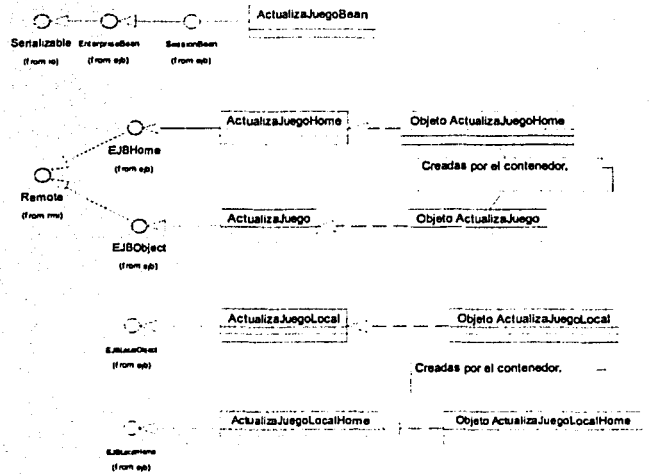


Ilustración 43 : Diagrama de clases de los beans de sesión. Ejemplo.

Esto puede parecer complejo en un principio, sin embargo, la implementación de los beans es muy sencilla ya que siempre se deben seguir los mismos pasos para su codificación como se verá a continuación.

Para efectos de este ejemplo se toma el bean `ActualizaJuegoBean` que es el encargado de actualizar el estado que tiene el juego en un momento dado, ya sea actualizando la posición de un elemento (personaje o ítem), actualizando el inventario de algún personaje, etc.

Interfaz Remota. La interfaz remota es aquella que se encarga de duplicar cada uno de los métodos del negocio que el bean expone al cliente. Además, esta interfaz debe extender la clase `javax.ejb.EJBObject` y cada uno de los métodos incluidos debe lanzar la excepción `java.rmi.RemoteInterface`, ya que esta permite indicar si existe algún error en la red o un error crítico y permita realizar las gestiones necesarias.

Como nota adicional, cabe hacer mención de que todos los parámetros pasados a estos métodos, así como los valores de retorno, deben cumplir con el requisito de implementar la interfaz `java.io.Serializable` ya que de lo contrario se pueden tener problemas al intentar pasar dichos objetos por la red. Todos los tipos de datos primitivos, así como las clases envolventes de estos implementan esta interfaz.

```
package tonalli.servidor;

import tonalli.estructura.elementos.Personaje;
import tonalli.estructura.elementos.Item;

import javax.ejb.*;
import java.util.*;
import java.rmi.*;

public interface ActualizaJuego extends javax.ejb.EJBObject {

    public void colocaPersonaje(String nombre, int coordX, int coordY)
                throws RemoteException;
    public void colocaItem(String nombre, int coordX, int coordY)
                throws RemoteException;
    public void sacaPersonaje(String nombre) throws RemoteException;
    public void sacaItem(String nombre) throws RemoteException;
    public void muevePersonaje(String nombre, int destinoX, int
                                destinoY)
                throws RemoteException;
    public void aplicaEfecto(Collection personajes, int[] efecto)
                throws RemoteException;
    public void eliminaPersonaje(String nombre) throws RemoteException;
    public void eliminaItem(String nombre) throws RemoteException;
    public void asociaItem(String nombrePersonaje, String nombreItem)
                throws RemoteException;
    public void asociaMagia(String nombrePersonaje, String nombreMagia)
                throws RemoteException;
    public void desasociaItem(String nombrePersonaje, String
                                nombreItem)
                throws RemoteException;
    public void desasociaMagia(String nombrePersonaje, String
                                nombreMagia)
                throws RemoteException;
}
```

Interfaz Home. La interfaz Home es aquella que contiene los métodos necesarios para poder crear y destruir objetos del tipo `javax.ejb.EJBObject`. Esta interfaz debe extender a la clase `javax.ejb.EJBHome` y debe incluir en el caso de los beans de sesión un método `create()` cuyo valor de regreso debe ser del tipo `javax.ejb.EJBObject`; de igual forma este método debe lanzar la excepción `java.rmi.RemoteException` y la excepción `javax.ejb.CreateException`. Esta última indica si ha existido algún problema al momento de crear el objeto remoto con el que el cliente se comunicará.

```
package tonalli.servidor;

import javax.ejb.*;
import java.util.*;
import java.rmi.*;

public interface ActualizaJuegoHome extends javax.ejb.EJBHome {
    public ActualizaJuego create()
                throws CreateException, RemoteException;
}
```

Como puede observarse, la interfaz `ActualizaJuegoHome`, sólo implementa el método `create()`, el cual regresa un objeto de tipo `ActualizaJuego` (que es a su vez, del tipo `javax.ejb.EJBObject`).

Clase del Bean. Ya que contamos con las interfaces necesarias para que el cliente pueda hacer uso de los métodos del negocio, debemos ahora codificar la clase que realmente implementa la funcionalidad de los métodos definidos en la clase `ActualizaJuego`. Esta clase es la clase del bean y debe cumplir con los siguientes requisitos:

- Debe implementar la interfaz `javax.ejb.SessionBean`. Esta interfaz define ciertos métodos que son utilizados por el contenedor para dar mantenimiento al bean (se exponen en la siguiente tabla).
- Debe contener un método con la siguiente firma `public void ejbCreate()`, el cual es llamado por el contenedor cuando se invoca al método `create()` en la interfaz EJBHome. La utilidad de este método radica en que si existe algo que deba inicializarse se hace aquí.
- De implementar todos los métodos del negocio definidos en la interfaz `EJBObject`, sólo que en este caso, no se deben lanzar excepciones del tipo `java.rmi.RemoteException`.

```
package tonalli.servidor;

import tonalli.estructura.Juego;
import tonalli.estructura.elementos.Casilla;
import tonalli.estructura.elementos.Personaje;
import tonalli.estructura.elementos.Item;
import tonalli.estructura.elementos.Magia;
import java.util.*;
import javax.ejb.*;
import javax.naming.*;
import javax.rmi.*;

public class ActualizaJuegoBean implements SessionBean {

    SessionContext sessionContext;
    public Juego _juego;
    private ActualizaBDHome actualizaBDHome = null;
    private ActualizaBD actualizaBDRemote = null;

    /*Metodos requeridos por el contenedor*/

    public void ejbCreate() throws CreateException {
        try{
            Context contexto = new InitialContext();
            Object ref = contexto.lookup("ActualizaBD");
            this.actualizaBDHome = (ActualizaBDHome)
                PortableRemoteObject.narrow(ref,ActualizaBDHome.class);
            this.actualizaBDRemote = actualizaBDHome.create();
        }catch(java.rmi.RemoteException re){
              throw new CreateException("Remote:No se creo Bean" +
                                        "ActualizaJuego:"+re.toString());
        }catch(javax.naming.NamingException ne){
              throw new CreateException("Naming:No se creo Bean " +
                                        "ActualizaJuego"+ne.toString());
        }
    }

    public void ejbRemove() {}
    public void ejbActivate() {}
    public void ejbPassivate() {}
```

```
    public void setSessionContext(SessionContext sessionContext) {
        this.sessionContext = sessionContext;
    }

    /*Metodos del negocio, idénticos a los encontrados en la interfaz
    ActualizaJuego, para efectos practicos no se coloca la
    implementación.*/

    public void colocaPersonaje(String nombre, int coordX, int coordY){
       ...
    }
    public void sacaPersonaje(String nombre) {
       ...
    }
    public void sacaItem(String nombre) {
       ...
    }
    public void muevePersonaje(String nombre, int destinoX, int
                               destinoY) {
       ...
    }
    public void aplicaEfecto(Collection personajes, int[] efecto) {
       ...
    }
    public void eliminaPersonaje(String nombre) {
       ...
    }
    public void asociaMagia(String nombrePersonaje, String
                            nombreMagia){
       ...
    }
    public void desasociaItem(String nombrePersonaje, String
                              nombreItem){
       ...
    }
    public void desasociaMagia(String nombrePersonaje, String
                               nombreMagia){
       ...
    }
}
```

Interfaces Locales.

Las interfaces explicadas anteriormente (EJBObject y EJBHome) implican que el acceso al bean se hace desde una aplicación remota (un cliente u otro bean), pero ¿qué sucede si tengo dos beans dentro del mismo contenedor y deseo que se comuniquen entre ellos? En este caso, la comunicación remota entre dos beans que se encuentran en el mismo contenedor utilizando sus interfaces remotas, sería engorrosa e implicaría demasiado trabajo para el servidor, por esta razón a partir de la especificación 2.0 de EJB, se han incluido las **interfaces locales**, mismas que permiten reducir el pre-procesamiento involucrado en las llamadas remotas a los beans, agilizando el proceso y la carga de trabajo para el servidor.

Debe quedar claro que el uso de interfaces locales y remotas no es excluyente, de hecho es recomendable que se incluyan ambas. Debido a que independientemente de si se accede a un bean por cualquiera de sus interfaces (local o remota), el bean no cambia, sino que simplemente se añaden dos interfaces más:

Interfaz Local. Las diferencias existentes entre la interfaz local y la remota son triviales.

- Debe extenderse la clase `javax.ejb.EJBLocalObject` y
- Las firmas de los métodos del negocio no deben lanzar la excepción `java.rmi.RemoteException`, ya que no se está haciendo uso de la red para comunicarse con el bean.

```
package tonalli.servidor;

import tonalli.estructura.elementos.Personaje;
import tonalli.estructura.elementos.Item;

import javax.ejb.*;
import java.util.*;
import java.rmi.*;

public interface ActualizaJuegoLocal extends javax.ejb.EJBLocalObject {

    public void colocaPersonaje(String nombre, int coordX, int coordY);
    public void colocaItem(String nombre, int coordX, int coordY);
    public void sacaPersonaje(String nombre);
    public void sacaItem(String nombre);
    public void muevePersonaje(String nombre, int destinoX, int
                               destinoY);
    public void aplicaEfecto(Collection personajes, int[] efecto);
    public void eliminaPersonaje(String nombre);
    public void eliminaItem(String nombre);
    public void asociaItem(String nombrePersonaje, String nombreItem);
    public void asociaMagia(String nombrePersonaje, String
                            nombreMagia);
    public void desasociaItem(String nombrePersonaje, String
                              nombreItem);
    public void desasociaMagia(String nombrePersonaje, String
                               nombreMagia);
}
```

Interfaz Home Local. Al igual que con la interfaz remota, las diferencias entre la interfaz Home y la Home local son triviales:
Se debe extender a la clase `javax.ejb.EJBLocalHome`.
Los métodos no lanzan la excepción `java.rmi.RemoteException`.

```
package tonalli.servidor;

import javax.ejb.*;
import java.util.*;
import java.rmi.*;

public interface ActualizaJuegoLocalHome extends javax.ejb.EJBLocalHome
{
    public ActualizaJuego create() throws CreateException;
}
```

Antes de continuar es interesante conocer la diferencia que existen entre los métodos requeridos por los beans de sesión (cuando tiene estado y cuando no) y que deben ser implementados en la clase del bean.[43]

[43] Roman Ed, Mastering EJB; Willey Computer Publishing, 2002.

Método	Descripción	Implementación	
		Beans de sesión con estado.	Beans de sesión sin estado.
setSessionContext(SessionContext ctx)	Asocia al bean con un contexto de session.	Almacena el contexto en una variable de instancia.	
ejbCreate(...)	Inicializa al bean de session.	Se lleva a cabo cualquier inicialización que sea necesaria con base en los argumentos que se pasan. Pueden existir cuantos métodos ejbCreate() sean necesarios.	Se lleva a cabo cualquier inicialización que sea necesaria. Sólo existe un método ejbCreate() sin argumentos.
ejbPassivate()	Llamado antes de que el bean sea pasivado.	Se liberan recursos utilizados actualmente por el bean.	No se utiliza ya que no guarda estado conversacional. Implementación Vacía.
ejbActivate()	Llamado antes de que el bean sea activado.	Se obtienen los recursos que el bean requiera para atender a un cliente.	No se utiliza ya que no guarda estado conversacional. Implementación Vacía.
ejbRemove()	Llamado por el contenedor cuando el bean se elimina de memoria.	Se prepara al bean para su destrucción. Ojo, no se garantiza que se elimine al invocar este método.	

Ciclo de Vida de los Bean de Sesión sin Estado.

Debido a que este tipo de beans no mantienen un estado conversacional, su ciclo de vida es relativamente sencillo y cuenta sólo con dos estados.

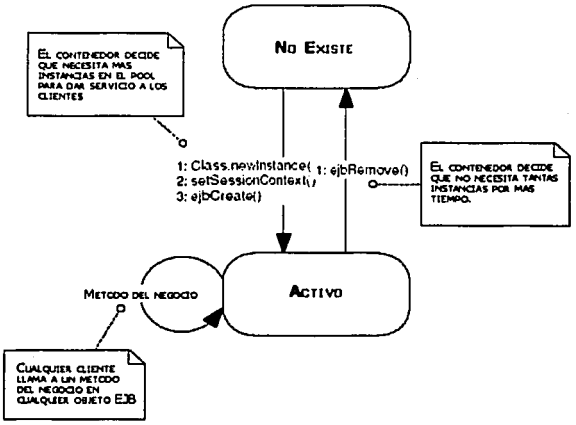


Ilustración 44 : Diagrama del ciclo de vida de sesión de un estado.

No Existe. No existe instancia alguna del bean de sesión en el contenedor. Probablemente el servidor ha sido iniciado.

Activo.[44] Las instancias de estos beans entran a este estado conforme el contenedor determina que son necesarias. Generalmente cuando se inicia el servidor el contenedor instancia un determinado número de estos beans y los coloca en este estado. Las instancias en este estado están listas para atender a las solicitudes de los clientes.

Transición hacia el estado Activo.
Si el bean no existe: Se instancia el bean con la llamada al método `Class.newInstance()`, posteriormente el contenedor invoca al método `SessionBean.setSessionContext()` para obtener una referencia del contexto que va a manejar durante su ciclo de vida. Finalmente se invoca al método `ejbCreate()` del bean. Cabe recordar que en estos beans, sólo existe un método `ejbCreate()` el cual no recibe argumentos, así mismo este método sólo se invoca una vez durante el ciclo de vida del bean y una llamada del cliente al método `create()` en la objeto EJBHome no mapea directamente al método `ejbCreate()`. De igual forma, debido a que los beans de sesión sin estado no están sujetos al mecanismo de activación-pasivación, pueden mantener las referencias a los recursos que utilizan.

Transición fuera del estado Activo.
Las instancias de estos beans pasan del estado Activo al estado No Existe cuando el contenedor ya no las necesita. Así, el proceso se inicia con la invocación al método `ejbRemove()`. Sin embargo, cuando el cliente invoca al método `remove()` en las interfaces Home o Remote, simplemente indica que ya no se requiere el bean, pero este sólo se pasa al estado de No Existe hasta que el contenedor lo determina.

Ciclo de vida de los Bean de Sesión con Estado.

Una de las principales diferencias entre los beans de sesión con estado en comparación con los demás tipos de beans, es que estos no utilizan lo que se conoce como instance pooling, es decir, que puedan estarse intercambiando de un cliente a otro de forma indistinta, ya que como se ha explicado, un bean de sesión con estado está dedicado a un cliente durante todo su ciclo de vida.[45]

[44] El nombre en inglés para este estado es: "Method-Ready Pool".
[45] Algunas implementaciones de contenedores sí permiten el instante pooling de los beans de sesión con estado, pero el resultado siempre es el mismo, ya que para el cliente siempre lo atenderá el mismo bean hasta que finalice su transacción.

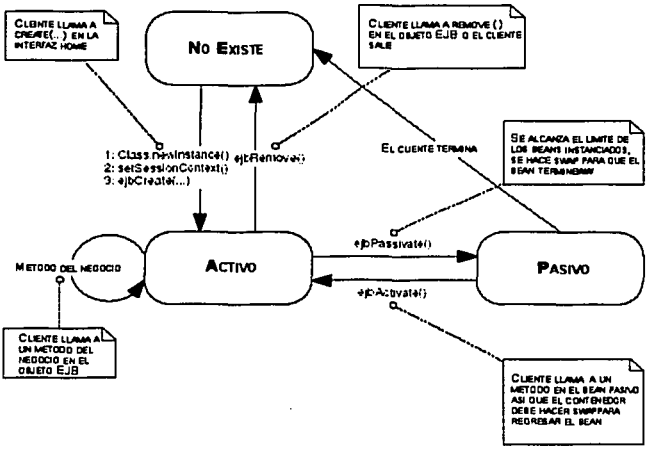


Ilustración 45 : Diagrama del ciclo de vida de un bean de sesión con estado.

- **No Existe.** No existe instancia alguna del bean de sesión en el contenedor. Probablemente el servidor ha sido iniciado.
- **Activo.**[46] Es el estado en el que existe una instancia del bean que puede atender a las peticiones de los clientes. Es diferente al estado Activo de los beans de sesión sin estado, ya que en este sí se mantiene un estado conversacional.
- **Pasivo.** Durante el ciclo de vida de un bean de sesión con estado, existen periodos de inactividad en los que el bean no se encuentra atendiendo solicitudes del cliente, así, para conservar recursos, el contenedor puede pasivar este bean para ahorrar recursos. Cuando se llega a este estado, el contenedor persiste temporalmente el estado del bean.[47] Cuando un bean va a ser pasivado, se manda llamar a su método `ejbPassivate()` para que cierre las referencias a los recursos que está utilizando.

Transición hacia el estado Activo.

- Si el bean no existe: Cuando el cliente invoca al método `create()` en un objeto del tipo `EJBHome`, el contenedor crea una nueva instancia del bean llamando a su metodo `Class.newInstance()`, posteriormente el contenedor invoca al método `SessionBean.setSessionContext()` del bean para que este conozca el contexto en el que va a trabajar durante su ciclo de vida. Finalmente se invoca al método `ejbCreate()` en el bean que

[46] El nombre en inglés para este estado es: "Method-Ready".
[47] Generalmente se utiliza la Serialización de objetos como almacenamiento para la pasivación.

coincida con el método `create()` invocado con el cliente. Una vez invocado este método, se regresa esta referencia al cliente para que pueda realizar sus peticiones y el bean toma este estado.

- Si el bean está pasivado: Cuando un cliente invoca a un método de un bean que está pasivado, el contenedor se encarga entonces de deserializar la instancia del bean pasivado, obteniendo nuevamente su estado conversacional. Una vez que este estado ha sido recuperado, se invoca al método `ejbActivate()` para que se inicialicen todas aquellas variables que no pudieron ser pasivadas. Así, el bean vuelve a entrar al estado de Activo.

Transición fuera del estado Activo.
Las instancias de un bean pueden dejar el estado de Activo hacia el estado de Pasivo o No Existe, dependiendo del uso que se les da por medio del cliente. Generalmente si es por inactividad y no se ha terminado una transacción, la instancia del bean pasa al estado de pasivo, si por el contrario, el contenedor determina que esa instancia debe ser eliminada, entonces pasa al estado de No Existe.

Convenciones.

Como puede verse de estos ejemplos cada clase sigue un código para ser nombrada, de forma que sea más fácil su identificación. Así, durante todo el proyecto se seguirán las siguientes convenciones de nombres:

- Interfaz Remota: Nombre de la clase, por ejemplo `ActualizaJuego`.
- Interfaz Local: Nombre de la clase con el sufijo `Local`, por ejemplo `ActualizaJuegoLocal`.
- Interfaz Home: Nombre de la clase con el sufijo `Home`, por ejemplo `ActualizaJuegoHome`.
- Interfaz Home Local: Nombre de la clase con el sufijo `LocalHome`, por ejemplo `ActualizaJuegoLocalHome`.
- Clase del Bean: Nombre de la clase con el sufijo `Bean`, por ejemplo `ActualizaJuegoBean`.

Descriptores.

Hasta este punto ya se cuenta con el código que me permite acceder de forma tanto remota como local a los beans y se ha codificado el comportamiento de los métodos del negocio, pero todavía quedan algunas cuestiones sin resolver, una de ellas, es la de cómo saber que mi clase del bean realmente implementa los métodos definidos en la interfaz remota ya que nunca se implementa de forma directa en el bean y de la misma forma, como saber si mi bean de sesión es con o sin estado.

Es en este punto donde se toma en consideración el descriptor de despliegue (deployment descriptor). Este descriptor es el encargado de describir las características del bean al contenedor. Estos descriptores son uno de los puntos fuertes de EJB ya que permiten especificar de forma declarativa los atributos del bean en lugar de programarlos directamente dentro del bean. Esto es bastante útil ya que permite modificar estas propiedades sin necesidad de modificar de forma significativa el código.

Físicamente un descriptor de despliegue es un documento XML bien formado, el cual, generalmente es generado automáticamente por las herramientas IDE (Integrated Development Environment o Ambiente Integrado de Desarrollo) como JBuilder, aunque nada impide al desarrollador crear sus propios descriptores o editar los generados por estas herramientas. A continuación se presenta el descriptor de despliegue para el bean que se ha creado.

Este archivo generado debe almacenarse con el nombre `ejb-jar.xml` y debe ser guardado en un directorio llamado `META-INF`.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE ejb-jar PUBLIC "-//Sun Microsystems, Inc.//DTD Enterprise
JavaBeans 2.0//EN" "http://java.sun.com/dtd/ejb-jar_2_0.dtd">
<ejb-jar>
    <enterprise-beans>
        <session>
            <display-name>ActualizaJuego</display-name>
            <ejb-name>ActualizaJuego</ejb-name>
            <Home>tonalli.servidor.ActualizaJuegoHome</Home>
            <remote>tonalli.servidor.ActualizaJuego</remote>
            <ejb-class>tonalli.servidor.ActualizaJuegoBean</ejb-class>
            <session-type>Stateless</session-type>
            <transaction-type>Container</transaction-type>
        </session>
    </enterprise-beans>
    <assembly-descriptor>
        <container-transaction>
            <method>
                <ejb-name>ActualizaJuego</ejb-name>
                <method-name>*</method-name>
            </method>
            <trans-attribute>Required</trans-attribute>
        </container-transaction>
    </assembly-descriptor>
</ejb-jar>
```

Veamos ahora parte por parte cada una de las partes que componen a este descriptor.

Encabezado.

Cada descriptor debe comenzar indicando la versión de XML que está utilizando:

```
<?xml version="1.0" encoding="UTF-8"?>
```

En este caso se indica que el documento se adhiere a la versión 1.0 y la codificación de caracteres utilizada es la UTF-8 que es la soportada por los contenedores EJB disponibles.

La siguiente etiqueta especifica el DTD que define al documento:

```
<!DOCTYPE ejb-jar PUBLIC "-//Sun Microsystems, Inc.//DTD
Enterprise JavaBeans 2.0//EN" "http://java.sun.com/dtd/ejb-
jar_2_0.dtd">
```

Esta etiqueta ofrece el URL desde el cual se puede descargar el DTD para la validación de este documento.

Cuerpo.

El cuerpo del descriptor comienza y termina con la etiqueta que determina la raiz del documento, en este caso la raíz corresponde a la etiqueta `<ejb-jar>`, por lo que el contenido del cuerpo debe estar contenido entre las siguientes etiquetas:

```
<ejb-jar>
    demas elementos
</ejb-jar>
```

Dentro de dichas etiquetas se coloca toda la información relativa a los beans, en este caso tenemos:

`<enterprise-beans>` Es una etiqueta requerida y contiene la descripción de todos los beans que se incluyen dentro de un archivo JAR[48]. Dentro de esta etiqueta se indica para cada bean su tipo con las etiquetas `<session>`, `<entity>` y `<message-driven>`. En este ejemplo solo contamos con un solo bean de sesión, por lo que se tiene lo siguiente:

```
<enterprise-beans>
    <session>
        atributos del bean...
    </session>
</enterprise-beans>
```

Dentro de la etiqueta `<session>...</session>`, `<entity>...</entity>` y `<message-driven>...</message-driven>` se incluye toda la información que describe propiamente a cada bean. En este ejemplo sólo se tomarán en cuenta aquellos atributos aplicables a todos los beans y particularmente a los beans de sesión, en el apartado siguiente se profundizarán en todos aquellos atributos relativos a los beans de entidad.

- `<ejb-name>...</ejb-name>`. Requerido por todos los beans. Especifica el nombre del componente EJB. El nombre utilizado en esta etiqueta para cada uno de los ejemplos es igual al nombre de la interfaz remota.
- `<Home>...</Home>`. Requerido por todos los beans. Especifica el nombre completamente calificado de la clase de la interfaz Home del bean.
- `<remote>...</remote>`. Requerido por todos los beans. Especifica el nombre completamente calificado de la clase de la interfaz remota del bean.
- `<local-Home>...</local-Home>`. Requerido por todos los beans que implementan interfaz Home local. Especifica el nombre completamente calificado de la clase de la interfaz Home local del bean.

[48] Nada impide que dentro de un solo descriptor se describan las propiedades de varios beans, siempre y cuando se incluya dicha descripción dentro de las etiquetas `<enterprise-beans>...</enterprise-beans>`

- `<local>...</local>`. Requerido por todos los beans que implementan interfaz local. Especifica el nombre completamente calificado de la clase de la interfaz local del bean.
- `<ejb-class>...</ejb-class>`. Requerido por todos los beans. Especifica el nombre completamente calificado de la clase del bean.
- `<session-type>...</session-type>`. Requerido solo por beans de sesión. Se indica si un bean de sesión es con estado (`Stateful`) o sin estado (`Stateless`).
- `<transaction-type>...</transaction-type>`. Requerido solo por los beans de sesión. Declara si el bean es el encargado de manejar sus propias transacciones (`Bean`) o si es el contenedor quien se encarga de dicho manejo (`Container`).

Hasta este punto hemos descrito todo lo que podemos acerca de nuestro bean propiamente, por lo que todo hemos concluido en describir todo aquello que se coloca dentro de las etiquetas `<enterprise-beans>...</enterprise-beans>`. Ahora debemos especificar la forma en la que los beans descritos se ensamblan dentro de una aplicación, para esto contamos con la etiqueta `<assembly-descriptor>...</assembly-descriptor>`, que dentro de nuestro documento XML se coloca al mismo nivel que `<enterprise-beans>`:

```
<ejb-jar>
    <enterprise-beans>
        descripcion de los beans
    </enterprise-beans>
    <assembly-descriptor>
        descripcion de ensamble de beans en la aplicación
    </assembly-descriptor>
</ejb-jar>
```

Cabe hacer notar que esta etiqueta es de uso opcional **sólo** por aquellos desarrolladores cuya función es la de generar el código de los beans, pero no para quien se encarga de realizar el despliegue de los componentes desarrollados. Ahora bien, esta etiqueta cumple con tres funciones básicas:

1. Describe los atributos transaccionales que se aplican a los métodos del bean.
2. Describe la lógica de seguridad.
3. Especifica los permisos de los métodos.

Las etiquetas que puede contener son:

`<container-transaction>...</container-transaction>`. Este atributo declara que atributos de transaccionales se aplican a que métodos. A su vez contiene uno o más elementos `<method>...</method>` y un elemento `<trans-attribute>...</trans-attribute>`.

```
<assembly-descriptor>
    <container-transaction>
        <method>
            <ejb-name>ActualizaJuego</ejb-name>
            <method-name>*</method-name>
```

```
            </method>
            <trans-attribute>Required</trans-attribute>
        </container-transaction>
    </assembly-descriptor>
```

En este caso dentro de la etiqueta `<method>...</method>` se indica en primera instancia el nombre del bean cuyos métodos se están describiendo, a continuación se desglosan los nombres de los métodos afectados; en este caso se utiliza el comodín * que hace referencia a todos los métodos del bean.

La etiqueta `<trans-attribute>...</trans-attribute>` nos indica con el valor `Requiered` que todos los métodos **requieren** que exista una transacción para ser ejecutados.

EJB-JAR[49].

Un archivo JAR (acrónimo de Java ARchive) no es más que un formato de archivo independiente de la plataforma utilizado para la compresión, empaquetamiento y despliegue de varios archivos dentro de uno sólo. Este formato esta basado en el formato .ZIP y los estándares de compresión ZLIB. El objetivo de utilizar este tipo de archivos para desplegar EJB's es el de empaquetar todas las clases e interfaces asociadas con un bean, incluyendo al descriptor de despliegue.

Este archivo JAR puede ser creado utilizando las herramientas del IDE o con la utilería jar que es parte de cualquier versión del JSDK. Un archivo JAR contiene:

El descriptor de despliegue.
Las clases del bean.
Las interfaces remotas y Home (y sus contrapartes locales).
Las clases llave primaria.[50]
Clases e interfaces utilizadas.

Dentro de este archivo JAR, el descriptor generado debe colocarse dentro del directorio META-INF y debe contener toda la información de los beans que se encuentran presentes en el JAR, ya que para cada bean declarado en el descriptor, el archivo JAR debe contener sus interfaces y la clase propia del bean.

Para crear un archivo JAR desde la línea de comando se utiliza la siguiente instrucción:

```
jar cf <nombre_jar.jar> <ruta_a_archivos>/*.class META-INF/ejb-jar.xml
```

Esta instrucción crea un archivo JAR con el nombre `nombre_jar.jar` al que se redirige el resultado del comando (opción `f`) al JAR especificado. La opción `c` indica que se desea comprimir los archivos que se procesen.

[49] Este apartado se aplica de igual forma a todos los beans (Sesión, Entidad, Mensajes).
[50] Propias de los beans de entidad y se explican en la sección siguiente.

Cabe hacer notar que antes de ejecutar esta instrucción se debió haber creado un directorio llamado `META-INF` y en ese se debió haber colocado el descriptor. Así mismo, la parte de `<ruta_a_archivos>/*.class` debe hacerse por cada paquete que se maneje.

Siguiendo el ejemplo anterior, la estructura del archivo ActualizaJuego.jar sería la siguiente:

```
META-INF/MANIFEST.MF
META-INF/ejb-jar.xml
tonalli/servidor/ActualizaJuegoBean.class
tonalli/servidor/ActualizaJuegoLocalHome.class
tonalli/servidor/ActualizaJuegoLocal.class
tonalli/servidor/ActualizaJuegoHome.class
tonalli/servidor/ActualizaJuego.class
```

Se puede observar que dentro del directorio `META-INF` existe un archivo llamado `MANIFEST.MF`, este archivo es generado de forma automática por la herramienta y contiene la información de todos los archivos que se encuentran dentro del JAR, por lo que no debe eliminarse bajo ningún motivo.

Beans de sesión Desarrollados.

Para hacer posible el desarrollo del juego, es necesario poner a disposición de los jugadores una serie de servicios que consulten, validen o actualicen el estado el juego. Esto se logra a través de beans de sesión, mismos que residen en el servidor, conformando la parte dinámica del negocio.

Los servicios del negocio están agrupados en cuatro beans de acuerdo a los factores que afectan o consultan.

Tabla: Listado de Beans de Sesion.

Nombre	Responsabilidad	Paquete	Discusión
ActualizaBD	Administra los registros del personaje en su estado actual.	*Tonalli.servidor.*	*Administra los registros de la base de datos en las tablas EstadoPersonaje, ConocimientoMágico e Inventario.*
ActualizaJuego	Administra los elementos del Juego para que los registros de la base de datos estén acordes con las actividades realizadas.	*Tonalli.servidor.*	*Actualiza o selecciona Personajes, ítems, magias o efectos.*
AdministraJuego	Hace consultas a los elementos que necesita para su posterior ubicación en el Juego.	*Tonalli.servidor.*	*Realiza las consultas que mantendrán el juego en condiciones idóneas al inicio del juego. Obtiene las condiciones iniciales para empezar a jugar.*
Conexion	Establece una conexión para conocer el usuario que ingresa.	*Tonalli.servidor.*	*Valida si el usuario existe en la base de datos.*

Bean de sesión: Conexión.

Encargado de ofrecer todos aquellos servicios externos al juego y que permiten el acceso y registro de jugadores y la creación de personajes. El cliente crea una instancia de este bean para realizar las funciones de autentificación de usuarios y de creación de cuentas y posteriormente es desechada.

Tabla: Listado de clases del Bean Conexion.

Nombre	*Responsabilidad*	*Paquete*	*Discusión*
Conexión	Interfaz remota del bean.	*Tonalli.servidor.*	*Declara los métodos de validación de la existencia de un usuario en la base de datos, implementados en la clase del bean.*
ConexiónHome	Interfaz Home del bean	*Tonalli.servidor*	*Declara el método de creación del bean de sesión para su implementación en la clase del bean.*
ConexiónBean	Clase del bean	*Tonalli.servidor*	*Implementación de los métodos de validación de la existencia de un usuario en la base de datos.*

TablaServicios de la Interfaz Home de Conexión

Nombre	*Parametros*	*Tipo Salida*	*Excepciones*	*Objetivo*
create	void	Conexión	CreateException, RemoteException	Crear un bean de sesión mediante el bean.

Tabla:Servicios de la Interfaz Remota de Conexión

Nombre	*Parametros*	*Tipo Salida*	*Excepciones*	*Objetivo*
validaUsuario	String extLogin, String extPassword	Boolean	NamingException, FinderException, RemoteException	Valida que los datos de la cuenta(login y password) sean los mismo que estan registrados en la Base de Datos.
existeCliente	String extLogin	Boolean	NamingException, FinderException, RemoteException	Valida que los datos de la cuenta(login y password) sean los mismo que estan registrados en la Base de Datos.

Bean de sesión: AdministraJuego.

Encargado de ofrecer los servicios de consulta de cualquier información de la imagen del juego, por lo que tiene una relación estrecha con la clase Juego. Representa el rol de "experto" en los patrones de la orientación a objetos, que es la clase que conoce la información y es capaz de proporcionarla a quien la solicite.. Puede definirse como un administrador de la información del juego, cada instancia de este bean activa corresponde a un cliente que la utilizara para obtener información.. La primera instancia de este bean se encarga de la creación del juego, por lo que tiene capacidad de acceso a la capa de acceso a datos, de quien recibe la información para dar condiciones iniciales y estructura al nuevo juego. Instancias posteriores toman el juego existente y solo agregan al nuevo personaje a la partida.

Tabla: Listado de clases del Bean AdministraJuego

Nombre	*Responsabilidad*	*Paquete*	*Discusión*
AdministraJuego	Interfaz remota del bean.	*Tonalli.servidor.*	*Declara los métodos que serán implementados en la clase del bean. Métodos de búsqueda de personaje, ítem, magia, tablero, partida, casillas y posición del personaje.*
AdministraJuegoHome	Interfaz Home del bean	*Tonalli.servidor*	*Declara el método de creación del bean de sesión para su implementación en la clase del bean.*
AdministraJuegoBean	Clase del bean	*Tonalli.servidor*	*Implementación de los métodos de búsqueda de personaje, ítem, magia, tablero, partida, casillas y posición del personaje.*

Tabla:Servicios de la Interfaz Home de AdministraJuego

Nombre	*Parámetros*	*Tipo Salida*	*Excepciones*	*Objetivo*
create		AdministraJuego	CreateException	Crea el bean de sesión, accediendo un Bean de entidad
create	String nombrePersonaje	AdministraJuego	CreateException	Crea el bean de sesión, accediendo un Bean de entidad con el nombre de un personaje que identificará-

Tabla:Servicios de la Interfaz Remota de AdministraJuego

Nombre	*Parámetros*	*Tipo Salida*	*Excepciones*	*Objetivo*
buscaPersonaje	String nombre	Casilla	RemoteException	Localiza a un personaje en el juego.
buscaItem	String nombre	Casilla	RemoteException	Localiza un ítem en el juego
obtenInventario		Inventario	RemoteException	Obtiene las referencias del inventario del personaje
estableceePosicionItems	Collection items	Void	RemoteException	Asigna coordenadas a los ítems para su inclusión en el juego.
obtenConocimientoMagico		ConocimientoMagico	RemoteException	Obtiene las referencias del conocimiento mágico del personaje
obtenTablero		Tablero	RemoteException	Obtiene el tablero
obtenPartida		Partida	RemoteException	Obtiene la partida de personajes que participarán en el juego.
obtenPersonaje	String nombre	Personaje	RemoteException	Obtiene las referencias de los personajes participantes.
obtenItem	String nombre	Ítem	RemoteException	Obtiene la referencia de los ítems existentes
obtenMagia	String nombre	Magia	RemoteException	Obtiene la referencia de las magias existentes.
obtenCasilla	int coordX, int coordY	Casilla	RemoteException	Obtiene el ambiente de las casillas para su localización en el tablero.
obtenNumeroCasillas		Int	RemoteException	Obtiene el número de casillas que se incluirán en el juego.
obtenZonaJuego	String colorZona	Collection	RemoteException	Obtiene la Zona en la que se puede aplicar el juego, es decir, las zonas activas.
posicionaPersonaje	String nombrePersonaje, Collection zonaCasillas	Bolean	RemoteException	Ubica a un personaje dentro del ambiente de una casilla.

Bean de sesión: ActualizaJuego.

Encargado de ofrecer los servicios relacionados con modificaciones al estado del juego. Este bean modifica la imagen del juego a solicitud de los clientes y del propio servidor mediante la realización de acciones. Tiene relación estrecha con la clase Juego pero con fines de actualización. Puede definirse como un administrador de modificaciones del juego y cada instancia activa del este bean esta relacionada con un cliente quien la utiliza para afectar el juego.

Tabla: Listado de clases del Bean ActualizaJuego.

Nombre	*Responsabilidad*	*Paquete*	*Discusión*
ActualizaJuego	Interfaz remota del bean.	*Tonalli.servidor.*	*Declara los métodos de negocio que actualizarán la posición del personaje, el estado de un ítem y magia, así como la aplicación de un efecto a un objetivo; para su implementación en el bean.*
ActualizaJuegoHome	Interfaz Home del bean	*Tonalli.servidor*	*Declara el método de creación del bean de sesión para su implementación en la clase del bean.*
ActualizaJuegoBean	Clase del bean	*Tonalli.servidor*	*Implementación de los métodos de negocio que actualizarán la posición del personaje, el estado de un ítem y magia, así como la aplicación de un efecto a un objetivo*

Tabla: Servicios de la Interfaz Home de ActualizaJuego

Nombre	*Parametros*	*Tipo Salida*	*Excepciones*	*Objetivo*
create		ActualizaJuego	CreateException, RemoteException	Crear el bean de sesión accediendo un Bean de entidad

Tabla: Servicios de la Interfaz Remota de ActualizaJuego

Nombre	*Parametros*	*Tipo Salida*	*Excepciones*	*Objetivo*
colocaPersonaje	String nombre, int coordX, int coordY	Void	RemoteException	Situar al personaje en una coordenada dentro del tablero.
colocaItem		Void	RemoteException	Colocar un personaje en una coordenada dentro del tablero
sacaPersonaje	String nombre	Void	RemoteException	Quitar al personaje del tablero.
sacaItem	String nombre	Void	RemoteException	Quitar la asociación de un ítem del inventario del personaje
muevePersonaje	String nombre, int destinoX, int destinoY	Void	RemoteException	Mover a un personaje de posición dentro del tablero.
aplicaEfecto	Collection personajes, int[] efecto	Void	RemoteException	Actualizar el registro de efecto de personaje.
eliminaPersonaje	String nombre	Void	RemoteException	Remover al personaje.
eliminaItem	String nombre	Void	RemoteException	Quitar la asociación de un ítem en el inventario.
asociaItem	String nombrePersonaje, String nombreItem	Void	RemoteException	Asociar un ítem con un inventario de personaje.
asociaMagia	String nombrePersonaje, String nombreMagia	Void	RemoteException	Asociar una magia con un inventario de personaje.
desasociaItem	String nombrePersonaje, String nombreItem	Void	RemoteException	Remover la asociación de un ítem sobre el inventario de personaje.
desasociaMagia()	String nombrePersonaje, String nombreMagia	Void	RemoteException	Remover la asociación de una magia sobre el Conocimiento mágico de personaje.

Bean de sesión: ActualizaBaseDeDatos.

Encargado de ofrecer los servicios de persistencia de información en la base de datos. De forma alterna al bean ActualizaJuego, las acciones que requieren persistencia llaman a este bean, para hacer permanentes las modificaciones que son reflejadas también en la imagen del juego. Posee la capacidad de utilizar los métodos proporcionados por la capa de acceso a datos para realizar susfunciones.

Tabla: Listado de clases del Bean ActualizaBD

Nombre	*Responsabilidad*	*Paquete*	*Discusión*
ActualizaBD	Interfaz remota del bean.	Tonalli.servidor	Declara los métodos de negocio que actualizarán el estado del personaje, el inventario y conocimiento mágico implementados en la clase del bean.
ActualizaBDHome	Interfaz Home del bean	Tonalli.servidor	Declara el método de creación del bean de sesión para su implementación en la clase del bean.
ActualizaBDBean	Clase del bean	Tonalli.servidor	Implementación de los métodos. Hace la búsqueda del entity bean de EntPersonaje para aplicar los métodos del negocio

Tabla: Servicios de la Interfaz Home de ActualizaBD

Nombre	*Parametros*	*Tipo Salida*	*Excepciones*	*Objetivo*
Create		ActualizaBD	RemoteException, CreateException	Crear un bean de sesión accediendo un Bean de entidad.

Tabla:Servicios de la Interfaz Remota de ActualizaBD

Nombre	*Parametros*	*Tipo Salida*	*Excepciones*	*Objetivo*
actualizaEstadoPersonaje	Personaje personaje	Void	RemoteException	Actualizar el registro de estado del personaje
validaEstadoPersonaje	Personaje personaje	Personaje	RemoteException	Validar el estado del personaje activo
agregaItemInventario	Item item, Personaje personaje	Void	RemoteException	Agregar una asociación de un ítem al inventario de un personaje
agregaMagiaConocimiento	Magia magia, Personaje personaje	Void	RemoteException	Agregar una asociación al Conocimiento Mágico del personaje
quitaItemInventario	Item item, Personaje personaje	Void	RemoteException	Quitar una asociación del inventario del personaje
quitaMagiaConocimiento	Magia magia, Personaje personaje	Void	RemoteException	Quitar una asociación del Conocimiento Mágico del personaje

Descripción de la capa de acceso a datos.

La capa de acceso a datos proporciona los servicios relacionados con en manejo de información de la base de datos, como son ingreso, modificación y consulta de registros. El esquema EJB define esta capa para controlar la persistencia de la información de forma independiente de la estructura que proporcione la herramienta que almacena realmente los datos. Así, la forma en la que están organizadas y relacionadas las tablas no afectan el manejo de la información, y puede alterarse siempre que se mantenga la relación de las tablas con los beans de entidad responsables de su administración. Por medio de este nivel de trabajo se prepara al sistema para que conozca la forma de gestionar los datos requeridos, y observar funciones de seguridad y transacciones.

Las responsabilidades de la capa de acceso a datos son:

- Almacenamiento de datos.
- Recuperación y consulta de datos.
- Mantenimiento y actualizaciones de datos.

Observación de la integridad de la infromación.

Ésta es la capa que interactúa de manera directa con los servicios de negocio, ya que recibe las peticiones de datos (cuando así se requiere), y se nutre de características propias de un RDBMS (Servidor de administración de bases de datos relacionales) como es Microsoft SQL Server. En ella residen todos los datos que son solicitados por la capa de negocio, y la de presentación.
Cada tabla de la base de datos, significativa para el juego, tiene imagen en un bean de entidad dentro de este nivel de trabajo, siendo posible y muy útil el mapeo de varias tablas a un solo bean, como se verá más adelante en este capítulo, rompiendo con algunas de las restricciones de las bases de datos distribuidas acercando al desarrollador a la imagen de la información que le es más conveniente.

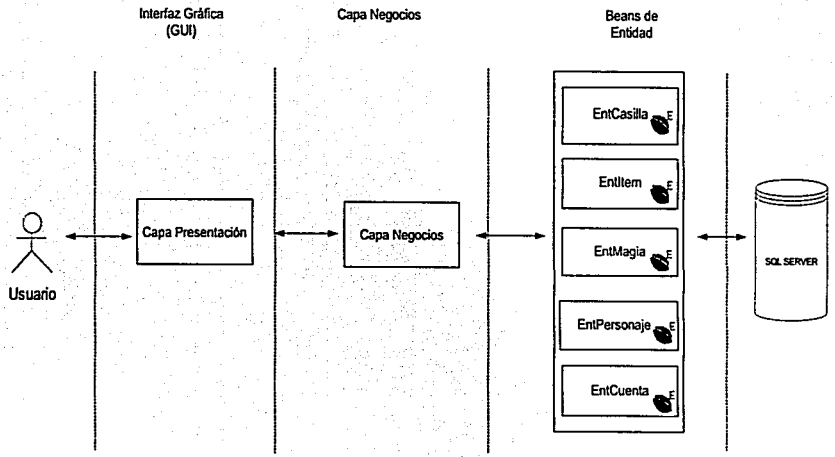


Ilustración 46 : Interacción de la capa de datos.

Los principales componentes de este nivel de trabajo son, por un lado la base de de datos y por otro los beans de entidad que le dan soporte dentro de la arquitectura J2EE.

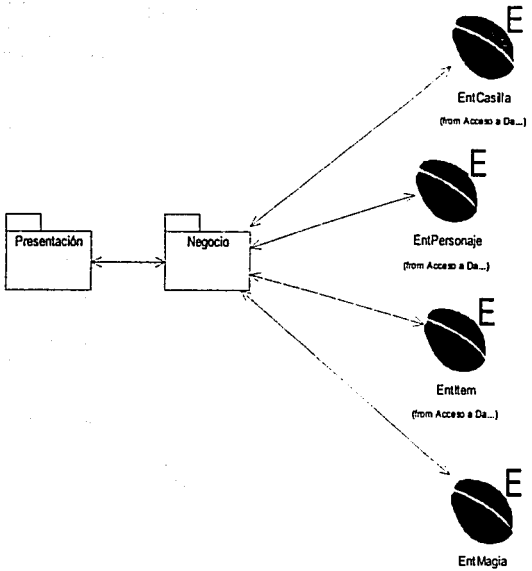


Ilustración 47 : Descripción de la capa de datos.

Estructura de la base de datos.

Definimos a las bases de datos como la serie de datos organizados y relacionados que son recolectados y explotados por los sistemas de información para cumplir con diversos objetivos del negocio.
Como se mencionó anteriormente, los datos almacenados en la base de datos sirven de soporte para la creación del juego así como la definición de los elementos válidos en el universo creado para el juego. En este proyecto, se utiliza para el almacenamiento y estructuración de la información un servidor de bases de datos SQL Server 7.0.
Del análisis de la información manejada por el sistema del videojuego, negocio del presente caso práctico, se derivó el diseño de la base de datos, mediante el esquema relacional. Se presenta a continuación el diagrama de la base de datos:

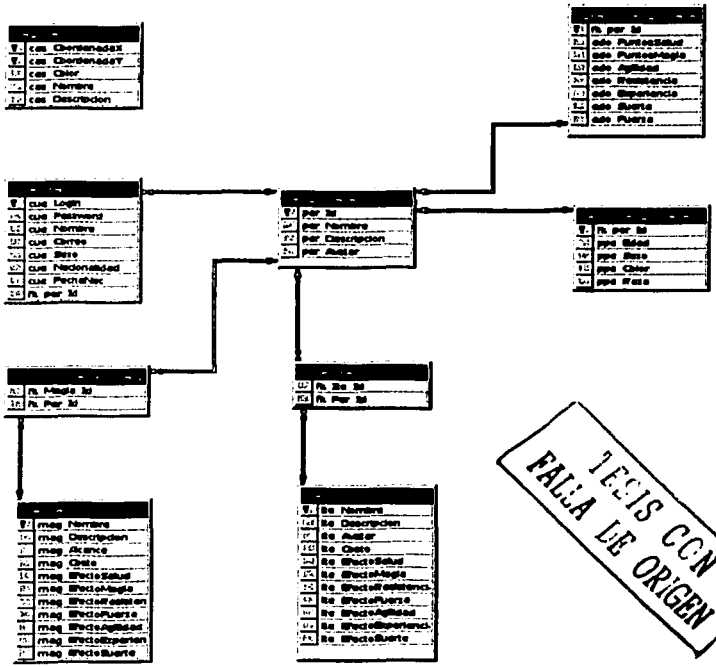


Ilustración 48 : Esquema de la base de datos.

Tabla: Cuenta.

Contiene la información necesaria para definir a los clientes que han sido registrados al sistema. Incluye información descriptiva del usuario, útil para fines estadísticos y de contacto, además de la información necesaria para validar su conexión al sistema, requerida por el método de autentificación de usuarios para acceso, adoptado por el sistema. Esta relacionada con el personaje del cliente, para establecer la imagen del usuario en el juego.

Tablas relacionadas: Personaje.

Atributo	Nulo/ No Nulo	Tipo de Llave	Tipo de Dato	Descripción
Fk_Per_Id	NN	PK/FK	Int	Entero identificador de la cuenta. Llave foránea que viene de la tabla personaje.
Cue_Login	NN		Varchar	Nombre de usuario dentro del sistema. Identificador del usuario en la aplicación
Cue_Password	NN		Varchar	Clave de acceso de la cuenta. Identificador del usuario en la aplicación
Cue_Nombre	NN		Varchar	Nombre del usuario
Cue_Correo	N		Varchar	Correo electrónico del usuario
Cue_Sexo	NN		Char	Sexo del usuario
Cue_Nacionalidad	NN		Varchar	Nacionalidad del usuario
Cue_FechaNac	N		Date	Fecha de nacimiento del usuario

Tabla: Personaje.

Cada cliente tiene, además de una cuenta que incluye su información personal y de validación, un personaje asociado. Este personaje será su imagen dentro del juego, es decir, la representación de ese cliente en particular como elemento de juego. Esta tabla contiene solo la información básica de cada personaje. La descripción completa del personaje se realiza mediante relaciones con las tablas PerfilPersonaje, EstadoPersonaje, Inventario y Conocimiento mágico, que detallan aspectos específicos y variables de la imagen del jugador en el juego.

Tablas relacionadas: Cuenta, PerfilPersonaje, EstadoPersonaje, Inventario, Conocimiento mágico.

Atributo	Nulo/ No Nulo	Tipo de Llave	Tipo de Dato	Descripción
Per_Id	NN	PK	Int	Entero identificador del personaje.
Per_Nombre	NN		Varchar	Nombre del personaje dado por el usuario.
Per_Descripción	NN		Varchar	Descripción del personaje en texto del personaje definida por la raza.
Per_Avatar	NN		Varchar	Nombre de la imagen utilizada para su representación en el juego.

Tabla: PerfilPersonaje.

Es parte de la definición de un Personaje.

Define información del personaje como un actor dentro del juego. Esta información es poco variable y se utiliza para personalizar la imagen del jugador en el juego, y es en su mayoría, referente a su descripción física, misma que nos permite definir estados iniciales para los valores del estado del personaje.

Tablas relacionadas: Personaje.

Atributo	Nulo/ No Nulo	Tipo de Llave	Tipo de Dato	Descripción
Fk_per_Id	NN	PK/FK	Int	Entero identificador del personaje. Llave foránea que viene de la tabla personaje
Ppe_Edad	NN		Int	Edad del personaje
Ppe_Sexo	NN		Char	Sexo del personaje
Ppe_Color	NN		Varchar	Color asignado al personaje según la raza
Ppe_Raza	NN		Varchar	Raza del personaje

Tabla: EstadoPersonaje.

Es parte de la definición de un Personaje.

Define información sobre aspectos cambiantes a lo largo del juego de un personaje. Puede verse como la descripción del avance o cambios que tienen los personajes derivados de la acción de jugar. Una gran cantidad de las acciones del juego, están destinadas a modificar el estado de los personajes y esta tabla refleja esos cambios.

Tablas relacionadas: Personaje.

Atributo	Nulo/ No Nulo	Tipo de Llave	Tipo de Dato	Descripción
Fk_per_Id	NN	PK/FK	Int	Entero identificador del personaje. Llave foránea que viene de la tabla Personaje
Edo_PuntosSalud	NN		Int	Puntos de vida del personaje para el juego.
Edo_PuntosMagia	NN		Int	Puntos de magia del personaje para el juego
Edo_Agilidad	NN		Int	Puntos de facilidad de movimiento, aplicables al movimiento del personaje y a la aplicación de magias
Edo_Resistencia	NN		Int	Puntos de defensa del personaje, aplicados en las batallas del personaje
Edo_Experiencia	NN		Int	Puntos que definen el nivel de experiencia adquirido por el personaje
Edo_Suerte	NN		Int	Puntos de suerte del personaje, aplicados en las batallas del personaje
Edo_Fuerza	NN		Int	Puntos de ataque del personaje, aplicados en las batallas del personaje

Tabla: Inventario.

Parte de la definición del personaje.

Define el conjunto de objetos que el personaje ha obtenido a lo largo de los juegos en que ha participado y que actualmente posee. Se trata de una tabla transitiva que relaciona uno o varios ítems con uno o varios personajes.

Tablas relacionadas: Personaje, Item.

Atributo	Nulo/ No Nulo	Tipo de Llave	Tipo de Dato	Descripción
Fk_Ite_Id	NN	PK/FK	Varchar	Llave foránea de la tabla Item
Fk_Per_Id	NN	PKFK	Int	Llave foránea de la tabla personaje

Tabla: ConocimientoMagico.

Parte de la definición del personaje.

Define el conjunto de magias que el personaje ha aprendido a lo largo de los juegos en los que ha participado y que le es posible utilizar. Se trata de una tabla transitiva que relaciona a uno o varios personajes con una o varias mágias.

Tablas relacionadas: Personaje, Magia.

Atributo	Nulo/ No Nulo	Tipo de Llave	Tipo de Dato	Descripción
Fk_Magia_Id	NN	PK/FK	Varchar	Llave foránea que viene de la tabla Magia
Fk_Per_Id	NN	PK/FK	Int	Llave foránea que viene de la tabla Personaje

Tabla: Item.

Describe a los objetos que existen dentro del juego y con los que se puede tener algún tipo de relación al momento de jugar. Se define como el catálogo de ítems del juego. Los atributos de efecto definen el uso que se puede dar al item en relación con la forma en la que afectan el estado de los personajes.

Tablas relacionadas: Inventario.

Atributo	Nulo/ No Nulo	Tipo de Llave	Tipo de Dato	Descripción
Ite_Nombre	NN	PK/FK	Varchar	Identificador del Ítem
Ite_Descripción	NN		Varchar	Descripción del ítem
Ite_Avatar	NN		Varchar	nombre de la imagen del ítem que se despliega en la aplicación
Ite_Costo	NN		Int	Costo para utilizar el ítem
Ite_EfectoSalud	NN		Int	Puntos aplicados al utilizar el ítem en un objetivo
Ite_EfectoMagia	NN		Int	Puntos aplicados al utilizar el ítem en un objetivo
Ite_EfectoResistencia	NN		Int	Puntos aplicados al utilizar el ítem en un objetivo
Ite_EfectoFuerza	NN		Int	Puntos aplicados al utilizar el ítem en un objetivo
Ite_EfectoAgilidad	NN		Int	Puntos aplicados al utilizar el ítem en un objetivo
Ite_EfectoExperiencia	NN		Int	Puntos aplicados al utilizar el ítem en un objetivo
Ite_EfectoSuerte	NN		Int	Puntos aplicados al utilizar el ítem en un objetivo

Tabla: Magia.

Describe a las magias que existen dentro del juego y con que pueden ser aprendidas y utilizadas al momento de jugar. Se define como el catálogo de magias del juego. Los atributos de efecto definen resultado que se da al aplicar la magia en relación con la forma en la que afecta el estado de los personajes.

Tablas relacionadas: ConocimientoMagico.

Atributo	Nulo/ No Nulo	Tipo de Llave	Tipo de Dato	Descripción
Mag_Nombre	NN	PK	Varchar	Identificador de la magia
Mag_Descripción	NN		Varchar	Descripción de la magia
Mag_Alcance	NN		Int	Alcance en el tablero de la magia, en

Atributo	Nulo/ No Nulo	Tipo de Llave	Tipo de Dato	Descripción
				términos de casillas.
Mag_Costo	NN		Int	Costo del uso de una magia
Mag_EfectoSalud	NN		Int	Puntos aplicados al utilizar la magia en un objetivo
Mag_EfectoMagia	NN		Int	Puntos aplicados al utilizar la magia en un objetivo
Mag_EfectoResistencia	NN		Int	Puntos aplicados al utilizar la magia en un objetivo
Mag_EfectoFuerza	NN		Int	Puntos aplicados al utilizar la magia en un objetivo
Mag_EfectoAgilidad	NN		Int	Puntos aplicados al utilizar la magia en un objetivo
Mag_EfectoExperiencia	NN		Int	Puntos aplicados al utilizar la magia en un objetivo
Mag_EfectoSuerte	NN		Int	Puntos aplicados al utilizar la magia en un objetivo

Tabla: Casilla.

Describe al conjunto de secciones del tablero. Cada casilla es una coordenada dentro del tablero del juego. Se define como el catálogo de casillas del tablero del juego.

Atributo	Nulo/ No Nulo	Tipo de Llave	Tipo de Dato	Descripción
cas_CoordenadaX	NN	PK	Int	Coordenada X de la casilla
cas_CoordenadaY	NN	PK	Int	Coordenada Y de la casilla
Cas_Color	NN		Char	Color de la casilla para el tablero
Cas_Nombre	NN		Char	Característica especial de la casilla que atribuye importancia en el juego
Cas_Descripcion	NN		Varchar	Descripción de la casilla

Mapeo de las tablas a beans de entidad.

Una vez realizado el diseño de la base de datos, es necesario determinar la forma en que la información será gestionada por los beans de entidad. El esquema J2EE nos permite organizar la información en la forma que el desarrollo requiera por lo que no se presenta la restricción en el número de tablas que un bean de entidad representa, aunque lo más común es encontrar ejemplos de mapeo de una tabla a un bean de entidad.

Para determinar esto, se identifican los conceptos principales para el sistema y se observa la forma en que la información que los define esta organizada en las tablas de la base de datos. Todas las tablas que se refieran a una sola entidad del sistema son representadas por un solo bean de entidad denominado con el nombre de la entidad.

Lo anterior nos permite agrupar las tablas en conceptos identificables, dejando a la capa de acceso a datos el trabajo pesado con la base de datos, haciendolo transparente hacia fuera de este nivel de trabajo.

Tabla	Bean	Clase Objeto Valor
Cuenta	EntCuenta	-
Personaje	EntPersonaje	Personaje
PerfilPersonaje		PerfilPersonaje
EstadoPersonaje		EstadoPersonaje
Inventario		Inventario
ConocimientoMágico		ConocimientoMagico
Item	EntItem	Item
Magia	EntMagia	Magia
Casilla	EntCasilla	Casilla

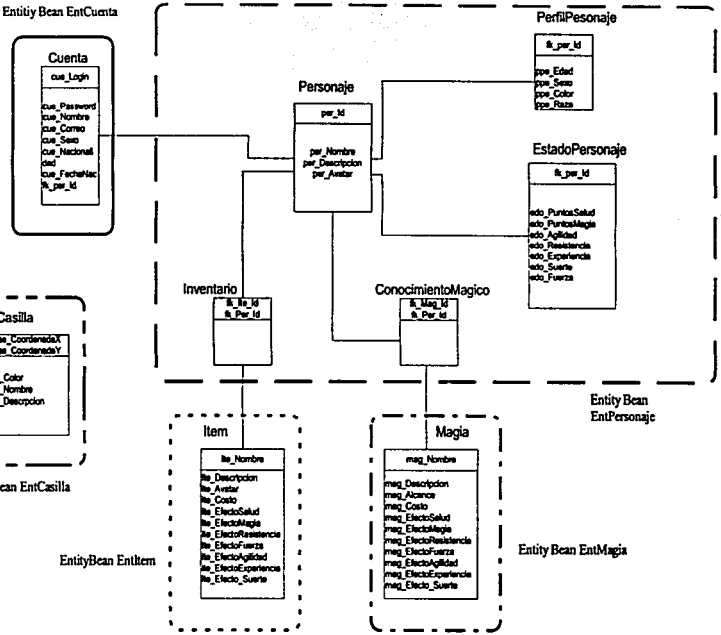


Ilustración 49 : Mapeo de las tablas a beans de entidad.

Objetos valor: tipos de datos.

Los beans de entidad entregan a quien lo solicite (capa de negocio) una imagen de un registro o de un conjunto de los mismo, que posteriormente son convertidos a clases manejables por cualquier componente del sistema. Estas clases se denominan Objetos Valor (Value Object). Usalmente un objeto valor es un pequeño objeto con uso extensivo dentro de un sistema, por lo que puede verse como un tipo de datos requerido por el sistema, para optimizar los procesos a la vez que se hace uso de las características de la orientación a objetos para el óptimo logro de los objetivos.

Nombre	Responsabilidad	Paquete	Discusión
Elemento	Super clase para todos los objetos de juego, define los métodos y atributos que les son comunes.	*Tonalli.client.estructura.elementos*	
Personaje	Representa a un jugador dentro del juego.		*Extiende a la clase Elemento*
PerfilPersonaje	Contiene y define a los atributos fijos de la descripción de un Personaje		*Integra y describe a la clase Personaje*
EstadoPersonaje	Contiene y define a los atributos variables de la descripción de un Personaje		*Integra y describe a la clase Personaje*
Item	Representa un objeto inanimado con el que se puede tener interacción.		*Extiende a la clase Elemento*
Magia	Representa una actividad de ataque o defensa que puede utilizar un personaje.		*Extiende a la clase Elemento*
Casilla	Representa un espacio o zona del juego. Puede contener elementos por medio de su atributo Ambiente.		*Extiende a la clase Elemento*
Ambiente	Representa el conjunto de Elementos que conviven en una casilla.		*Integra y describe a la clase Casilla*

Clase Elemento.

Clase padre de todos aquellos elementos de juego con los que es posible interactuar. Incluye los métodos y atributos que son comunes a cualquier elemento, aprovechando las características de herencia del lenguaje. Los personajes, las casillas, los ítems y las magias son agrupados como elementos por lo que extienden a esta Clase.

Clase: Personaje.

Representa a un personaje. Contiene la información para identificar a un personaje, y, de forma similar a la organización de la información en la base de datos, esta clase instancia a las clases PerfilPersonaje, EstadoPersonaje, Inventario y ConocimientoMágico, a manera de atributos del personaje, con lo que se hace posible el acceso a toda información relevante referente al personaje desde la clase que lo representa. Extiende a la clase Elemento para definirse como elemento de juego y heredar los atributos y métodos de un elemento. Implementa a la interfaz Serializable lo que permite que sea enviada a través de la red como un objeto valor.

Clase: PerfilPersonaje.

Incluido como atributo de la clase Personaje, contiene la información que describe las características de un personaje. Implementa a la interfaz Serializable lo que permite que sea enviada a través de la red como un objeto valor.

Clase EstadoPersonaje.

Incluido como atributo de la clase Personaje, contiene la información que define la situación del personaje, con respecto a los niveles que posee en atributos variables relacionados con las acciones del juego. Implementa a la interfaz Serializable lo que permite que sea enviada a través de la red como un objeto valor.

Clase: Inventario.

Anteriormente se había definido esta clase como un contenedor de ítems, refiriéndose a el inventario del juego. En este caso se refiere a el conjunto de ítems relacionados con el personaje, es decir los objetos que porta actualmente el personaje (que es un subconjunto del inventario del juego). En ambos casos se utiliza una sola clase, aprovechando su funcionalidad. Implementa a la interfaz Serializable lo que permite que sea enviada a través de la red como un objeto valor.

Clase: ConocimientoMagico.

Anteriormente se había definido esta clse como un contenedor de magias, refiriéndose al conocimiento mágico del juego. En este caso se refiere al conjunto de magias que el personaje conoce y es capaz de utilizar (que es un subconjunto del conocimiento mágico del juego). En ambos casos se utiliza una sola clase, aprovechando su funcionalidad. Implementa a la interfaz Serializable lo que permite que sea enviada a través de la red como un objeto valor.

Clase: Casilla.

Contiene la información que describe a una casilla del tablero y sus características. Las casillas pueden contener tanto ítems como personajes, esta información se conoce como ambiente de la casilla. Por lo anterior se crea una instancia de la clase Ambiente, como atributo de la clase, misma que es capaz de informar sobre qué elementos contiene la casilla.
Extiende a la clase Elemento para definirse como elemento de juego y heredar los atributos y métodos de un elemento. Implementa a la interfaz Serializable lo que permite que sea enviada a través de la red como un objeto valor.

Clase: Ambiente.

Contiene las colecciones de ítems y de personajes relacionados con una casilla. Representa la información variable de una casilla. Esta información no se almacena en la base de datos ya que las posiciones iniciales de los elementos de juego se calculan para cada juego creado. Implementa a la interfaz Serializable lo que permite que sea enviada a través de la red como un objeto valor.

Clase: Item.

Contiene la información que describe las características de un ítem en el juego.
Extiende a la clase Elemento para definirse como elemento de juego y heredar los atributos y métodos de un elemento. Implementa a la interfaz Serializable lo que permite que sea enviada a través de la red como un objeto valor.

Clase: Magia

Contiene la información que describe las características de una magia en el juego. Extiende a la clase Elemento para definirse como elemento de juego y heredar los atributos y métodos de un elemento. Implementa a la interfaz Serializable lo que permite que sea enviada a través de la red como un objeto valor.

Desarrollo de los beans de entidad.

Antes de comenzar a explicar lo que es un bean de entidad examinaremos distintas formas en las que se puede manejar la persistencia de la información, ya que esto nos dará una idea de la utilidad y funcionamiento de los beans de entidad.

La persistencia no es más que el almacenamiento de un objeto de forma más o menos permanente, para que pueda ser consultado posteriormente. Así, al hablar de persistencia, lo primero que puede venir a la mente es una Base de Datos, ya que en ella se almacena o persiste información con el objeto de poder consultarla posteriormente. Cabe hacer mención de que, además de las Bases de Datos, tanto Relacionales como Orientadas a Objetos, existen otras formas de manejar la persistencia de información, y en este caso en particular de objetos Java. Uno de estos enfoques es la Serialización de objetos Java.

La serialización no es más que la conversión de una clase (su comportamiento o métodos y atributos o estado) en un flujo de bytes que puede ser transmitido por la red (como funciona en RMI) o almacenado en un sistema de archivos. Sin embargo, si lo que se desea es una persistencia más sofisticada, entonces la serialización no es una opción, debido a que se consume mucho tiempo en la serialización-de-serialización de los objetos.
Examinemos ahora los dos enfoques restantes para la persistencia de información:

Mapeo Objeto-Relacional.

Este esquema de persistencia hace uso de las bases de datos relacionales, tales como Oracle o Microsoft SQL Server. Aquí, en lugar de serializar un objeto, se descompone éste en sus partes y estas se almacenan por separado, es decir en distintos campos de una o más tablas de la base de datos.

Para poder realizar este mapeo se hace uso de JDBC. De igual forma cuando se quiere componer un objeto con base en los valores almacenados en tablas, se debe crear una instancia de la clase, leer la base de datos y llenar los campos de dicha instancia con los valores leídos de la base de datos.

Bases de Datos Orientadas a Objetos.

Finalmente tenemos a los Sistemas Gestores de Bases de Datos Orientados a Objetos u OODBMS por sus siglas en inglés, los cuales son un almacén donde los objetos y sus relaciones son los tipos de datos básicos con los que trabaja. Debido a esto, no se requiere programar sobre un API relacional, sino que se debe trabajar directamente sobre el API de la base de datos.

Aunado a esto, muchos de estos OODBMS cuentan con herramientas que facilitan las consultas a los objetos almacenados, siendo una de estas, el lenguaje OQL o Lenguaje de Consultas de Objetos que permite recuperar información de forma fácil y sencilla de forma similar a SQL en las bases de datos relacionales.

Ahora que se han visto al menos dos de los ejemplos más utilizados en cuanto a la persistencia de la información manejada por un sistema definiremos lo que es un bean de entidad.

Un bean de entidad es aquel componente distribuido que tiene el conocimiento para poder representarse a sí mismo en un medio persistente utilizando alguno de los esquemas vistos (mapeo entidad-relación, OODMS o serialización). Este tipo de objetos representan los datos o información que queremos que se almacene.

A diferencia de los beans de sesión que se encargan de modelar los procesos o verbos de un sistema, los beans de entidad modelan los datos o sustantivos del sistema. En este punto es preciso hacer notar que el término bean de entidad se utilizará indistintamente para designar tanto a la instancia del bean de entidad, como a los datos que representa dicho objeto.

Características.

Las características principales de los beans de entidad son:

Supervivencia a fallos. Debido a que son objetos persistentes, estos pueden sobrevivir a fallos críticos tanto en el sistema como en la base de datos. En contraste, como los beans de sesión son objetos que permanecen en memoria, un fallo en el servidor implicaría perderlos por completo, mientras que en el mismo caso, un bean de entidad puede ser reconstruido simplemente con leer los valores de la base de datos.

Son una vista de la base de datos. Cuando se carga un bean de entidad, se debe leer la información almacenada en la base de datos[51], sin embargo, debe verse a un bean de entidad como la base de datos y viceversa, ya que todo cambio realizado en los valores de un bean de entidad se verán reflejados en la base de datos, mientras que todo cambio realizado a la base de datos, se verá reflejado en los valores que tome el bean.

Debido a que los beans de entidad son una vista de la base de datos, al momento de crear una nueva instancia de un bean de entidad, realmente lo que se está haciendo es insertar un nuevo registro (o representación de datos) en la base de datos. Análogamente, cuando se elimina un bean de entidad, lo que se hace realmente es eliminar un registro de la base de datos.

Múltiples instancias del bean pueden representar los mismos datos. El esquema adoptado por EJB es que cuando múltiples clientes requieran acceder a los mismos datos, el contenedor es el encargado de instanciar todas las copias necesarias de beans de entidad para satisfacer las peticiones realizadas. Cada una de estas instancias están sincronizadas con la base de datos para asegurarse de que los datos que representan en todo momento son los más actualizados. Este esquema se logra mediante el manejo de las transacciones que permite aislar cada petición de forma que los clientes crean que están tratando con una sola instancia de un bean.

Las instancias de beans de entidad pueden ser reutilizadas. Las instancias de los beans de entidad son objetos reciclables que pueden ser reutilizados

[51] Un bean de entidad no sólo puede hacer referencia a los datos contenidos en una tabla, sino que puede estar formado por datos recolectados de múltiples tablas.

dependiendo de las políticas del servidor, con esto se evitan largos tiempos de carga creando y destruyendo cuantas instancias sean necesarias, además de que permite que el contenedor administre de forma adecuada la cantidad de recursos con los que puede estar trabajando.

Los beans de entidad pueden ser encontrados. Debido a que los datos de un bean de entidad se encuentran identificados de forma única por los datos almacenados a los que hace referencia. Así, encontrar un bean de entidad es análogo a realizar una sentencia SELECT en SQL. De igual forma como se pueden tener múltiples sentencias SELECT sobre una tabla para encontrar datos de acuerdo a distintos criterios, de la misma forma se pueden tener múltiples formas de encontrar un bean de acuerdo a criterios específicos. Así, los beans de entidad cuentan con métodos denominados *finders* que establecen los parámetros que se requieren para poder encontrar la información deseada.

Estos métodos finder pueden regresar la referencia a un solo bean o a múltiples beans, dependiendo de la funcionalidad que se les desee dar. En el segundo caso, el objeto regresado debe ser del tipo `java.util.Collection` que no es más que una Colección de referencias a los beans que cumplan con los requisitos establecidos por los parámetros que se pasan al método.

Se pueden modificar los datos de un bean de entidad sin necesidad de utilizar EJB. Generalmente la creación, búsqueda, actualización y destrucción de los datos de un bean de entidad se realiza mediante el objeto Home, sin embargo, no es la única forma de interactuar con los datos. Debido a que los beans de entidad dependen de un almacén de datos (base de datos), se puede modificar directamente la base de datos mediante algún otro medio y dichos cambios se verán reflejados en los valores que toma el bean de entidad.

Al igual que con los beans de sesión, los beans de entidad ofrecen al desarrollador opciones para que éstos elijan la que más le conviene utilizar de acuerdo a una situación en particular. En el caso de los beans de entidad, estos ofrecen dos formas de mantener la persistencia:

- Persistencia manejada por el bean (BMP)[52]. En este caso, es el propio bean quien debe encargarse de realizar todas las llamadas a la base de datos para poder crear, buscar, actualizar y borrar los datos que mapea. Para lograr esta funcionalidad, debe hacerse uso del API JDBC. Durante el desarrollo del presente proyecto, este es el modelo de persistencia utilizado.
- Persistencia manejada por el contenedor (CMP)[53]. En este caso, como su nombre lo indica, es el contenedor en lugar del bean el encargado de manejar todo lo relacionado con la creación, búsqueda, actualización y borrado de los datos. Así, mediante el descriptor de despliegue, nosotros indicamos al contenedor la forma en la que queremos que se implementen estas operaciones y el contenedor es quien se encarga de generar el código con base en los parámetros que nosotros indicamos. Si se opta por este modo de manejar la persistencia, el tamaño del código de la clase del bean se reduce de forma radical, sin embargo, no permite la flexibilidad y el sentimiento de control sobre los datos que ofrece el enfoque BMP.

[52] Bean Managed Persistance en inglés.
[53] Container Manager Persistance en inglés.

Antes de comenzar con la demostración de la codificación de los beans de entidad se explicarán los métodos que deben ser implementados por la clase del bean y la forma en la que deben ser implementados bajo el esquema BMP y CMP.

Método	Explicación	Implementación	
		Persistencia Manejada por el Bean	**Persistencia Manejada por el Contenedor**
setEntityContext()	Asocia a un bean con la información del contexto en el que trabaja el bean. Una vez invocado este método, el bean puede acceder a información sobre su ambiente.	Se almacena el valor del contexto en una variable de instancia, de forma que pueda ser consultado posteriormente. *La instancia del bean se encuentra en un pool, pero no está asociado a datos específicos de la base de datos.*	
ejbFind<...>(<...>) métodos finder.	Encargados de localizar uno o más ocurrencias de los datos a los que hace referencia el bean. Se debe definir al menos el método ejbFindByRpimaryKey() el cual requiere como argumento un objeto del tipo de la clase llave primaria que se va a utilizar.	Mediante el uso de algún API como JDBC se realiza una búsqueda en la base de datos. Al encontrar los datos buscados se regresan las **llaves primarias** con las que el contenedor se encarga de crear los objetos EJBObject para que el cliente pueda interactuar con éstos. *La instancia del bean ya no se encuentra en el pool, está asociado a datos específicos y a un objeto EJBObject.*	No se implementan ya que es el contenedor quien se encarga de todo lo relacionado con la búsqueda de la información. Se utiliza EJB-QL para determinar la forma en la que se deben encontrar los datos.
ejbSelect<...>(<...>)	Son métodos que ayudan a realizar consultas de forma interna por el bean, pero no están expuestos a los clientes.	No se implementan.	Se definen como métodos abstractos y se utiliza EJB-QL para determinar las consultas.
ejbHome<...>(<...>)	Estos son métodos que no son específicos a alguna instancia en particular de un bean de entidad. Estos son métodos especiales del negocio ya que son llamados de un bean que se encuentra en el pool antes de ser asociado a datos específicos.	Se realizan operaciones globales de la base de datos como por ejemplo contar un número de registros, etc.	Se realizan operaciones globales de la base de datos como por ejemplo contar un número de registros, etc. En este caso lo más sencillo es la utilización de JDBC, mientras que lo más adecuado es la utilización de métodos ejbSelect().

Método	Explicación	Implementación	
		Persistencia Manejada por el Bean	**Persistencia Manejada por el Contenedor**
ejbCreate(<...>) *Si no se desea que los clientes sean capaces de insertar nuevos datos en la base de dato vía EJB, entonces no debe definirse ningún método ejbCreate().*	Cuando se invoca este método, el contenedor invoca al método ejbCreate() de un bean en el pool y se insertan nuevos datos en la base de datos, además de que se asocian estos datos a un bean en particular. De igual forma se inicializan los valores adicionales del bean.	Se debe validar que los parámetros de inicialización son correctos y vía JDBC se insertan los datos en la base de datos. Equivale a una sentencia INSERT de SQL	No se crean registros en la base de datos. En este caso sólo se validan los argumentos recibidos y se invocan métodos set() para inicializar los valores del bean. El contenedor utiliza estos datos para crear los registros necesarios
ejbPostCreate(<...>)	Por cada método ejbCreate() definido, se debe definir su correspondiente ejbPostCreate().	El contenedor invoca a este método inmediatamente después de la llamada al método ejbCreate(). Generalmente este método se utiliza para completar la inicialización de las variables del bean relacionadas con parámetros relacionados con transacciones.	
ejbActivate()	Cuando un cliente realiza una llamada a un método del negocio, pero no existe un bean de entidad asociado a un objeto EJBObject, entonces el contenedor debe tomar el bean del pool y colocarlo en un estado en el que esté listo para aceptar las peticiones de los clientes.	Se adquieren todos aquellos recursos que el bean requiere para satisfacer las peticiones de un cliente en particular. *En este método no se lee la información de la base de datos, esto lo hace el método ejbLoad() que se invoca inmediatamente después del método ejbActivate().*	
ejbLoad()	El contenedor invoca a este método para cargar los datos de la base de datos en el bean (sentencia SELECT de SQL), con base en el estado transaccional actual.	Mediante una llamada al método getPrimaryKey() en el contexto de la entidad, el bean conoce que datos son los que debe cargar. Posteriormente, vía JDBC se leen los datos de la base de datos.	No se leen datos de la base de datos. En lugar de esto, es el contenedor quien se encarga de leer los datos de la base de datos.
ejbStore()	El contenedor invoca este método para actualizar los valores de los datos en memoria con los existentes en la base de datos. Este método es llamado antes de la pasivación.	Se actualiza de forma explícita la base de datos. Equivale a una sentencia UPDATE de SQL.	No se actualizan datos de la base de datos. En su lugar, el contenedor se encarga de realizar todas las operaciones para actualizar la base de datos.

Método	Explicación	Implementación	
		Persistencia Manejada por el Bean	**Persistencia Manejada por el Contenedor**
ejbPassivate()	El contenedor invoca este método cuando desea regresar un bean de entidad al pool.	Se liberan los recursos utilizados por el bean. No se almacenan los datos en la base de datos, esto lo hace el método ejbStore(), llamado antes de este método por el contenedor.	
ejbRemove()	Elimina los datos de la base de datos y no se utiliza para destruir el objeto Java.	Mediante la invocación al método getPrimaryKey() del contexto se determinan los datos que deben eliminarse y vía JDBC se eliminan los datos de la base de datos. Equivale a una sentencia DELETE de SQL.	No se eliminan registros de la base de datos. En lugar de esto, es el contenedor el encargado de eliminar éstos de la base de datos.
unsetEntityContext()	Desasocia al bean de su ambiente. El contenedor invoca este método antes de que la instancia del bean sea destruida.	Se liberan los recursos obtenidos durante la invocación al método setEntityContext().	

Clases e Interfaces.

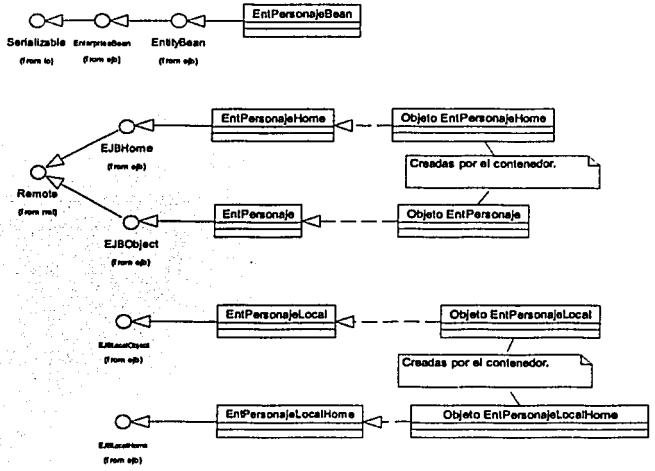


Ilustración 50 : Diagrama de clases de los beans de entidad.

Las reglas que se siguen para el desarrollo de beans de entidad son las mismas que para los beans de sesión, es decir, se requieren de todos aquellos objetos (`EJBHome` y `EJBObject`) que el cliente requiere para establecer una comunicación con el bean. Sin embargo, en el caso de los beans de entidad en algunas ocasiones se requiere de una clase especial denominada **clase de llave primaria.**[54]

Una clase de llave primaria es aquella que permite identificar de forma única a cada uno de los beans, puede hacerse una analogía con el concepto de llave primaria de una tabla bajo el enfoque relacional, en el que una llave primaria es un registro o conjunto de campos que identifican de forma única a cada registro.

Lo mismo sucede en el caso de los beans de entidad. Gracias a esta clase de llave primaria, el contenedor puede realizar todas aquellas operaciones relacionadas con la carga, actualización y borrado de la información. Similarmente a las bases de datos relaciones, estas clases pueden ser tan sencillas como las clases envolventes de los tipos de datos primitivos (Long, Integer, etc.), o clases más complejas definidas por el desarrollador.

[54] Del inglés primary key class.

Cuando se determina que se debe utilizar una clase propietaria para ser utilizada como clase llave primaria, esta sólo debe cumplir con los siguientes requisitos:

- Se requiere que implemente el método `public String toString()`, de forma que el contenedor pueda obtener una representación en cadena del valor de la llave primaria.
- Se requiere que implemente el método `public int hashCode()`. De esta forma, la clase de llave primaria puede ser almacenada en un objeto del tipo Hashtable; esto es requerido por el contenedor ya que es la forma en la que organiza las instancias de los beans que tiene en memoria.
- Se requiere de la implementación del método `public bolean equals(Object o)`. El contenedor lo requiere para poder hacer comparaciones entre clases de llaves primarias y determinar su igualdad.

Para efectos de este ejemplo se toma el bean `EntPersonajeBean` que es un bean con persistencia manejada por él mismo, encargado de mantener toda la información de los personajes que se encuentren dentro del juego.

Una de las principales razones por las que se utiliza este ejemplo es para demostrar una de las capacidades de los beans de entidad de que en un solo bean se pueda hacer referencia a múltiples tablas. En este caso, el bean `EntPersonajeBean` hace no sólo referencia a la tabla Personaje, sino que además debe obtener información de las tablas EstadoPersonaje, PerfilPersonaje, Inventario y ConocimientoMagico.

Interfaz Remota. En el caso de los beans de entidad la interfaz remota incluye todos los métodos que el bean expone al cliente. En este caso, pueden existir métodos get() o set() para poder asignar y/u obtener valores específicos del bean de entidad. De la misma forma que con los beans de sesión, esta interfaz debe extender al objeto `javax.ejb.EJBObject`.

Cada uno de los métodos definidos en esta interfaz deben lanzar la excepción `java.rmi.RemoteException` para poder realizar las gestiones necesarias en caso de que exista algún problema con la comunicación por la red. Así mismo, cada uno de los parámetros pasados a los métodos, así como los tipos de datos de los valores que regresan, deben implementar la interfaz `java.io.Serializable`, de forma que se asegure su correcto transporte a través de la red.

```
package tonalli.servidor;

import java.rmi.*;
import javax.ejb.*;
import java.util.*;
import tonalli.estructura.ConocimientoMagico;
import tonalli.estructura.elementos.*;
import tonalli.estructura.Inventario;

public interface EntPersonaje extends EJBObject {

    /*Metodos para asociar valores*/
    public void setPuntosSalud(int intPuntosSalud) throws
            RemoteException;
    public void setPuntosMagia(int intPuntosMagia) throws
            RemoteException;
```

```
    public void setAgilidad(int intAgilidad) throws RemoteException;
    public void setExperiencia(int intExperiencia) throws
            RemoteException;
    public void setResistencia(int intResistencia) throws
            RemoteException;
    public void setFuerza(int intFuerza) throws RemoteException;
    public void setSuerte(int intSuerte)throws RemoteException;
    public void setAgregaItemInventario(Item item) throws
            RemoteException;
    public void setAgregaMagiaConMagico(Magia magia) throws
            RemoteException;
    public void setEliminaItemInventario(Item item) throws
            RemoteException;
    public void setEliminaMagiaConMagico(Magia magia) throws
            RemoteException;

    /*Metodos para obtener valores*/
    public int getIDPersonaje() throws RemoteException;
    public int getPuntosSalud() throws RemoteException;
    public int getPuntosMagia() throws RemoteException;
    public int getAgilidad() throws RemoteException;
    public int getExperiencia() throws RemoteException;
    public int getResistencia() throws RemoteException;
    public int getFuerza() throws RemoteException;
    public int getSuerte() throws RemoteException;
    public String getNombre() throws RemoteException;
    public String getDescripcion() throws RemoteException;
    public String getAvatar() throws RemoteException;
    public String getSexo() throws RemoteException;
    public String getColor() throws RemoteException;
    public String getRaza() throws RemoteException;
    public String getEdad() throws RemoteException;
    public Inventario getInventario() throws RemoteException;
    public ConocimientoMagico getConocimientoMagico() throws
            RemoteException;
    public Personaje getPersonaje() throws RemoteException;
}
```

Interfaz Home. La interfaz Home es aquella que contiene todos los métodos necesarios para poder crear nuevos datos en la base de datos (en caso de que se desee ofrecer dicho servicio). Al igual que en los beans de sesión, esta interfaz debe extender al objeto `javax.ejb.EJBHome` y cada uno de los métodos `create()` que definamos debe lanzar la excepción `javax.ejb.CreateException`, además de que cada uno de los métodos definidos en esta interfaz, debe lanzar la excepción `java.rmi.RemoteException`.

Específicamente en el caso de los beans de entidad, es en esta interfaz donde se definen cada uno de los métodos finder, es decir, aquellos que permiten realizar las búsquedas de información en la base de datos y los métodos `ejbHome()`, es decir, todos aquellos métodos que realizan operaciones genéricas sobre la base de datos y que no requieren estar asociados a datos específicos.

```
package tonalli.servidor;

import java.rmi.*;
import javax.ejb.*;
import java.util.Collection;
import tonalli.estructura.ConocimientoMagico;
import tonalli.estructura.elementos.*;
import tonalli.estructura.Inventario;
```

```
public interface EntPersonajeHome extends javax.ejb.EJBHome {

    /*Metodos create()*/
    public EntPersonaje create(Personaje personaje) throws
            RemoteException,

CreateException;
    public EntPersonaje create(String strNombrePersonaje,
                               String strDescripcion,
                               String strAvatar, int intPuntosSalud,
                               int intPuntosMagia, int intAgilidad,
                               int intResistencia, int
                               intExperiencia,
                               int intSuerte, int intFuerza,
                               String strEdad, String strSexo,
                               String strColor, String strRaza)
                                  throws RemoteException,
                                         CreateException;

    /*Metodos Finder*/
    public EntPersonaje findByPrimaryKey(String primKey)
          throws ObjectNotFoundException, RemoteException,
                 FinderException;
    public Collection findAll() throws ObjectNotFoundException,
                                       RemoteException,
                                       FinderException;
    public EntPersonaje findByCuenta(String login)throws
                               ObjectNotFoundException,
                               RemoteException,
                               FinderException;

    /*Metodos ejbHome: Nota, este método no existe dentro de la clase)
      EntPersonajeHome, pero se añade para ejemplificar el uso de estos
      métodos.*/
    public int getTotalPersonajes() throws RemoteException;
}
```

En el caso de este bean de entidad se definen tres métodos finder:

- `public EntPersonaje findByPrimaryKey(String primKey)`. Recibe como argumento un objeto del tipo String, que en para esta clase es el objeto utilizado como clase llave primaria y regresa un objeto del tipo de la interfaz remota.
- *public Collection findAll(). No recibe argumentos y es equivalente a un SELECT *, es decir, regresar todos los registros. Se debe notar que regresa un objeto de tipo Collection el cual no es más que una Colección de objetos del tipo de la interfaz remota.*
- *public EntPersonaje findByCuenta(String login). Este método se utiliza para encontrar los datos relativos a un personaje pero con base en la cuenta del usuario que lo crea. De igual forma que el método findByPrimaryKey() regresa un objeto del tipo de la interfaz remota.*

Así mismo se cuenta con un método ejbHome: `getTotalPersonajes()`, el cual, como se ha visto, no requiere estar asociado a datos en particular y sólo regresa un entero que nos indica cuantos personajes existen.

Clase Llave Primaria. Para el bean de entidad *EntPersonajeBean* no se ha definido una clase llave primaria propia, sino que se ha optado por la utilización de objetos del tipo String para identificar de forma única a cada instancia de este bean. Esta decisión se tomó en base al diseño de la base de datos y al análisis realizado. Por una parte en la tabla Personaje observamos que el campo que se utiliza como llave primaria es de tipo entero. Generalmente, para tablas únicas que se mapean a beans de entidad, el tipo de dato del campo que es la llave primaria, es el Objeto que se utiliza como clase llave primaria. Sin embargo, en este caso, como parte del análisis se ha determinado que no pueden existir dos nombres de personajes iguales, por lo tanto, lo que nos permite identificar de forma única a cada bean de entidad de un personaje es su nombre y no el número consecutivo que se le asigna.

Clase del Bean. En el caso de los beans de entidad que manejan su propia persistencia, estos deben, además de implementar todos los métodos definidos en la interfaz remota, ser capaces de manejar mediante un API como JDBC todos los accesos a la base de datos que les permita crear, actualizar y eliminar datos de ésta. Esta clase de cumplir con los siguientes requisitos:

Debe implementar la interfaz `javax.ejb.EntityBean`.

Debe contar con variables en las que se almacenen los datos que se mapean de la base de datos.

Debe implementar los métodos de la lógica del negocio, es decir, aquellos especificados en la interfaz remota y aquellos métodos finder y ejbHome definidos en la interfaz Home.

Debe implementar los métodos requeridos por la especificación EJB y que permiten la inserción, actualización y borrado de datos de la base de datos.

Para comprender mejor el código, este se presenta en cuatro partes:

Parte 1.

- *Variables.* En la parte inicial se declaran todas las variables que se encargan de mapear los datos de la base de datos en el bean. En el caso de este bean toda la información de la base de datos se va a mapear a un objeto de tipo `Personaje` en la variable `personaje`. Así mismo debe prestarse especial atención a la variable `_dataSource` que es la que permite encontrar la fuente de datos en la que se deben realizar todas las operaciones con éstos.

- *Métodos ejbCreate().* Para cada uno de los métodos `create()` definidos en la interfaz Home, se debe tener su correspondiente método `ejbCreate()` en la clase del bean. Debe notarse que a diferencia de los métodos en la interfaz Home, los métodos del bean no regresan un objeto del tipo de la interfaz remota, sino que regresan un objeto del tipo de la clase llave primaria, en este caso, de tipo `String`.

La estructura de los métodos `ejbCreate()` es muy similar y se siguen los siguientes pasos:

1. Se obtiene una conexión a la base de datos[55].
2. Se valida que los datos que se desean insertar no existan previamente en la base de datos ya que si es así se debe lanzar una excepción del tipo `javax.ejb.CreateException`. Si los datos que se desean insertar no existen, entonces se preparan todos los Statements y se ejecutan para realizar la inserción en cada una de las tablas a las que el bean haga referencia.
3. Una vez insertados los registros, se actualiza el valor de las variables que mapean los datos a la base de datos, en este caso, a la variable `personaje` se le asigna un `Personaje` con la información que se insertó en la base de datos.
4. Finalmente se cierran las conexiones a la base de datos y se regresa el valor que identifica de forma única a este bean (clase llave primaria). En este caso se regresa el nombre del `Personaje` que se ha creado.

Todo lo anterior debe englobarse en un `try {...}` para poder tratar cualquier problema que se haya presentado con la base de datos y poder enviar dicha información al usuario.

```
package tonalli.servidor;

import java.util.Collection;
import java.util.Iterator;
import java.util.ArrayList;
import java.rmi.*;
import javax.ejb.*;
import java.sql.*;
import javax.sql.*;
import javax.naming.*;
import tonalli.estructura.elementos.Personaje;
import tonalli.estructura.elementos.PerfilPersonaje;
import tonalli.estructura.elementos.EstadoPersonaje;
import tonalli.estructura.ConocimientoMagico;
import tonalli.estructura.elementos.Magia;
import tonalli.estructura.elementos.Item;
import tonalli.estructura.Inventario;

public class EntPersonajeBean implements EntityBean {
    /*Se declaran todas las variables que serán utilizadas por el bean.*/
    EntityContext entityContext;
    DataSource _dataSource;
    int _iIDPersonaje;
    Personaje _personaje;

    public String ejbCreate(Personaje personaje) throws CreateException {
        Connection conexion = null;
        PreparedStatement creaPersonaje = null;
        int idPersonaje=0;
        try{
            conexion = _dataSource.getConnection();
            creaPersonaje = conexion.prepareStatement("Select per_Id FROM "+
                                " \"Personaje\" WHERE per_Nombre = ?");
            creaPersonaje.setString(1,personaje.obtenNombre());
            ResultSet llavePrimaria = creaPersonaje.executeQuery();
            if(llavePrimaria.next()){
                throw new CreateException("El personaje ya existe");
            }
            llavePrimaria.close();

            creaPersonaje = conexion.prepareStatement("SELECT MAX(per_Id)"+
                                                    " FROM Personaje");
```

[55] Se toma en cuenta que el lector cuenta con los conocimientos básicos en cuanto al uso del API JDBC para conexión a base de datos.

```
        ResultSet rsIdPersonaje = creaPersonaje.executeQuery();
        if(rsIdPersonaje.next()){
            int maxIdPersonaje = rsIdPersonaje.getInt(1);
            idPersonaje = maxIdPersonaje+1;
        }
        creaPersonaje = conexion.prepareStatement("INSERT INTO "+
                                  "Personaje (per_Id,per_Nombre,"+
                                  "per_Descripcion,per_Avatar) "+
                                  "VALUES (?,?,?,?)");
        creaPersonaje.setInt(1,idPersonaje);
        creaPersonaje.setString(2,personaje.obtenNombre());
        creaPersonaje.setString(3,personaje.obtenDescripcion());
        creaPersonaje.setString(4,personaje.obtenAvatar());

        if(creaPersonaje.executeUpdate()!=1){
            throw new SQLException ("No es inserto registro en Personaje");
        }

        creaPersonaje = conexion.prepareStatement("INSERT INTO" +
                                        "EstadoPersonaje "+
                        "(fk_per_Id,edo_PuntosSalud,edo_PuntosMagia,"+
                        "edo_Agilidad,edo_Resistencia,"+
                        "edo_Experiencia,edo_Suerte,edo_Fuerza)"+
                        " VALUES (?,?,?,?,?,?,?,?)");

        creaPersonaje.setInt(1,idPersonaje);
        creaPersonaje.setInt(2,personaje.obtenEstado()._iPuntosSalud);
        creaPersonaje.setInt(3,personaje.obtenEstado()._iPuntosMagia);
        creaPersonaje.setInt(4,personaje.obtenEstado()._iAgilidad);
        creaPersonaje.setInt(5,personaje.obtenEstado()._iResistencia);
        creaPersonaje.setInt(6,personaje.obtenEstado()._iExperiencia);
        creaPersonaje.setInt(7,personaje.obtenEstado()._iSuerte);
        creaPersonaje.setInt(8,personaje.obtenEstado()._iFuerza);

        if(creaPersonaje.executeUpdate()!=1){
            throw new SQLException ("No se inserto registro en Estado " +
                                    "Personaje");
        }

        creaPersonaje = conexion.prepareStatement("INSERT INTO "+
                                              "PerfilPersonaje "+
                                    "(fk_per_Id,ppe_Edad,ppe_Sexo,"+
                                    "ppe_Color,ppe_Raza) "+
                                    "VALUES (?,?,?,?,?)");
        creaPersonaje.setInt(1,idPersonaje);
        creaPersonaje.setInt(2,
                Integer.parseInt(personaje.obtenPerfil().obtenEdad()));
        creaPersonaje.setString(3,
                personaje.obtenPerfil().obtenSexo().substring(0,0));
        creaPersonaje.setString(4,personaje.obtenPerfil().obtenColor());
        creaPersonaje.setString(5,personaje.obtenPerfil().obtenRaza());

        if(creaPersonaje.executeUpdate()!=1){
              throw new SQLException ("No se inserto registro en Perfil");
        }

        this._iIDPersonaje = idPersonaje;
        this._personaje = new Personaje(personaje.obtenNombre(),
        personaje.obtenDescripcion(),personaje.obtenPerfil(),
        personaje.obtenAvatar(),personaje.obtenEstado());

    }catch(Exception e){
        throw new CreateException("No se registro personaje" +
                                  e.toString());
    }finally{
        closeConnection(conexion,creaPersonaje);
        return this._personaje.obtenNombre();
    }
}

public String ejbCreate(String strNombrePersonaje, String strDescripcion,
```

```
                        String strAvatar,int intPuntosSalud,
                        int intPuntosMagia, int intAgilidad,
                        int intResistencia, int intExperiencia,
                        int intSuerte, int intFuerza, String strEdad,
                        String strSexo,String strColor,
                        String strRaza)
                                throws CreateException {

Connection conexion = null;
PreparedStatement creaPersonaje = null;
int idPersonaje=0;
try{
    conexion = _dataSource.getConnection();
    creaPersonaje = conexion.prepareStatement("Select per_Id FROM "+
                    " \"Personaje\" WHERE per_Nombre = ?");
    creaPersonaje.setString(1,strNombrePersonaje);
    ResultSet llavePrimaria = creaPersonaje.executeQuery();
    if(llavePrimaria.next()){
        throw new CreateException("El personaje ya existe");
    }
    llavePrimaria.close();

    creaPersonaje = conexion.prepareStatement("SELECT MAX(per_Id)"+
                    " FROM Personaje");
    ResultSet rsIdPersonaje = creaPersonaje.executeQuery();
    if(rsIdPersonaje.next()){
        int maxIdPersonaje = rsIdPersonaje.getInt(1);
        idPersonaje = maxIdPersonaje+1;
    }

    creaPersonaje = conexion.prepareStatement("INSERT INTO "+
                    "Personaje (per_Id,per_Nombre,"+
                    "per_Descripcion,per_Avatar) "+
                    "VALUES (?,?,?,?)");
    creaPersonaje.setInt(1,idPersonaje);
    creaPersonaje.setString(2,strNombrePersonaje);
    creaPersonaje.setString(3,strDescripcion);
    creaPersonaje.setString(4,strAvatar);
    if(creaPersonaje.executeUpdate()!=1){
        throw new SQLException ("No es inserto registro en Personaje");
    }

    creaPersonaje = conexion.prepareStatement("INSERT INTO "+
                                    "EstadoPersonaje "+
                    "(fk_per_Id,edo_PuntosSalud,edo_PuntosMagia,"+
                    "edo_Agilidad,edo_Resistencia,"+
                    "edo_Experiencia,edo_Suerte,edo_Fuerza)"+
                    " VALUES (?,?,?,?,?,?,?,?)");
    creaPersonaje.setInt(1,idPersonaje);
    creaPersonaje.setInt(2,intPuntosSalud);
    creaPersonaje.setInt(3,intPuntosMagia);
    creaPersonaje.setInt(4,intAgilidad);
    creaPersonaje.setInt(5,intResistencia);
    creaPersonaje.setInt(6,intExperiencia);
    creaPersonaje.setInt(7,intSuerte);
    creaPersonaje.setInt(8, intFuerza);

    if(creaPersonaje.executeUpdate()!=1){
        throw new SQLException ("No se inserto registro en Estado "+
                        "Personaje");
    }

    creaPersonaje = conexion.prepareStatement("INSERT INTO "+
                                    "PerfilPersonaje "+
                                    "(fk_per_Id,ppe_Edad,"+
                    "ppe_Sexo,ppe_Color,ppe_Raza) "+
                                    "VALUES (?,?,?,?,?)");
    creaPersonaje.setInt(1,idPersonaje);
    creaPersonaje.setInt(2,Integer.parseInt(strEdad));
    creaPersonaje.setString(3,strSexo.substring(0,0));
    creaPersonaje.setString(4,strColor);
```

```
            creaPersonaje.setString(5,strRaza);

            if(creaPersonaje.executeUpdate()!=1){
                  throw new SQLException ("No se inserto registro en Perfil");
            }

            this._iIDPersonaje = idPersonaje;
            this._personaje = new Personaje( strNombrePersonaje,strDescripcion,
                                    new PerfilPersonaje(strRaza,strColor,
                                    strEdad,strSexo),strAvatar,
                                    new EstadoPersonaje( intPuntosSalud,
                                    intPuntosMagia,intResistencia,
                                    intFuerza, intAgilidad,
                                    intExperiencia, intSuerte));

        }catch(Exception e){
            throw new EJBException("No se pudo insertar registro para "+
                                    "crear Personaje "+e.toString());
        }finally{
            closeConnection(conexion,creaPersonaje);
            return this._personaje.obtenNombre();
        }
    }

    public void ejbPostCreate(Personaje personaje) throws CreateException {
    }

    public void ejbPostCreate(String strNombrePersonaje, String strDescripcion,
                                    String strAvatar,int intPuntosSalud,
                                    int intPuntosMagia, int intAgilidad,
                                    int intResistencia, int intExperiencia,
                                    int intSuerte, int intFuerza, String strEdad,
                                    String strSexo,String strColor,
                                    String strRaza )
                                                throws CreateException{
    }
```

Parte 2.

- *Métodos* `ejbLoad()`, `ejbStore()` *y* `ejbRemove()`. Estos son los métodos que se utilizan para la carga, almacenamiento y eliminación de registros de la base de datos. Su estructura es muy similar a la de los métodos `ejbCreate()`, ya que primero se debe establecer una conexión con la base de datos, para posteriormente cargar/almacenar/remover los datos de las variables persistentes del bean en la base de datos. En el caso de los métodos `ejbLoad()` y `ejbStore()`, estos lanza una excepción del tipo `javax.ejb.EJBException` en caso de que encuentren algún problema durante su ejecución. Por su parte, el método `ejbRemove()` lanza una excepción del tipo `javax.ejb.RemoveException()`.

- *Métodos* `ejbActivate()` *y* `ejbPassivate()`. En este caso, como el bean no utiliza recursos que deban ser adquiridos o liberados durante la pasivación y activación, no se implementan.

- *Métodos* `setEntityContext()` y `unsetEntityContext()`. En el caso del método `setEntityContext()`, es aquí donde obtenemos la referencia a la fuente de datos en la que se encuentra la base de datos. Esto se hace mediante el siguiente código:

```
        try{
            Context contexto = new InitialContext();
            try{
                _dataSource = (DataSource)contexto.lookup("java:comp/env/jdbc/"+
                                                          "dsTonalli");
            } catch(Exception e){
                throw new EJBException("No se pudo encontrar la base de datos :
" + e);
            }
        } catch(Exception e) {...}
```

En primer lugar se adquiere un contexto inicial, para posteriormente hacer la búsqueda (lookup) de la fuente de datos y asignarla a la variable _dataSource.

```
    public void ejbLoad() {
        Connection conexion = null;
        PreparedStatement stmtConsultaDatos = null;
        EstadoPersonaje estado = null;
        PerfilPersonaje perfil = null;
        Personaje personaje = null;
        String strDescripcion="";
        String strAvatar="";
        /*Se obtiene la el valor de la llave primaria de los datos que se
          quieren cargar*/
        String strNombre = ((String) this.entityContext.getPrimaryKey()).trim();

        try{
            conexion = _dataSource.getConnection();
            stmtConsultaDatos = conexion.prepareStatement("SELECT * "+
                                                    "FROM Personaje "+
                                                    "WHERE per_Nombre = ?");
            stmtConsultaDatos.setString(1,strNombre);
            ResultSet datosPersonaje = stmtConsultaDatos.executeQuery();
            if(datosPersonaje.next()){
                this._iIDPersonaje = datosPersonaje.getInt(1);
                strDescripcion =  datosPersonaje.getString(3).trim();
                strAvatar = datosPersonaje.getString(4).trim();
            }
            datosPersonaje.close();
            stmtConsultaDatos = conexion.prepareStatement("SELECT "+
                            "edo_PuntosSalud, edo_PuntosMagia, "+
                            "edo_Resistencia, edo_Fuerza, edo_Agilidad," +
                            " edo_Experiencia,edo_Suerte FROM EstadoPersonaje"+
                            " WHERE fk_per_Id = ?" );
            stmtConsultaDatos.setInt(1,this._iIDPersonaje);

            ResultSet datosEstado = stmtConsultaDatos.executeQuery();

            if(datosEstado.next()){
                estado = new EstadoPersonaje(datosEstado.getInt
                                                    ("edo_PuntosSalud"),
                                    datosEstado.getInt("edo_PuntosMagia"),
                                    datosEstado.getInt("edo_Resistencia"),
                                    datosEstado.getInt("edo_Fuerza"),
                                    datosEstado.getInt("edo_Agilidad"),
                                    datosEstado.getInt("edo_Experiencia"),
                                    datosEstado.getInt("edo_Suerte"));
            }
            datosEstado.close();
            stmtConsultaDatos = conexion.prepareStatement("SELECT ppe_Edad,"+
                                                    "ppe_Color,ppe_Raza,"+
                                                    "ppe_Sexo FROM "+
                                                    "\"PerfilPersonaje\" "+
                                                    "WHERE fk_per_Id = ?");
            stmtConsultaDatos.setInt(1,this._iIDPersonaje);
            ResultSet datosPerfil = stmtConsultaDatos.executeQuery();
            if(datosPerfil.next()){
              int Edad = datosPerfil.getInt(1);
```

```
            String strColor = datosPerfil.getString(2).trim();
            perfil = new PerfilPersonaje(datosPerfil.getString(3).trim(),
                                         strColor.trim(),
                                String.valueOf(Edad),datosPerfil.getString(4));
        }
        datosPerfil.close();
        stmtConsultaDatos = conexion.prepareStatement("SELECT * FROM "
                                                      +"Inventario" +
                                                      " WHERE fk_Per_Id = ?" );
        stmtConsultaDatos.setInt(1,this._iIDPersonaje);

        ResultSet datosInventario =stmtConsultaDatos.executeQuery();
        stmtConsultaDatos.close();
        Collection alListaNombreItems= new ArrayList();
        while(datosInventario.next()){
            alListaNombreItems.add( datosInventario.getString(1).trim());
       }
        datosInventario.close();
        Collection alListaItems = new ArrayList();
        Iterator itAlListaNombreItems = alListaNombreItems.iterator();
        while(itAlListaNombreItems.hasNext()){
            String strNombreItem = (String) itAlListaNombreItems.next();
            stmtConsultaDatos = conexion.prepareStatement(
                    "SELECT ite_EfectoSalud,ite_EfectoMagia,"+
                    "ite_EfectoResistencia,ite_EfectoFuerza,"+
                    "ite_EfectoAgilidad,ite_EfectoExperiencia,"+
                    "ite_EfectoSuerte,ite_Descripcion,ite_Avatar,"+
                    "ite_Costo FROM Item WHERE ite_Nombre = ?");
            stmtConsultaDatos.setString(1,strNombreItem);
            ResultSet datosItem = stmtConsultaDatos.executeQuery();

            while (datosItem.next()){
                int[] efecto = new int[7];
                efecto[0] = datosItem.getInt(1);
                efecto[1] = datosItem.getInt(2);
                efecto[2] = datosItem.getInt(3);
                efecto[3] = datosItem.getInt(4);
                efecto[4] = datosItem.getInt(5);
                efecto[5] = datosItem.getInt(6);
                efecto[6] = datosItem.getInt(7);
                alListaItems.add(new Item(strNombreItem,
                     datosItem.getString(8),datosItem.getString(9).trim(),
                                               datosItem.getInt(10),
                                                                 efecto));
            }
            datosItem.close();
        }

    mtConsultaDatos = conexion.prepareStatement("SELECT fk_Magia_Id FROM "
                                                +"ConocimientoMagico"+
                                                " WHERE fk_Per_Id = ?");
    stmtConsultaDatos.setInt(1,this._iIDPersonaje);
    ResultSet datosConocimiento = stmtConsultaDatos.executeQuery();
    stmtConsultaDatos.close();
    Collection alListaNombreMagias= new ArrayList();
    while(datosConocimiento.next()){
            alListaNombreMagias.add(datosConocimiento.getString(1).trim());
    }
    datosConocimiento.close();
    Collection alListaMagias = new ArrayList();
    Iterator itAlListaNombreMagias = alListaNombreMagias.iterator();
    while(itAlListaNombreMagias.hasNext()){
        String strNombreMagia = (String)itAlListaNombreMagias.next();
          stmtConsultaDatos = conexion.prepareStatement("SELECT "+
                              " mag_EfectoSalud,mag_EfectoMagia,"+
                              "mag_EfectoResistencia,mag_EfectoFuerza,"+
                              "mag_EfectoAgilidad,mag_EfectoExperiencia,"+
                              "mag_EfectoSuerte,"+
                              "mag_Costo,mag_Alcance,"+
                              "mag_Descripcion FROM Magia"+
                              " WHERE mag_Nombre = ?");
```

```
            stmtConsultaDatos.setString(1,strNombreMagia);
            ResultSet datosMagia = stmtConsultaDatos.executeQuery();
            while (datosMagia.next()){
                int[] efecto= new int[7];
                efecto[0]=datosMagia.getInt(1);
                efecto[1]=datosMagia.getInt(2);
                efecto[2]=datosMagia.getInt(3);
                efecto[3]=datosMagia.getInt(4);
                efecto[4]=datosMagia.getInt(5);
                efecto[5]=datosMagia.getInt(6);
                efecto[6]=datosMagia.getInt(7);
                int intCosto =  datosMagia.getInt(8);
                int intAlcance = datosMagia.getInt(9);
               String strDescripcionMagia = datosMagia.getString(10).trim();
               alListaMagias.add(new Magia(strNombreMagia,
                                intAlcance,strDescripcionMagia,
                                intCosto,efecto));
            }
            datosMagia.close();
            stmtConsultaDatos.close();
        }
        this._personaje = new Personaje(strNombre, strDescripcion,
                                        perfil,strAvatar,estado);

       this._personaje.obtenInventario().agregaItems(alListaItems);
       this._personaje.obtenConMagico().agregaMagias(alListaMagias);

    }catch(Exception e){
        e.printStackTrace();
        throw new EJBException("No se pudo realizar Load "+e.toString());
    }finally{
        closeConnection(conexion,stmtConsultaDatos);
    }
}

public void ejbStore() {
    Connection conexion = null;
    PreparedStatement stmtActualizaDatos = null;
    try{
        conexion = _dataSource.getConnection();
        stmtActualizaDatos = conexion.prepareStatement("UPDATE Personaje"+
                  " SET per_Nombre= ?,per_Descripcion = ? ,per_Avatar= ? "+
                  " WHERE per_Id = ?");
        stmtActualizaDatos.setString(1,this._personaje.obtenNombre());
        stmtActualizaDatos.setString(2,this._personaje.obtenDescripcion());
        stmtActualizaDatos.setString(3,this._personaje.obtenAvatar());
        stmtActualizaDatos.setInt(4,this._iIDPersonaje);
        if(stmtActualizaDatos.executeUpdate()!=1){
            throw new SQLException("No se actualizo personaje");
        }
        stmtActualizaDatos =conexion.prepareStatement("UPDATE \"Estado"+
                           "Personaje\" SET edo_PuntosSalud =?,"+
                           "edo_PuntosMagia = ?,edo_Agilidad = ?,"+
                           "edo_Resistencia = ?,edo_Experiencia = ?,"+
                           "edo_Suerte = ?, edo_Fuerza = ?"+
                           " WHERE fk_per_Id =?");
        stmtActualizaDatos.setInt(1,this._personaje.obtenEstado().
                                                    _iPuntosSalud);
        stmtActualizaDatos.setInt(2,this._personaje.obtenEstado().
                                                    _iPuntosMagia);
        stmtActualizaDatos.setInt(3,this._personaje.obtenEstado().
                                                    _iAgilidad);
        stmtActualizaDatos.setInt(4,this._personaje.obtenEstado().
                                                    _iResistencia);
        stmtActualizaDatos.setInt(5,this._personaje.obtenEstado().
                                                    _iExperiencia);
        stmtActualizaDatos.setInt(6,this._personaje.obtenEstado()._iSuerte);
        stmtActualizaDatos.setInt(7,this._personaje.obtenEstado()._iFuerza);
        stmtActualizaDatos.setInt(8,this._iIDPersonaje);

        if(stmtActualizaDatos.executeUpdate()!=1){
```

```
                throw new EJBException("No se pudo actualizar registro en Estado"+
                                       "Personaje ");
            }

            stmtActualizaDatos = conexion.prepareStatement("UPDATE "+
                                                " \"PerfilPersonaje\" SET "+
                                                " ppe_Edad = ?,ppe_Sexo = ?,"+
                                                "ppe_Color = ?,ppe_Raza =? "+
                                                "WHERE fk_per_Id = ?");

            stmtActualizaDatos.setInt(1,Integer.parseInt(
                                      this._personaje.obtenPerfil().
                                      obtenEdad()));
            stmtActualizaDatos.setString(2,this._personaje.obtenPerfil().
                                      obtenSexo());
            stmtActualizaDatos.setString(3,this._personaje.obtenPerfil().
                                      obtenColor());
            stmtActualizaDatos.setString(4,this._personaje.obtenPerfil().
                                      obtenRaza());
            stmtActualizaDatos.setInt(5,this._iIDPersonaje);

            if(stmtActualizaDatos.executeUpdate()!=1){
                throw new EJBException("No se pudo actualizar registro"+
                                       " en Perfil Personaje ");
            }

        PreparedStatement borraConocimiento = conexion.prepareStatement("DELETE"+
                                        "  FROM ConocimientoMagico "+
                                        " WHERE fk_Per_Id =?");
            borraConocimiento.setInt(1,this._iIDPersonaje);
            borraConocimiento.executeUpdate();
            borraConocimiento.close();
            Iterator itColMagias =

_personaje.obtenConMagico().obtenMagias().iterator();
            while(itColMagias.hasNext()){
                Magia magia = (Magia)itColMagias.next();
                stmtActualizaDatos = conexion.prepareStatement( "INSERT INTO"+
                                                " \"ConocimientoMagico\" "+
                                                "(fk_Magia_Id,fk_Per_Id)"+
                                                " VALUES (?,?)");
                stmtActualizaDatos.setString(1,magia.obtenNombre());
                stmtActualizaDatos.setInt(2,this._iIDPersonaje);
                if(stmtActualizaDatos.executeUpdate()!=1){
                    throw new SQLException("No se actualizo registro"+
                                           " en ConocimientoMagico ");
                }


            }
            stmtActualizaDatos = conexion.prepareStatement("DELETE"+
                                        " FROM Inventario "+
                                        "WHERE fk_Per_Id =?");
            stmtActualizaDatos.setInt(1,this._iIDPersonaje);
            stmtActualizaDatos.executeUpdate();

            Iterator itColItems = this._personaje.obtenInventario().
                                      obtenItems().iterator();
            while(itColItems.hasNext()){//inicio while
                Item item = (Item) itColItems.next();
                stmtActualizaDatos = conexion.prepareStatement("INSERT INTO"+
                                                " \"Inventario\" "+
                                                "(fk_Ite_Id,fk_Per_Id)"+
                                                " VALUES (?,?)");
                stmtActualizaDatos.setString(1,item.obtenNombre());
                stmtActualizaDatos.setInt(2,this._iIDPersonaje);
                if(stmtActualizaDatos.executeUpdate()!=1){
                    throw new SQLException("No se pudo insertar registro"+
                                           " en Inventario");
                }
            }
```

```
        } catch(Exception e){
            e.printStackTrace();
            throw new EJBException("No se pudo actualizar Personaje
                                   "+e.toString());
        }finally{
            closeConnection(conexion,stmtActualizaDatos);
        }
    }

    public void ejbRemove() throws RemoveException {
        Connection conexion = null;
        PreparedStatement stmtBorrarDatos = null;
        try{
            conexion = _dataSource.getConnection();
            stmtBorrarDatos =conexion.prepareStatement("DELETE  FROM \"Estado"+
                             "Personaje\" WHERE fk_per_Id =?");
            stmtBorrarDatos.setInt(1,this._iIDPersonaje);

            if(stmtBorrarDatos.executeUpdate()!=1){
              throw new RemoveException("No se pudo borrar registro en Estado");
            }
            stmtBorrarDatos = conexion.prepareStatement("DELETE  FROM "+
                             "\"PerfilPersonaje\" WHERE fk_per_Id =?");
            stmtBorrarDatos.setInt(1,this._iIDPersonaje);
            if(stmtBorrarDatos.executeUpdate()!=1){
                throw new RemoveException("No se pudo borrar registro"+
                                          " en Perfil Personaje ");
            }
            if(!this._personaje.obtenConMagico().obtenMagias().isEmpty()){
                stmtBorrarDatos = conexion.prepareStatement("DELETE  FROM"+
                             " \"ConocimientoMagico\" WHERE fk_Per_Id =?");
                stmtBorrarDatos.setInt(1,this._iIDPersonaje);
                if(stmtBorrarDatos.executeUpdate()!=1){
                    throw new RemoveException("No se borro registro"+
                                              " en ConocimientoMagico ");
                }
            }
            if(!this._personaje.obtenInventario().obtenItems().isEmpty()){
                stmtBorrarDatos = conexion.prepareStatement(
                           "DELETE  FROM \"Inventario\" WHERE fk_Per_Id =?");
                stmtBorrarDatos.setInt(1,this._iIDPersonaje);
              if(stmtBorrarDatos.executeUpdate()!=1){
               throw new RemoveException("No se borro registro en Inventario ");
                }
            }
            stmtBorrarDatos = conexion.prepareStatement(
                             "DELETE  FROM \"Personaje\" WHERE "+
                             "fk_Per_Id =?");
            stmtBorrarDatos.setInt(1,this._iIDPersonaje);
            if(stmtBorrarDatos.executeUpdate()!=1){
                throw new RemoveException("No se borro registro en Personaje ");
            }
        }catch(Exception e){
            e.printStackTrace();
            throw new EJBException("No se pudo borrar Personaje "+e.toString());
        }finally{
            closeConnection(conexion,stmtBorrarDatos);
        }
    }

    public void ejbActivate() {
    }

    public void ejbPassivate() {
    }

    public void setEntityContext(EntityContext entityContext) {
        this.entityContext = entityContext;
        try{
            Context contexto = new InitialContext();
            try{
```

```
                _dataSource = (DataSource)contexto.lookup("java:comp/env/jdbc/"+
                                                          "dsTonalli");
        }catch(Exception e){
         throw new EJBException("No se pudo encontrar la base de datos : " + e);
            }
        }catch(Exception e){
            throw new EJBException("No se pudo encontrar el contexto : " + e);
        }
    }

    public void unsetEntityContext() {
        entityContext = null;
    }
```

Parte 3.

- *Métodos Find.* Como se ha descrito anteriormente, estos son los métodos que permiten encontrar la información dentro de la base de datos y son análogos a instrucciones SELECT. Cuando se trabajan con beans de entidad que manejan su persistencia, se debe de implementar toda la lógica necesaria para que puedan acceder a la base de datos y obtener la información que requieren. En este sentido no difieren de los métodos explicados hasta el momento, ya que en primer lugar se debe establecer una conexión con la base de datos, para posteriormente ejecutar la sentencia SELECT deseada.

  En este punto se debe mencionar que estos métodos regresan uno o más objetos del tipo seleccionado como llave primaria. Así, el valor de retorno del método ejbFindByPrimaryKey() es de tipo String. Sin embargo, cuando se deseen regresar múltiples valores, estos deben englobarse dentro de una colección.

```
    public String ejbFindByPrimaryKey(String primKey) throws

ObjectNotFoundException {
        Connection conexion= null;
        PreparedStatement buscaPersonaje = null;
        String strNombre="";
        try{
            conexion = _dataSource.getConnection();
            buscaPersonaje = conexion.prepareStatement("Select "+
                                "per_Nombre FROM "+
                                " \"Personaje\" WHERE per_Nombre = ?");
            buscaPersonaje.setString(1,primKey);
            ResultSet llavePrimaria = buscaPersonaje.executeQuery();
            if(llavePrimaria.next()){
                strNombre = llavePrimaria.getString(1).trim();
            }else{
         throw new ObjectNotFoundException("No se encontro personaje");
            }
            llavePrimaria.close();

        }catch(Exception e){
     throw new EJBException("No se encontro Personaje" + e.toString());
        } finally {
            closeConnection(conexion,buscaPersonaje);
            return strNombre;
        }
    }
```

```
    public Collection ejbFindAll() throws   ObjectNotFoundException {
        Connection conexion= null;
        PreparedStatement buscaPersonaje = null;
        Collection colPersonajes = new ArrayList();
        try{
            conexion = _dataSource.getConnection();
            buscaPersonaje = conexion.prepareStatement("Select "+
                                    "per_Nombre FROM "+
                                     " \"Personaje\" ");
            ResultSet llavePrimaria = buscaPersonaje.executeQuery();
            Personaje personaje = null;
            while(llavePrimaria.next()){
                colPersonajes.add(llavePrimaria.getString(1).trim());
            }

            llavePrimaria.close();
        }catch(Exception e){
            throw new EJBException("No se encontraron Personajes"+
                                   e.toString());
        }finally{
            closeConnection(conexion,buscaPersonaje);
            return colPersonajes;
        }
    }

    public String ejbFindByCuenta(String login)throws

ObjectNotFoundException {
        Connection conexion= null;
        PreparedStatement buscaPersonaje = null;
        String strNombre="";
        try{
            conexion = _dataSource.getConnection();
            buscaPersonaje = conexion.prepareStatement("Select "+
"                                   "per_Nombre FROM "+
                                   "Personaje,Cuenta"+
                                   " WHERE cue_Login = ?"+
                                   " AND Personaje.per_Id = "+
                                   "Cuenta.fk_per_Id");
            buscaPersonaje.setString(1,login);
            ResultSet llavePrimaria = buscaPersonaje.executeQuery();
            if(llavePrimaria.next()){
                strNombre = llavePrimaria.getString(1).trim();
            }else{
         throw new ObjectNotFoundException("No se encontro personaje");
            }
            llavePrimaria.close();

        }catch(Exception e){
     throw new EJBException("No se encontro Personaje" + e.toString());
        }finally{
            closeConnection(conexion,buscaPersonaje);
            return strNombre;
        }
    }
```

Parte 4.

- *Métodos del negocio.* Se implementan todos aquellos métodos que se definieron en la interfaz remota. Generalmente abarca métodos set y get para poder manipular de forma indirecta los datos a los que hace referencia el bean, ya que el cliente nunca entra en contacto directamente con estos.

- *Métodos ejbHome.* En el caso de este ejemplo se tiene un método de este tipo, el método `getTotalPersonajes`, que dentro de la clase del bean se le antepone el prefijo ejbHome, quedando como `ejbHomeGetTotalPersonajes()`. Este método tan sólo regresa una cuenta de todos los personajes que se encuentran registrados en la base de datos y por lo tanto no requiere estar asociado a datos en particular.

- *Métodos adicionales.* Como cualquier otra clase, un bean, no sólo de entidad, puede contar con métodos propios que no están expuestos al cliente y que le ayudan a realizar sus funciones de forma más ordenada. En este caso, se cuenta con el método `closeConnection()` que permite dar mantenimiento tanto a las conexiones (`Connection`) como a los objetos de tipo `Statement` que se utilizan a lo largo de todo el bean para acceder a la base de datos.

```
/*Metodos set()*/
public void setPuntosSalud(int intPuntosSalud){
    this._personaje.obtenEstado()._iPuntosSalud = intPuntosSalud;
}

public void setPuntosMagia(int intPuntosMagia){
    this._personaje.obtenEstado()._iPuntosMagia = intPuntosMagia;
}

public void setAgilidad(int intAgilidad){
this._personaje.obtenEstado()._iAgilidad= intAgilidad;
}

public void setExperiencia(int intExperiencia){
    this._personaje.obtenEstado()._iExperiencia = intExperiencia;
}

public void setResistencia(int intResistencia){
    this._personaje.obtenEstado()._iResistencia = intResistencia;
}

public void setFuerza(int intFuerza){
    this._personaje.obtenEstado()._iFuerza = intFuerza;
}

public void setSuerte(int intSuerte){
    this._personaje.obtenEstado()._iSuerte = intSuerte;
}

public void setAgregaItemInventario(Item item){
    this._personaje.obtenInventario().agregaItem(item);
}

public void setAgregaMagiaConMagico(Magia magia){
    this._personaje.obtenConMagico().agregaMagia(magia);
}

public void setEliminaItemInventario(Item item){
    this._personaje.obtenInventario().eliminaItem(item.obtenNombre());
}

public void setEliminaMagiaConMagico(Magia magia){
    this._personaje.obtenConMagico().eliminaMagia(magia.obtenNombre());
}

/*Metodos get()*/
public int getIDPersonaje(){
    return this._iIDPersonaje;
}

public int getPuntosSalud(){
```

```
 return new Integer(this._personaje.obtenEstado()._iPuntosSalud).intValue();
}

public int getPuntosMagia(){
 return new Integer(this._personaje.obtenEstado()._iPuntosMagia).intValue();
}

public int getAgilidad(){
 return new Integer(this._personaje.obtenEstado()._iAgilidad).intValue();
}

public int getExperiencia(){
return  new Integer(this._personaje.obtenEstado()._iExperiencia).intValue();
}

public int getResistencia(){
 return new Integer(this._personaje.obtenEstado()._iResistencia).intValue();
}

public int getFuerza(){
 return new Integer(this._personaje.obtenEstado()._iFuerza).intValue();
}

public int getSuerte(){
 return new Integer(this._personaje.obtenEstado()._iSuerte).intValue();
}

public String getNombre(){
 return new String(this._personaje.obtenNombre());
}

public String getDescripcion(){
 return this._personaje.obtenDescripcion();
}

public String getAvatar(){
 return this._personaje.obtenAvatar();
}

public String getSexo(){
 return this._personaje.obtenPerfil().obtenSexo();
}

public String getColor(){
 return this._personaje.obtenPerfil().obtenColor();
}

public String getRaza(){
 return this._personaje.obtenPerfil().obtenRaza();
}

public String getEdad(){
 return this._personaje.obtenPerfil().obtenEdad();
}

public Inventario getInventario(){
 return this._personaje.obtenInventario();
}

public ConocimientoMagico getConocimientoMagico(){
 return this._personaje.obtenConMagico();
}

public Personaje getPersonaje(){
 return this._personaje;
}

public int ejbHomeGetTotalPersonajes() {
    Connection conexion = null;
    PreparedStatement stmtTotal = null;
    int iTotal = 0;
```

```
        try {
            conexion = _dataSource.getConnection();
            stmtTotal = conexion.prepareStatement("select count(*) as total" +
                                                  " from Personaje");
            ResultSet rsTotal = stmtTotal.executeQuery();
            if (rsTotal.next()) {
                iTotal = rsTotal.getInt("total");
            }
        } catch(Exception e) {
            System.out.println("Excepcion en el metodo " +
                               "ejbHomeGetTotalPersonajes" + e);
        } finally {
            closeConnection(conexion, stmtTotal);
            return iTotal;
        }
}

    void closeConnection(Connection con, Statement stmt) {
        try {
            if (stmt != null) {
                stmt.close();
            }
        } catch(SQLException sqle) {
            System.err.println("Error: No se pudo cerrar el statement");
        }
        try {
            if (con != null) {
                con.close();
            }
        } catch(SQLException sqle) {
            System.err.println("Error: no se pudo cerrar la conexion");
        }
    }
}
```

Interfaces Locales.

En el caso de los beans de entidad se sigue el mismo procedimiento para la definición de interfaces locales. Así, el bean de entidad puede ser accedido por otros beans de sesión que se encuentren dentro del mismo contenedor sin necesidad de que se tengan que hacer referencias remotas.

Interfaz Local. Las diferencias existentes entre la interfaz local y la remota son triviales.

- Debe extenderse la clase `javax.ejb.EJBLocalObject` y
- Las firmas de los métodos del negocio no deben lanzar la excepción `java.rmi.RemoteException`, ya que no se está haciendo uso de la red para comunicarse con el bean.

```
package tonalli.servidor;

import java.rmi.*;
import javax.ejb.*;
import java.util.*;
import tonalli.estructura.ConocimientoMagico;
import tonalli.estructura.elementos.*;
import tonalli.estructura.Inventario;

public interface EntPersonajeLocal extends EJBLocalObject {

    /*Metodos para asociar valores*/
```

```
    public void setPuntosSalud(int intPuntosSalud);
    public void setPuntosMagia(int intPuntosMagia);
    public void setAgilidad(int intAgilidad);
    public void setExperiencia(int intExperiencia);
    public void setResistencia(int intResistencia);
    public void setFuerza(int intFuerza);
    public void setSuerte(int intSuerte);
    public void setAgregaItemInventario(Item item);
    public void setAgregaMagiaConMagico(Magia magia);
    public void setEliminaItemInventario(Item item);
    public void setEliminaMagiaConMagico(Magia magia);

    /*Metodos para obtener valores*/
    public int getIDPersonaje();
    public int getPuntosSalud();
    public int getPuntosMagia();
    public int getAgilidad();
    public int getExperiencia();
    public int getResistencia();
    public int getFuerza();
    public int getSuerte();
    public String getNombre();
    public String getDescripcion();
    public String getAvatar();
    public String getSexo();
    public String getColor();
    public String getRaza();
    public String getEdad();
    public Inventario getInventario();
    public ConocimientoMagico getConocimientoMagico();
    public Personaje getPersonaje();
}
```

Interfaz Home Local. Al igual que con la interfaz remota, las diferencias entre la interfaz Home y la Home local son triviales:
Se debe extender a la clase `javax.ejb.EJBLocalHome`.
Los métodos no lanzan la excepción `java.rmi.RemoteException`.

```
package tonalli.servidor;
import java.rmi.*;
import javax.ejb.*;
import java.util.Collection;
import tonalli.estructura.ConocimientoMagico;
import tonalli.estructura.elementos.*;
import tonalli.estructura.Inventario;

public interface EntPersonajeLocalHome extends javax.ejb.EJBLocalHome {

    /*Metodos create()*/
    public    EntPersonaje    create(Personaje    personaje)    throws
CreateException;
    public EntPersonaje create(String strNombrePersonaje,
                               String strDescripcion,
                               String strAvatar, int intPuntosSalud,
                               int intPuntosMagia, int intAgilidad,
                               int intResistencia, int intExperiencia,
                               int intSuerte, int intFuerza,
                               String strEdad, String strSexo,
                               String strColor, String strRaza)
                              throws CreateException;
```

```
    /*Metodos Finder*/
    public EntPersonaje findByPrimaryKey(String primKey)
          throws ObjectNotFoundException, FinderException;
    public Collection findAll()
          throws ObjectNotFoundException, FinderException;
    public EntPersonaje findByCuenta(String login)
          throws ObjectNotFoundException, FinderException;

    /*Metodos ejbHome: Nota, este metodo no existe dentro de la clase
      EntPersonajeHome, pero se añade para ejemplificar el uso de estos
      métodos.*/
    public int getTotalPersonajes();
}
```

Descriptores.

Hasta este punto ya se cuenta con el código que me permite acceder de forma tanto remota como local a los beans y se ha codificado el comportamiento de los métodos del negocio, pero todavía quedan algunas cuestiones sin resolver, una de ellas, es la de cómo saber que mi clase del bean realmente implementa los métodos definidos en la interfaz remota ya que nunca se implementa de forma directa en el bean.

Es en este punto donde se toma en consideración el descriptor de despliegue (deployment descriptor). Este descriptor es el encargado de describir las características del bean al contenedor.

Estos descriptores son uno de los puntos fuertes de EJB ya que permiten especificar de forma declarativa los atributos del bean en lugar de programarlos directamente dentro del bean. Esto es bastante útil ya que permite modificar estas propiedades sin necesidad de modificar de forma significativa el código.

Físicamente un descriptor de despliegue es un documento XML bien formado, el cual, generalmente es generado automáticamente por las herramientas IDE (Integrated Development Environment o Ambiente Integrado de Desarrollo) como JBuilder, aunque nada impide al desarrollador crear sus propios descriptores o editar los generados por estas herramientas. A continuación se presenta el descriptor de despliegue para el bean que se ha creado.

Este archivo generado debe almacenarse con el nombre `ejb-jar.xml` y debe ser guardado en un directorio llamado `META-INF`.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE ejb-jar PUBLIC "-//Sun Microsystems, Inc.//DTD Enterprise
JavaBeans 2.0//EN" "http://java.sun.com/dtd/ejb-jar_2_0.dtd">
<ejb-jar>
    <enterprise-beans>
        <entity>
            <ejb-name>EntPersonaje</ejb-name>
            <home>tonalli.servidor.EntPersonajeHome</home>
            <remote>tonalli.servidor.EntPersonaje</remote>
            <ejb-class>tonalli.servidor.EntPersonajeBean</ejb-class>
            <persistence-type>Bean</persistence-type>
            <prim-key-class>java.lang.String</prim-key-class>
            <reentrant>False</reentrant>
            <resource-ref>
                <description />
                <res-ref-name>jdbc/dsTonalli</res-ref-name>
                <res-type>javax.sql.DataSource</res-type>
                <res-auth>Container</res-auth>
            </resource-ref>
        </entity>
```

```
    </enterprise-beans>
    <assembly-descriptor>
        <container-transaction>
            <method>
                <ejb-name>EntPersonaje</ejb-name>
                <method-name>*</method-name>
            </method>
            <trans-attribute>Required</trans-attribute>
        </container-transaction>
    </assembly-descriptor>
</ejb-jar>
```

Veamos ahora parte por parte cada una de las partes que componen a este descriptor.

Encabezado.

Cada descriptor debe comenzar indicando la versión de XML que está utilizando:

```
<?xml version="1.0" encoding="UTF-8"?>
```

En este caso se indica que el documento se adhiere a la versión 1.0 y la codificación de caracteres utilizada es la UTF-8 que es la soportada por los contenedores EJB disponibles.

La siguiente etiqueta especifica el DTD que define al documento:

```
<!DOCTYPE ejb-jar PUBLIC "-//Sun Microsystems, Inc.//DTD Enterprise JavaBeans 2.0//EN" "http://java.sun.com/dtd/ejb-jar_2_0.dtd">
```

Esta etiqueta ofrece el URL desde el cual se puede descargar el DTD para la validación de este documento.

Cuerpo.

El cuerpo del descriptor comienza y termina con la etiqueta que determina la raiz del documento, en este caso la raíz corresponde a la etiqueta `<ejb-jar>`, por lo que el contenido del cuerpo debe estar contenido entre las siguientes etiquetas:

```
<ejb-jar>
    demas elementos
</ejb-jar>
```

Dentro de dichas etiquetas se coloca toda la información relativa a los beans, en este caso tenemos:

<enterprise-beans> Es una etiqueta requerida y contiene la descripción de todos los beans que se incluyen dentro de un archivo JAR[56]. Dentro de esta etiqueta se indica para cada bean su tipo con las etiquetas <session>, <entity> y <message-driven>. En este ejemplo solo contamos con un solo bean de sesión, por lo que se tiene lo siguiente:

```
<enterprise-beans>
    <session>
        atributos del bean...
    </session>
</enterprise-beans>
```

Dentro de la etiqueta <session>...</session>, <entity>...</entity> y <message-driven>...</message-driven> se incluye toda la información que describe propiamente a cada bean. En este ejemplo sólo se tomarán en cuenta aquellos atributos aplicables a todos los beans y particularmente a los beans de entidad en el apartado siguiente se profundizarán en todos aquellos atributos relativos a los beans de entidad.

- <ejb-name>...</ejb-name>. Requerido por todos los beans. Especifica el nombre del componente EJB. El nombre utilizado en esta etiqueta para cada uno de los ejemplos es igual al nombre de la interfaz remota.
- <home>...</home>. Requerido por todos los beans. Especifica el nombre completamente calificado de la clase de la interfaz home del bean.
- <remote>...</remote>. Requerido por todos los beans. Especifica el nombre completamente calificado de la clase de la interfaz remota del bean.
- <local-home>...</local-home>. Requerido por todos los beans que implementan interfaz home local. Especifica el nombre completamente calificado de la clase de la interfaz home local del bean.
- <local>...</local>. Requerido por todos los beans que implementan interfaz local. Especifica el nombre completamente calificado de la clase de la interfaz local del bean.
- <ejb-class>...</ejb-class>. Requerido por todos los beans. Especifica el nombre completamente calificado de la clase del bean.
- <persistence-type>...</persistence-type>. Requerido solo por beans de entidad. Se indica si un bean de entidad maneja su propia persistencia (Bean) o la maneja el contenedor (Container).
- <prim-key-class>...</prim-key-class>. Requerido solo por los beans de entidad. Se indica el nombre completamente calificado de la clase utilizada como llave primaria.
- <reentrant>...</reentrant>. Requerido sólo por los beans de entidad. Declara si el bean permite invocaciones reentrantes y puede tener dos valores True o False.

[56] Nada impide que dentro de un solo descriptor se describan las propiedades de varios beans, siempre y cuando se incluya dicha descripción dentro de las etiquetas <enterprise-beans>...</enterprise-beans>

Referencias a recursos externos.

Dentro de nuestro bean de entidad nosotros observamos la etiqueta `<resource-ref>` Esta etiqueta se utiliza cuando queremos que nuestro bean haga referencia a recursos externos, en este caso, este recurso externo al que el bean hace referencia es la conexión a la base de datos. El mecanismo para lograrlo implica el mapeo del recurso deseado a un nombre JNDI ENC. Dentro de esta etiqueta se pueden englobar los siguientes atributos:

- `<description>...</description>`. Descripción del recurso externo al que se hace referencia.

- `<res-ref-name>...</res-ref-name>`. Indica una ruta relativa al contexto: "`java:comp/env`" y en este caso bajo el subdirectorio `jdbc` ya que se hace uso de un de un `DataSource`.

- `<res-type>...</res-type>`. Se utiliza para indicar el nombre completamente calificado del recurso; en este caso al tratarse de un `DataSource` se tiene `javax.sql.DataSource`.

- `<res-auth>...</res-auth>`. Se utilice para indicar al servidor quien es el responsable de la autentificación y puede ser tanto el contenedor (`Container`) o la Aplicación (`Application`). En el primer caso todos los accesos al recurso se realizan de forma automática, mientras que en el último el bean debe implementar dicha lógica.

Hasta este punto hemos descrito todo lo que podemos acerca de nuestro bean propiamente, por lo que todo hemos concluido en describir todo aquello que se coloca dentro de las etiquetas `<enterprise-beans>...</enterprise-beans>`. Ahora debemos especificar la forma en la que los beans descritos se ensamblan dentro de una aplicación, para esto contamos con la etiqueta `<assembly-descriptor>...</assembly-descriptor>`, que dentro de nuestro documento XML se coloca al mismo nivel que `<enterprise-beans>`:

```
<ejb-jar>
    <enterprise-beans>
        descripcion de los beans
    </enterprise-beans>
    <assembly-descriptor>
        descripcion de ensamble de beans en la aplicación
    </assembly-descriptor>
</ejb-jar>
```

Cabe hacer notar que esta etiqueta es de uso opcional **sólo** por aquellos desarrolladores cuya función es la de generar el código de los beans, pero no para quien se encarga de realizar el despliegue de los componentes desarrollados. Ahora bien, esta etiqueta cumple con tres funciones básicas:

1. Describe los atributos transaccionales que se aplican a los métodos del bean.
2. Describe la lógica de seguridad.
3. Especifica los permisos de los métodos.

Las etiquetas que puede contener son:

`<container-transaction>...</container-transaction>`. Este atributo declara que atributos de transaccionales se aplican a que métodos. A su vez contiene uno o más elementos `<method>...</method>` y un elemento `<trans-attribute>...</trans-attribute>`. Los valores que puede tener son: `NotSupported`, `Supports`, `Requiered`, `RequieresNew`, `Mandatory` y `Never`.

```
<assembly-descriptor>
    <container-transaction>
        <method>
            <ejb-name>EntPersonaje</ejb-name>
            <method-name>*</method-name>
        </method>
        <trans-attribute>Required</trans-attribute>
    </container-transaction>
</assembly-descriptor>
```

En este caso dentro de la etiqueta `<method>...</method>` se indica en primera instancia el nombre del bean cuyos métodos se están describiendo, a continuación se desglosan los nombres de los métodos afectados; en este caso se utiliza el comodín * que hace referencia a todos los métodos del bean.

La etiqueta `<trans-attribute>...</trans-attribute>` nos indica con el valor `Requiered` que todos los métodos **requieren** que exista una transacción para ser ejecutados.

Fuentes de Datos (DataSources).

Para que un bean de entidad BMP trabaje de forma adecuada, este debe acceder a la base de datos o el recurso en el que debe persistir la información de sus campos. Generalmente, para lograr este objetivo, el bean utiliza una fábrica de recursos[57] del ambiente del contexto de nombres (JNDI ENC).

En EJB cada bean tiene acceso a un ambiente JNDI ENC debido a que es parte del contrato existente entre el bean y el contenedor. Así, en el descriptor de despliegue se puede establecer toda la información necesaria para acceder a recursos del tipo: JDBC DataSource, JavaMail y Java Message Service.

Cuando el bean es desplegado, toda la información del tag `<resource-ref>` se mapea a la base de datos y aunque esto se implementa de distintas formas de acuerdo al fabricante del contenedor, siempre se llega al mismo resultado.

[57] Del inglés resource factory.

Persistencia Manejada por el Contenedor.

Como puede observarse del ejemplo anterior, la clase del bean es demasiado extensa y el código es muy difícil de mantener. Por esta razón se tiene un segundo enfoque para manejar los beans de entidad, que es la persistencia manejada por el contenedor. En este caso el código de la clase del bean se reduce considerablemente, ya que es el contenedor y no el desarrollador quien se encarga de generar el código que permite los accesos a la base de datos. A continuación se desarrollará la clase del bean `EntItem`, que es el bean encargado de manejar toda la información relacionada con los Ítems existentes dentro del juego, manejando una persistencia manejada por el contenedor.[58] De igual forma se quiere hacer notar que dentro del proyecto no se manejan beans CMP, aquí sólo se utilizan con carácter demostrativo.

Clases e Interfaces.

Interfaz Remota.

```
package tonalli.servidor;

import java.rmi.*;
import javax.ejb.*;

public interface EntItem extends EJBObject {
    public void setDescripcion(String strDescripcion)
                                        throws RemoteException;
    public void setAvatar(String strAvatar)throws RemoteException;
    public void setCosto(int intCosto)throws RemoteException;
    public void setEfectoSalud(int intEfectoSalud)
                                        throws RemoteException;
    public void setEfectoMagia(int intEfectoMagia)
                                        throws RemoteException;
    public void setEfectoResistencia(int intEfectoResistencia)
                                        throws RemoteException;
    public void setEfectoFuerza(int intEfectoFuerza)
                                        throws RemoteException;
    public void setEfectoAgilidad(int intEfectoAgilidad)
                                        throws RemoteException;
    public void setEfectoExperiencia(int intEfectoExperiencia)
                                        throws RemoteException;
    public void setEfectoSuerte(int intEfectoSuerte)
                                        throws RemoteException;
    public String getNombre()throws RemoteException;
    public String getDescripcion()throws RemoteException;
    public String getAvatar()throws RemoteException;
    public int getCosto()throws RemoteException;
    public int getEfectoSalud()throws RemoteException;
    public int getEfectoMagia()throws RemoteException;
    public int getEfectoResistencia()throws RemoteException;
    public int getEfectoFuerza()throws RemoteException;
    public int getEfectoAgilidad()throws RemoteException;
    public int getEfectoExperiencia()throws RemoteException;
    public int getEfectoSuerte()throws RemoteException;
}
```

[58] Para simplicidad del código no se codificarán las interfaces Local y LocalHome.

Interfaz Home.

```
package tonalli.servidor;

import java.rmi.*;
import javax.ejb.*;
import java.util.*;

public interface EntItemHome extends EJBHome {
  public EntItem create(String strNombre, String strDescripcion,
                        String strAvatar, int intCosto, int
                        intEfectoSalud, int intEfectoMagia, int
                        intEfectoResistencia, int intEfectoFuerza,
                        int intEfectoAgilidad, int
                        intEfectoExperiencia, int intEfectoSuerte)
                      throws RemoteException, CreateException;
  public EntItem findByPrimaryKey(String primKey)
    throws ObjectNotFoundException, RemoteException, FinderException;
  public Collection findAll()
    throws ObjectNotFoundException, RemoteException, FinderException;
}
```

Clase del Bean.

```
package tonalli.servidor;

import java.rmi.*;
import javax.ejb.*;
import java.sql.*;
import javax.sql.*;
import javax.naming.*;
import java.util.*;
import tonalli.estructura.elementos.Item;

public abstract class EntItemBean implements EntityBean {
    EntityContext entityContext;

  public EntItem create(String strNombre, String strDescripcion,
                        String strAvatar, int intCosto, int
                        intEfectoSalud, int intEfectoMagia, int
                        intEfectoResistencia, int intEfectoFuerza,
                        int intEfectoAgilidad, int
                        intEfectoExperiencia, int intEfectoSuerte)
                      throws CreateException{
        setNombre(strNombre);
        setDescripcion(strDescripcion);
        setAvatar(strAvatar);
        setCosto(intCosto);
        setEfectoSalud(intEfectoSalud);
        setEfectoMagia(intEfectoMagia);
        setEfectoResistencia(intEfectoResistencia);
        setEfectoFuerza(intEfectoFuerza);
        setEfectoAgilidad(intEfectoAgilidad);
        setEfectoExperiencia(intEfectoExperiencia);
        setEfectoSuerte(intEfectoSuerte);
        return strNombre;
    }

    public void ejbPostCreate(String strNombre, String strDescripcion,
                        String strAvatar, int intCosto, int
                        intEfectoSalud, int intEfectoMagia, int
                        intEfectoResistencia, int intEfectoFuerza,
```

```
                        int intEfectoAgilidad, int
                        intEfectoExperiencia, int intEfectoSuerte)
                    throws CreateException{  }

    public void ejbLoad() {}
    public void ejbStore() {}
    public void ejbRemove() {}
    public void ejbActivate() {}
    public void ejbPassivate() {}
    public void setEntityContext(EntityContext entityContext) {
        this.entityContext = entityContext;
    }
    public void unsetEntityContext() {
        entityContext = null;
    }
    public abstract void setNombre(String strNombre);
    public abstract void setDescripcion(String strDescripcion);
    public abstract void setAvatar(String strAvatar);
    public abstract void setCosto(int intCosto);
    public abstract void setEfectoSalud(int intEfectoSalud);
    public abstract void setEfectoMagia(int intEfectoMagia);
    public abstract void setEfectoResistencia(int intEfectoResistencia);
    public abstract void setEfectoFuerza(int intEfectoFuerza);
    public abstract void setEfectoAgilidad(int intEfectoAgilidad);
    public abstract void setEfectoExperiencia(int intEfectoExperiencia);
    public abstract void setEfectoSuerte(int intEfectoSuerte);
    public abstract String getNombre();
    public abstract String getDescripcion();
    public abstract String getAvatar();
    public abstract int getCosto();
    public abstract int getEfectoSalud();
    public abstract int getEfectoMagia();
    public abstract int getEfectoResistencia();
    public abstract int getEfectoFuerza();
    public abstract int getEfectoAgilidad();
    public abstract int getEfectoExperiencia();
    public abstract int getEfectoSuerte();
}
```

Como puede observarse el código necesario para codificar un bean cuya persistencia es manejada por el contenedor es mucho menor que aquel cuya persistencia es manejada por el mismo bean. Esto es debido a que bajo este esquema, quien se encarga de realizar todos los accesos a la base de datos es el contenedor y no el bean.

Para comprender mejor este código existen ciertos puntos que deben quedar claros sobre el manejo de los CMP.

- La clase se declara como abstracta. Es el contenedor el encargado de generar el código que realiza las implementaciones concretas, tanto los métodos de acceso a la base de datos, así como las variables persistentes y aquellos métodos get y set; todo esto extendiendo esta clase y con base en la información proporcionada en el descriptor de despliegue.
- No se declaran directamente las variables en las que se mapean los valores de la base de datos. Esto es debido a que se sigue un modelo de programación abstracta, en la que se manejan campos de persistencia virtuales, es decir, el desarrollador de beans no debe declararlos de forma explícita, sino que por el contrario se declaran métodos abstractos get y set.
- La forma en la que se indican las variables en las que se deben almacenar la información de la base de datos se declaran en el descriptor de despliegue.

Descriptor.

El descriptor de un bean de entidad CMP no difiere mucho, en su parte fundamental, de un bean de entidad BMP, veamos ahora el descriptor para el bean que se acaba de crear.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE ejb-jar PUBLIC "-//Sun Microsystems, Inc.//DTD Enterprise
JavaBeans 2.0//EN" "http://java.sun.com/dtd/ejb-jar_2_0.dtd">
<ejb-jar>
    <enterprise-beans>
        <entity>
            <ejb-name>EntItem</ejb-name>
            <home>tonalli.servidor.EntItemHome</home>
            <remote>tonalli.servidor.EntItem</remote>
            <ejb-class>tonalli.servidor.EntItemBean</ejb-class>
            <persistence-type>Container</persistence-type>
            <prim-key-class>java.lang.String</prim-key-class>
            <reentrant>False</reentrant>
            <cmp-version>2.x</cmp-version>
            <abstract-schema-name>EntItem</abstract-schema-name>

            <cmp-field>
                <field-name>nombre</field-name>
            </cmp-field>
            <cmp-field>
                <field-name>descripcion</field-name>
            </cmp-field>
            <cmp-field>
                <field-name>avatar</field-name>
            </cmp-field>
            <cmp-field>
                <field-name>costo</field-name>
            </cmp-field>
            <cmp-field>
                <field-name>efectoSalud</field-name>
            </cmp-field>
            <cmp-field>
                <field-name>efectoMagiaNombre</field-name>
            </cmp-field>
            <cmp-field>
                <field-name>efectoResistencia</field-name>
            </cmp-field>
            <cmp-field>
                <field-name>efectoAgilidad</field-name>
            </cmp-field>
            <cmp-field>
                <field-name>efectoExperiencia</field-name>
            </cmp-field>
            <cmp-field>
                <field-name>efectoSuerte</field-name>
            </cmp-field>

            <query>
                <query-method>
                    <method-name>findAll</method-name>
                    </method-params>
                </query-method>
                <ejb-ql>
                    SELECT Object(i) FROM EntItem AS i
                </ejb-ql>
            </query>
```

```
        </entity>
    </enterprise-beans>
    <assembly-descriptor>
        <container-transaction>
            <method>
                <ejb-name>EntItem</ejb-name>
                <method-name>*</method-name>
            </method>
            <trans-attribute>Required</trans-attribute>
        </container-transaction>
    </assembly-descriptor>
</ejb-jar>
```

- La etiqueta <persistence-type> se cuenta con el valor Container en lugar de Bean.
- No se cuenta con una etiqueta <resource-ref> que nos indique la referencia a la base de datos.
- Mediante la etiqueta <cmp-version> indicamos la versión de CMP que se está utilizando, ya que la versión 2.0 no es compatible con la versión 1.1; sin embargo, los contenedores deben soportar ambas.
- Cada bean de entidad que será referenciado en un método EJBQL debe tener un atributo especial conocido como <abstract-schema>...</abstract-schema>. El valor de esta etiqueta debe ser único para cada bean, ya que dentro de las sentencias EJB-QL se hace referencia a este nombre.
- Los campos persistentes del bean se declaran dentro de las etiquetas:

```
<cmp-field>
    <field-name>nombre_campo</field-name>
</cmp-field>
```

Sentencias EJB-QL.

EJB-QL[59] es un lenguaje de consultas declarativo similar a SQL, pero está orientado para trabajar con el esquema de persistencia abstracta de los beans de entidad, y no directamente sobre la base de datos. Cuando la clase abstracta del bean es desplegada por el contenedor, las sentencias EJB-QL son examinados y traducidos en el código de acceso a datos correspondiente.

EJB-QL hace posible que los desarrolladores describan el comportamiento de los métodos de consulta de una forma abstracta. Cada una de las consultas EJB-QL se declaran dentro de las etiquetas <query> del descriptor de despliegue. Esta etiqueta cuenta con dos elementos importantes: <query-method> Identifica el método de las interfaces remota y/o local y <ejb-ql> que es donde propiamente se declara la sentencia EJB-QL.

Dentro de la etiqueta <query-method> es donde se define tanto el nombre del método al que aplica la sentencia EJB-QL mediante la etiqueta <method-name>...</meted-name>, así como los parámetros que debe recibir este método para poder trabajar, esto con las etiquetas:

[59] Acrónimo de Enterprise Java Beans Query Languaje

```
<query-method>
    <method-name></method-name>
    <method-params>
        <method-param></method-param>
    </method-params>
</query-method>
```

Con toda esta información adicional suministrada, el contenedor ya tiene el conocimiento para poder generar de forma automática todos aquellos métodos que en el caso del bean de entidad BMP tenían que ser codificados por el desarrollador. Por esta razón el código de la clase del bean bajo un esquema CMP es mucho menor al de su contraparte BMP, sin embargo, el descriptor de despliegue del primero es más complejo que el segundo.

Ciclo de vida de los Beans de Entidad.

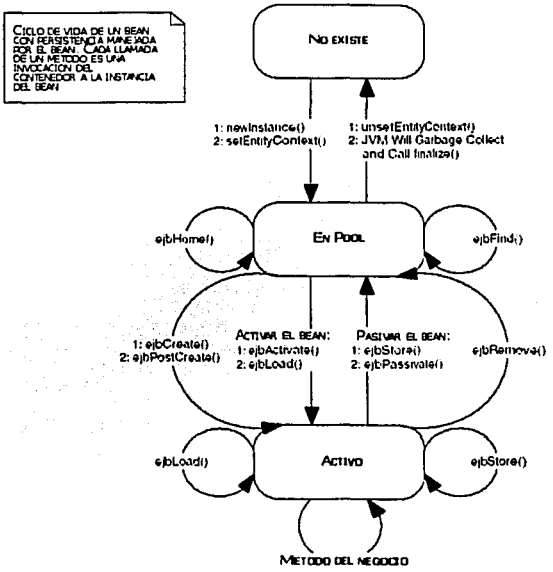


Ilustración 51 : Diagrama del ciclo de vida de un bean de entidad.

- **No Existe.** No existe instancia alguna del bean de entidad en el contenedor.
- **En Pool.** Al iniciar el servidor, el contenedor instancia múltiples instancias de las clases del bean y las coloca en un pool. Estas instancias son creadas con el método `Class.newInstance()`. Posteriormente se le asigna un contexto para que el bean pueda conocer su entorno invocando el método `setEntityContext()` de la interfaz `EntityBean`. Ahora el bean está listo para poder atender las solicitudes de los clientes. Todas las instancias en el pool son equivalentes, ya que no están asociadas a datos en particular. Los métodos que pueden ser llamados del bean son los métodos finder, ejbHome, create y ejbSelect.
- **Activo.** Este estado indica que el bean ha sido asignado a un objeto EJB, es decir, que ha sido asociado a datos particulares; esto puede deberse a dos situaciones: La inserción de nuevos datos en la base de datos (`create()`) o la activación de una entidad.

Transición al estado Activo.

- Invocación de un método `create()` en la interfaz Home. Una vez invocado este método, se delega la responsabilidad al método `ejbCreate()` correspondiente dentro del bean y finalmente se ejecuta el método `ejbPostCreate()` para finalizar la inicialización de todo aquello que es necesario para el bean.
- Cuando se ejecuta un método finder, se toma una o más instancias de un bean del pool y cambia su estado a Activo. Este tipo de beans siguen las mismas reglas que aquellos beans que han sido pasivados, ya que se activan al momento en el que el cliente ejecuta un método del negocio. En el caso de los beans de entidad CMP, el contenedor es el encargado de manejar la sincronización entre los datos almacenados en el bean y los datos en la base de datos; en los beans de entidad BMP esta sincronización se logra mediante el método `ejbLoad()`.

Transición del estado Activo al Pool.

- Cuando un bean no está ocupado atendiendo las solicitudes del cliente, este se coloca en el pool (pasiva) para ahorrar recursos del servidor. En el caso de los beans de entidad CMP, el contenedor es quien sincroniza los datos del bean con los datos de la base de datos; en el caso de los beans de entidad BMP, esta sincronización se logra mediante el método `ejbStore()`.
- Cuando el cliente invoca alguno de los métodos `remove()`. Una vez finalizada la eliminación de los datos de la base de datos, la instancia del bean regresa al pool. En el caso de los beans de entidad CMP, el contenedor no sólo se encarga de eliminar los datos de la base de datos, sino que además elimina las relaciones que el bean pudiera tener; en el caso de los beans de entidad BMP, esta eliminación debe codificarse en el método `ejbRemove()`.

Beans de entidad Desarrollados

Por medio de los beans de entidad, la capa de acceso a datos pone a disposición los servicios para realizar consultas inserciones y actualizaciones a los registros almacenados en la base de datos, completando así la funcionalidad referente al manejo de información persistente.

Los beans de entidad son accesados por la capa de negocio para abastecerse de información que permita construir el juego, a la vez que hace posibles las modificaciones de aquellos datos en donde los cambios trascienden al desarrollo de un juego en particular, alterando el estado de algún elemento.

Cada bean de entidad incluye, como parte de sus servicios:

- 1.- Métodos de creación de registros, para insertar en la base de datos.
- 2.- Métodos de búsqueda de registros.
- 3.- Métodos para obtener el valor de los atributos del bean.
- 4.- Métodos para establecer valores para los atributos del bean.
- 5.- Otros métodos destinados al manejo de información en casos particulares.

La capa de negocio posee la capacidad para traducir las solicitudes que recibe en comandos y consultas a nivel del manejador de la base de datos, completando así, el proceso de gestión de la información.

Un estudio detallado de la organización de la base de datos y de los requerimientos de información del sistema, permite decidir la forma en que los datos serán representados por los beans de entidad.

Nombre	***Responsabilidad***	***Paquete***	***Discusión***
EntItem	Administrar los registros en la Tabla Item de la Base de Datos.	*Tonalli.servidor.*	*Administra los datos de un item, tales como nombre, descripción y efecto en la Base de datos.*
EntMagia	Administrar los registros de la Tabla Magia de la Base de Datos.	*Tonalli.servidor*	*Administra los datos de una magia, tales como nombre, descripción y efecto en la Base de datos*
EntCuenta	Administrar los registros de la Tabla Magia de la Base de Datos.	*Tonalli.servidor*	*Administra los datos de una cuenta, tales como login, password, correo electrónico y personaje asociado en la Base de datos*
EntCasilla	Administrar los registros de la Tabla Cuenta de la Base de Datos.	*Tonalli.servidor*	*Administra los datos de una casilla, tales como sus coordenadas y el color asociado en la Base de datos*
EntPersonaje	Administrar los registros en la Base de Datos asociados a un Personaje.		*Administra los datos del personaje como nombre, edad, sexo, raza, estado, inventario y conocimiento mágico por medio de las tablas Personaje,EstadoPersonaje,PerfilPersonaje,Inventario y ConocimientoMágico.*

Bean de entidad: EntCuenta.

Representa la información del cliente y de la cuenta que utiliza para acceder a los servicios del sistema. La capa de negocio solicita los servicios de este bean cuando requiere información sobre un usuario y para efectos de autentificación, mediante los datos login y password de esta entidad. El dato más significativo es el nombre del usuario o login, por lo que es tomado como llave primaria del bean. Mapea directamente la información de la tabla cuenta en la base de datos.

Tabla: Listado de clases del EntCuenta

Nombre	*Responsabilidad*	*Paquete*	*Discusión*
EntCuenta		Tonalli.servidor.	
EntCuentaHome		Tonalli.servidor	
EntCuentaBean		Tonalli.servidor	

Tabla:Servicios de la Interfaz Home de EntCuenta

Nombre	*Parámetros*	*Tipo Salida*	*Excepciones*	*Objetivo*
ejbCreate	String strLogin String strPassword String strNombre String strCorreo String strSexo, String strNacionalidad int intFk_per_id	String/EntCuenta	SQLException Javax.ejb.Create Exception.	Crear un registro en la tabla Cuenta..
ejbRemove				Borra un registro en la tabla ;.
ejbFindByPrimaryKey	String nombre	String/EntCuenta		Encuentra un registro en la Base de Datos tomando en cuenta el nombre de la Cuenta.

Tabla:Servicios de la Interfaz Remote de EntCuenta

Nombre	*Parametros*	*Tipo Salida*	*Objetivo*
getNombre		String	Obtiene el nombre del usuario
getPassword		String	Obtiene el password de la cuenta
getCorreo		String	Obtiene el correo electrónico del usuario.
getSexo		String	Obtiene el sexo del usuario.
getNacionalidad		String	Obtiene la nacionalidad. del usuario.

Bean de entidad: EntPersonaje.

Representa la información de un personaje. La capa de negocio solicita los servicios de este bean cuando ingresa un nuevo personaje a la partida y posteriormente, cada vez que ocurren eventos que afectan el estado del personaje de forma que deban ser persistidas las modificaciones. El atributo más significativo del bean es el nombre del personaje, por lo que es tomado como llave primaria de la entidad. Mapea a las tablas Personaje, PerfilPersonaje, EstadoPersonaje, Inventario y ConocimientoMagico de la base de datos.

Tabla: Listado de clases del Bean EntPersonaje.

Nombre	*Responsabilidad*	*Paquete*	*Discusión*
EntPersonaje		Tonalli.servidor.	
EntPersonajeHome		Tonalli.servidor	
EntPersonajeBean		Tonalli.servidor	

Tabla:Servicios de la Interfaz Home de EntPersonaje

Nombre	*Parametros*	*Tipo Salida*	*Excepciones*	*Objetivo*
ejbCreate	String strNombrePersonaje, String strDescripcion String strAvatar, int intPuntosSalud, int intPuntosMagia, int intAgilidad, int intResistencia, int intExperiencia, int intSuerte, int intFuerza String strEdad String strSexo String strColorString strRaza	EntPersonaje	SQLException Javax.ejb.CreateException.	Crear un registro de un Personaje. Inserta registros en las tablas Personaje,EstdoPersonaje,PerfilPersonaje.
ejbRemove				Borra un registro en la tabla ;.
ejbFindByPrimaryKey	String primKeyprimKey	EntPersonaje		Encuentra un registro en la Base de Datos tomando en cuenta el nombre del Personaje.
ejbFindAll		Collection		Regresa una colección con todos los registros.

Tabla:Servicios de la Interfaz Remote de EntCasilla

Nombre	*Tipo Salida*	*Objetivo*
getDescripcion	String	Obtiene la descripción del Personaje.
getAvatar	String	Obtiene la imagen asociada al personaje.
getPuntosSalud	int	Obtiene los puntos salud del estado del personaje.
getPuntosMagia,	int	Obtiene los puntos magia del estado del personaje
getAgilidad,		Obtiene la agilidad del estado del personaje
getResistencia,	Int	Obtiene la resistencia del estado del personaje
getExperiencia,	Int	Obtiene la experiencia del estado del personaje
getSuerte	Int	Obtiene la suerte del estado del personaje
getFuerza	Int	Obtiene la fuerza del estado del personaje
getEdad	Int	Obtiene la edad del perfil personaje.
getSexo	Int	Obtiene el sexo del perfil personaje.
getColor,	Int	Obtiene el colordel perfil personaje.
getRaza	Int	Obtiene raza del perfil personaje.
getPersonaje	Int	Obtiene el personaje junto con su inventario y su conocimiento mágico.
getIDPersonaje	Int	Obtiene el identificador del personaje.
getInventario	Int	Obtiene el inventario del personaje.
getConocimientoMagico	Int	Obtiene el conocimiento mágico del Personaje.
setPuntosSalud		Establece los puntos salud del estado del personaje.
setPuntosMagia		Establece los puntos magia del estado del personaje
setAgilidad		Establece la agilidad del estado del personaje
setResistencia,		Establece la resistencia del estado del personaje
setExperiencia		Establece la experiencia del estado del personaje
setSuerte		Establece la suerte del estado del personaje
setFuerza		Establece la fuerza del estado del personaje
setAgregaItemInventario		Agrega un item al inventario del Personaje.
setAgregaMagiaConocimentoMagico		Agrega una magia al conocimiento mágico del personaje.
setEliminaMagiaConocimientoMagico		Elimina un item al inventario del Personaje.
setEliminaItemInventario		Elimina una magia al conocimiento mágico del personaje.

Bean de entidad: EntCasilla.

Representa la información de una casilla del juego. Los servicios de este bean son solicitados para la creación del juego, proceso en el que se recogen todas las casillas existentes y se organizan en el tablero que integra la imagen del juego en el servidor. Gracias a que una casilla es una coordenada del tablero, los atributos más significativos son precisamente los que definen las coordenadas "x" y "y" de la casilla, por lo que se crea una clase PrimaryKey que contiene esos dos valores. Mapea directamente la informació contenida en la tabla casilla de la base de datos.

Tabla: Listado de clases del Bean EntCasilla

Nombre	Responsabilidad	Paquete	Discusión
EntCasilla		Tonalli.servidor.	
EntCasillaHome		Tonalli.servidor	
EntCasillaBean		Tonalli.servidor	
EntCasillaPK	Clase que funge como Llave primaria del EntCasilla.	Tonalli.servidor	Se creo esta clase debido a la llave compuesta que tiene la Tabla Casilla.

Tabla:Servicios de la Interfaz Home de EntCasilla

Nombre	Parametros	Tipo Salida	Excepciones	Objetivo
ejbCreate	int intCoordX int intCoordY String strColor, String strNombre String strDescripcion	EntItem	SQLException Javax.ejb.CreateException.	Crear un registro en la tabla Casilla.
ejbRemove				Borra un registro en la tabla ;.
ejbFindByPrimaryKey	CasillaPK primKey	EntItem		Encuentra un registro en la Base de Datos tomando en cuenta las coordenadas de la Casilla.
ejbFindAll		Colllection		Regresa una colección con todos los registros.

Tabla:Servicios de la Interfaz Remote de EntCasilla

Nombre	*Parametros*	*Tipo Salida*	*Objetivo*	*Discusion*
getNombre		String	Obtiene el nombre de la Casilla.	
getDescripcion		String	Obtiene la descripción de la Casilla	
getColor		String	Obtiene el color asociado a la casilla.	
setNombre	String cas_Nombre		Define un nombre de la Casilla.	Se actualiza el nuevo valor en la Base de Datos.
setDescripcion	String cas_Descripcion		Define la descripción de la Casilla	Se actualiza el nuevo valor en la Base de Datos.
setColor	String cas_Color		Establece el color asociado a la Casilla.	Se actualiza el nuevo valor en la Base de Datos.

Tabla:Servicios de la Clase: CasillaPK

Nombre	*Parametros*	*Tipo Salida*	*Objetivo*	*Discusion*
CasillaPK	int intCoordenadaX, int intCoordenadaY	CasillaPK	Método Constructor..	Inicializa las coordenadas de la Casilla.
equals	Object obj)	boolean	Iguala las coordenadas de la Casilla con un objeto.	Método sobrescrito de la Clase Object.

Bean de entidad: EntMagia.

Representa la información de una magia del juego. Los servicios de este bean son solicitados en la creación del juego para integrar el catálogo de magias del juego, proceso en el que se recogen todos los registros de magias de la base de datos y se agregan al conocimiento mágico del juego. Su dato más significativo es el nombre de la magia mismo que es usado como llave primaria de la entidad. Mapea directamente la información contenida en la tabla magia de la base de datos.

Tabla: Listado de clases del Bean EntMagia.

Nombre	*Responsabilidad*	*Paquete*	*Discusión*
EntMagia		Tonalli.servidor.	
EntMagiaHome		Tonalli.servidor	
EntMagiaBean		Tonalli.servidor	

Tabla:Servicios de la Interfaz Home de EntMagia

Nombre	*Parametros*	*Tipo Salida*	*Excepciones*	*Objetivo*
ejbCreate	String strNombre, String strDescripcion, String int intAlcance, int intCosto, int intEfectoSalud int intEfectoMagia, int intEfectoResistencia, int intEfectoFuerza, int, intEfectoAgilidad, int intEfecto Experiencia, int intEfectoSuerte	EntMagia	SQLExc eption Javax.ejb .CreateE xception.	Crear un registro en la tabla Magia.
ejbRemove				Borra un registro.
ejbFindByP rimaryKey	String primKey	EntMagia		Encuentra un registro tomando en cuenta el nombre de la Magia
ejbFindAll		Collection		Encuentra todos los registros de magias-

Tabla:Servicios de la Interfaz Remote de EntMagia

Nombre	*Parametros*	*Tipo Salida*	*Objetivo*
getNombre		String	Obtiene el nombre de la Magia.
getDescripcion		String	Obtiene la descripción la Magia..
getAlcance		int	Obtiene el alcance en perímetro del efecto de la Magia..
getCosto		int	Obtiene el costo la Magia.al usarlo.
getEfectoSalud		int	Obtiene el efecto la Magia.sobre los PuntosSalud del Personaje
getEfectoMagia		int	Obtiene el efecto la Magia. sobre los PuntosSalud del Personaje
getEfectoFuerza		int	Obtiene el efecto la Magia. sobre los PuntosSalud del Personaje
getEfectoExperiencia		int	Obtiene el efecto la Magia. sobre los PuntosMagia del Personaje
getEfectoResistencia		int	Obtiene el efecto la Magia.sobre la resistencia del Personaje
getEfectoAgilidad		int	Obtiene el efecto la Magia.sobre la Agilidad del Personaje
getEfectoSuerte		int	Obtiene el efecto de la Magia .sobre la suerte del Personaje
setDescripcion	String strDescripcion		Establece una descripción de la magia.
setAlcance	int Alcance		Establece el rango de alcance del efecto de la magia.
setCosto	int intCosto		Establece el costo de la Magia al usarlo.
setEfectoSalud	intEfectoSalud		Establece el efecto de la Magia sobre los PuntosSalud del Personaje
setEfectoMagia	intEfectoMagia		Establece el efecto de la Magia sobre los PuntosSalud del Personaje
setEfectoFuerza	intEfectoFuerza		Establece el efecto de la Magia sobre los PuntosSalud del Personaje
setEfectoExperiencia	intEfectoExperiencia		Establece el efecto de la Magia sobre los PuntosMagia del Personaje
setEfectoResistencia	int intEfectoResistencia		Establece el efecto de la Magia sobre la resistencia del Personaje
SetEfectoAgilidad	int intEfectoAgilidad		Establece el efecto de la Magia sobre la Agilidad del Personaje
setEfectoSuerte	int intEfectoSuerte		Establece el efecto de la Magia sobre la suerte del Personaje

Bean de entidad: EntItem.

Representa la información de un item del juego. Los servicios de este bean son solicitados en la creación del juego para integrar el catálogo de ítems del juego, proceso en el que se recogen todos los registros de ítems de la base de datos y se agregan al inventario del juego. Su dato más significativo es el nombre del item mismo que es usado como llave primaria de la entidad. Mapea directamente la información contenida en la tabla item de la base de datos.

Tabla: Listado de clases del Bean EntItem.

Nombre	*Responsabilidad*	*Paquete*	*Discusión*
EntItem		Tonalli.servidor.	
EntItemHome		Tonalli.servidor	
EntItemBean		Tonalli.servidor	

Tabla: Servicios de la Interfaz Home de EntItem

Nombre	*Parámetros*	*Tipo Salida*	*Excepciones*	*Objetivo*
ejbCreate	String strNombre, String strDescripcion,, String strAvatar, int intCosto, int intEfectoSalud, int intEfectoMagia, int intEfectoResistencia, int intEfectoFuerza, int intEfectoAgilidad, int intEfectoExperiencia, int intEfectoSuerte	EntItem	SQLException Javax.ejb.CreateException.	Crear un registro en la tabla Item.
ejbRemove				Borra un registro en la tabla Item.
ejbFindByPrimaryKey	String nombre	EntItem		Encuentra un registro en la Base de Datos tomando en cuenta el nombre del Item.
ejbFindAll		Collection		Encuentra todos los registros en la Base de Datos.

Tabla: Servicios de la Interfaz Remote de EntItem

Nombre	*Parametros*	*Tipo Salida*	*Objetivo*
getNombre		String	Obtiene el nombre del item.
getDescripcion		String	Obtiene la descripción del item.
getAvatar		String	Obtiene el nombre de la imagen asociada al item.
getCosto		int	Obtiene el costo del item al usarlo.
getEfectoSalud		int	Obtiene el efecto del item sobre los PuntosSalud del Personaje
getEfectoMagia		int	Obtiene el efecto del item sobre los PuntosSalud del Personaje
getEfectoFuerza		int	Obtiene el efecto del item sobre los PuntosSalud del Personaje
getEfectoExperiencia		int	Obtiene el efecto del item sobre los PuntosMagia del Personaje
getEfectoResistencia		int	Obtiene el efecto del item sobre la resistencia del Personaje
getEfectoAgilidad		int	Obtiene el efecto del item sobre la Agilidad del Personaje
getEfectoSuerte		int	Obtiene el efecto del item sobre la suerte del Personaje
setDescripcion	String strDescripcion		Establece una descripción del Item
setAvatar	String strAvatar		Establece el nombre de la imagen asociada al Item.
setCosto	int intCosto		Establece el costo del Item al usarlo.
setEfectoSalud	intEfectoSalud		Establece el efecto del item sobre los PuntosSalud del Personaje
setEfectoMagia	intEfectoMagia		Establece el efecto del item sobre los PuntosSalud del Personaje
setEfectoFuerza	intEfectoFuerza		Establece el efecto del item sobre los PuntosSalud del Personaje
setEfectoExperiencia	intEfectoExperiencia		Establece el efecto del item sobre los PuntosMagia del Personaje
setEfectoResistencia	int intEfectoResistencia		Establece el efecto del item sobre la resistencia del Personaje
SetEfectoAgilidad	int intEfectoAgilidad		Establece el efecto del item sobre la Agilidad del Personaje
setEfectoSuerte	int intEfectoSuerte		Establece el efecto del item sobre la suerte del Personaje

Transacciones.

Una transacción puede ser definida como un intercambio entre dos partes, en el que se lleva a cabo un monitoreo para asegurar que ésta satisfaga a las partes involucradas. En el caso del software orientado a negocios, una transacción es vista como una unidad de trabajo que accede a uno o más recursos compartidos, generalmente bases de datos. Esta unidad de trabajo es un conjunto de actividades que se interconectan entre sí y que deben ser completadas de forma conjunta.

Debido a que las transacciones se utilizan en un sinfín de sistemas de cómputo, muchos de los cuales tienen una tolerancia casi nula a los fallos, se debe vigilar que toda transacción sea segura. Se dice que una transacción es segura cuando cumple con las propiedades ACID[60]:

- **Atomica**. Para ser atómica, una transacción se dene ejecutar completamente o no se ejecuta, es decir, que todas aquellas actividades que componene una unidad de trabajo deben ejecutarse sin errores; en caso de que alguna de estas tareas fallen, toda la transacción se aborta y aquellos cambios realizados deben deshacerse.
- **Consistente**. Se refiere a que debe garantizarse la integridad del almacén de datos. Esto debe ser logrado tanto por el sistema transaccional (garantiza que la transacción se atómica, aislada y durable), así como por el desarrollador, que en la base de datos no viola ninguna regla de integridad, además de que se validan los datos para evitar que éstos no representen de forma adecuada el mundo real.
- **Aislada**. A toda transacción se le debe permitir trabajar sin la interrupción de algún otro método o transacción, es decir, que los datos utilzados por una transacción no pueden ser afectados por otro componente del sistema hasta que la transacción no haya finalizado.
- **Durable**. Se refiere a que todos los cambios realizados en los datos que manipula una transacción, debe de almacenarse en algún dispositivo antes de que termine la transacción, ya que desta manera se garantiza que esta podrá continuarse si el sistema tiene algún error fatal.

Modelos transaccionales.

Existen varios modelos por medio de los cuales se pueden llevar a cabo las transacciones, siendo los más populares el de Transacciones Planas y Transacciones Anidadas. A continuación se da una descripción de cada uno de estos modelos.

- Transacciones Planas. Este es el modelo transaccional más simple. Bajo este modelo varias operaciones son las que conforman una sola unidad de trabajo. Al final de cada transacción se tiene un resultado que puede tener dos valores: éxito o fracaso. Cuando la transacción es exitosa, entonces esta se almacena[61], de lo contrario se aborta.

[60] Acrónimo de Atomic (Atomica), Consistent (Consitente), Isolated(Aislada) y Durable.
[61] Todas aquellas operaciones que implican persistencia se almacenan en la base de datos.

- Transacciones Anidadas. Este modelo permite que se puedan tener unidades de trabajo dentro de otras unidades de trabajo; de esta forma se pueden abortar estas sub-unidades sin necesidad de abortar toda la transacción, esta es una de las características más importantes de este modelo.

Administración declarativa de las transacciones.

Una de las principales ventajas de los EJB es que permiten una administración declarativa de la forma en la que deben de manejarse las transacciones mediante propiedades establecidas en el descriptor de despliegue. De igual forma se deja abierta la posibilidad a que el desarrollador sea quien se encarge de programar todo lo relacionado con éstas mediante el uso del API JTS.[62]

Una de las ventajas del esquema declarativo, es que la forma en la que se manejan las transacciones puede realizarse sin necesidad de afectar la lógica del negocio, adicionalmente, un bean desplegado en una aplicación puede tener un comportamiento transaccional distinto al mismo bean pero en otra aplicación, todo lo anterior sin modificar una sola línea de código.

Atributos de las transacciones.

Gracias a la administración declarativa de las transacciones, los desarrolladores raramente deben preocuparse sobre la forma en la que éstas se llevan a cabo, sino que por el contrario, delegan esta tarea al servidor. En el descriptor de despliegue, en la etiqueta `<trans-attribute>` se pueden tener los siguientes valores:

- `NotSuported`. Este valor nos indica que el bean no puede estar involucrado dentro de una transacción. Este valor sólo debe ser utilizado si se tiene la seguridad de que las operaciones del bean no deben cumplir con las propiedades ACID, es decir, cuando el bean no lleva a cabo operaciones críticas.
- `Supports`. Cuando un bean tiene esta propiead, el bean sólo corre en una transacción si el cliente tiene una, a la cual se une, de lo contrario corre sin transacción. En esencia es similar a Required, con la excepción de que aquí no se crea una nueva transacción en caso de que no exista una previamente.
- `Required`. Se utiliza cuando se desea que el bean siempre se ejecute dentro de una transacción. Si existe una transacción ejecutándose, entonces el bean se une a esta, de lo contrario, el contenedor inicia una nueva.
- `RequiresNew`. Se utiliza cuando se desea que se inicie una nueva transacción cuando se manda ejecutar algún método del bean. Si una transacción se está ejecutando al momento de mandar llamar al bean, esta se suspende y se crea una nueva transacción. La transacción suspendida se activa al finalizar la nueva transacción creada.
- `Mandatory`. Este valor demanda que exista una transacción cuando se invoca algún método del bean. En caso de que no exista transacción alguna se lanza la excepción `javax.ejb.Transaction-RequiredLocalException`.

[62] Java Transaction Service o Servicio de Transacciones Java.

- `Never`. Este valor indica que el bean no puede de ninguna forma estar involucrado dentro de una transacción. En caso de que se invoque al bean dentro de una transacción, el contenedor envía al cliente una excepción `java.rmi.RemoteException` (en caso de interfaces remotas) o `javax.ejb.EJBException` (en caso de interfaces locales).

En algunos de estos atributos se dice que una transacción se *suspende*, esto significa que la transacción no se propaga al método invocado.

Ahora que se conocen los atributos que pueden tener los beans, ¿cómo se que atributo es el adecuado para un bean? A continuación se muestra una tabla en la que se indican los tipos de transacciones que puede manejar cada bean.

Atributo de Transacción	Bean de sesión sin estado.	Bean de sesión con estado.	Bean de entidad.
`Required`	Si	Si	Si
`RequiresNew`	Si	Si	Si
`Mandatory`	Si	Si	Si
`Supports`	Si	No	No
`NotSupported`	Si	No	No
`Never`	Si	No	No

Aislamiento y bloqueo de la base de datos.

El aislamiento de una transacción se define en términos de las condiciones de aislamiento llamadas *lecturas sucias (dirty reads)*, *lecturas irrepetibles (unrepeatable reads)* y *lecturas fantasma (phantom reads)*. Estas condiciones describen lo que puede pasar cuando dos o más transacciones operan sobre los mismos datos.

Lecturas sucias. Ocurren cuando una aplicación lee datos de una base de datos cuya información no se encuentra actualizada, es decir, no se ha hecho un commit. Esto se explica con el siguiente ejemplo:

1. Se lee la variable X de la base de datos. Su valor es de 0.
2. Se actualiza el valor de la variable a 10 y se almacena en la base de datos.
3. Otra aplicación lee la variable X, cuyo valor actual es de 10.
4. Se aborta la transacción inicial y por lo tanto el valor de X es 0.
5. La segunda aplicación añade 10 a la variable X, pero como el valor que leyó anteriormente era 10, ahora X = 20.

Lecturas irrepetibles. Es quella en la que no se garantiza que si durante una transacción se leen varias veces los mismos datos, estos siempre serán los mismos. Esto puede evitarse de dos dos formas: Una es bloqueando los cambios que puedan realizarse a la información que se lee o creando una imagen de estos datos que no refleja los cambios realizados.

Lecturas Fantasma. Un dato fantasma es un nuevo conjunto de datos que aparecen de forma mágica entre dos operaciones de lectura. Por ejemplo:

1. La aplicación hace una consulta a la base de datos con un criterio específico.
2. Otra aplicación inserta un registro en la base de datos que cumple con los requisitos de nuestro criterio de búsqueda.
3. La aplicación realiza nuevamente la misma consulta y un nuevo registro aparece de forma mágica.

Para evitar estos problemas, las bases de datos y especialmente las que se apegan al enfoque relacional implemtan mecanismos de bloqueo de datos, siendo los más comunes:

- Bloqueos de Lectura. Previene que otras transacciones modifiquen los datos utilizados por otra transacción evitando así el problema de lecturas irrepetibles. Las demás transacciones sólo podrán leer estos datos, mas no modificarlos.
- Bloqueos de Escritura. Son utilizados para actualizaciones. Previene que se cambien los datos hasta que se finalice la transacción, pero no previene el problema de lecturas sucias ya que la misma transacción puede leer los mismos datos que no han sido confirmados (uncommited).
- Bloqueos de Escritura Exclusiva. Son similares a los anteriores y se utilizan para las actualizaciones, sólo que a diferencia de el bloqueo de escritura, este sí soluciona el problema de las lecturas sucias
- Imágenes[63]. Son vistas estáticas de los datos a los que se hace referencia cuando se inicia la transacción y estos datos no se pueden modificar. Este enfoque puede prevenir los tres problemas mencionados.

Niveles de aislamiento de transacciones.

Estos son los niveles que se aplican en la base de datos para describir la forma en que funciona el mecanismo de bloqueo a los datos utilizados dentro de una transacción.

- *Lectura Sin Confirmación (Read Uncommited).* Se pueden leer aquellos datos sobre los que no se tiene confirmación de que han sido almacenados por la transacción, por lo que se presentan los problemas de lecturas sucias, lecturas irrepetibles y lecturas fantasma.
- *Lectura Confirmada (Read Commited).* La transacción sólo puede leer aquellos datos sobre los que se tiene la confirmación de que han sido almacenados por la misma. Se previene las lecturas sucias, pero se siguen presentando el problema de las lecturas irrepetibles y las lecturas fantasma.
- *Lectura Repetible (Repeatable Read).* La transacción no puede cambiar los datos que sonreídos por otra transacción. Se previenen las lecturas sucias e irrepetibles, pero se pueden presentar lecturas fantasma.
- *Serializable.* La transacción tiene privilegios exclusivos sobre los datos ya que ninguna otra transacción puede operar de forma alguna sobre los datos que utiliza. En este caso se previenen las lecturas sucias, lecturas irrepetibles y lecturas fantasma.

[63] Monson-Haefel Richard, Enterprise JavaBeans 3rd Edition, Ed. O'Reilly.

Seguridad.

Los servidores de EJB pueden soportar hasta tres tipos de seguridad:

- *Autenticación.* Se encarga de validar la entrada de los usuarios. La forma más simple de autenticación es una pantalla en la que se le pide al usuario que teclee su login y contraseña. Una vez que han pasado esta parte, los usuarios son libres de utilizar el sistema. A pesar de que este tipo de seguridad no se especifica en EJB, generalmente se logra mediante el API JNDI, ya que este API puede ofrecer información de autentificación al aceder a los recursos del servidor.
- *Control de Acceso.* También se conoce como autenticación e implica la aplicación de políticas de seguridad que permiten regular lo que un usuario puede o no hacer dentro del sistema, asegurnado que cada usuario sólo accese a aquellos recursos para los que tiene permisos. Ete es el único de estos tipos de seguridad que atañe directamente a los EJB.
- *Comunicación Segura.* Una de las preocupaciones principales en cuanto a cuestiones de seguridad es la que atañe al canal de comunicación entre el cliente y el servidor. Este canal puede asegurarse ya sea fisicamente (mediante una conexión dedicada) o por medio de encriptación de la comunicación entre las partes involucradas. Muchos de los servidores EJB soportan este tipo de seguridad, generalmente mediante el protocolo SSL (Capa de Socket Segura), pero esta especificación sólo se hace para los componentes que existen del lado del servidor.

EJB especifica que cada aplicación cliente que accesa a un sistema EJB debe estar asociada a una identidad de seguridad. Esta identidad representa alcliente ua sea como un usuario o como un rol. Un usuario puede ser una persona, una computadora, una tarjeta inteligente, etc. Así, un usuario es una persona cuya identidad es asignada cuando ingresa al sistema.

Por su parte un rol representa una agrupación de identidades que comparten características comunes. Por ejemplo el grupo Administradores, es un conjunto de usuarios que se consideran como los administradores de un sistema, compañía, etc.

Cada vez que un usuario accesa a un sistema EJB, este es asociado con una identidad de seguridad durante la duración de la sesión. Una vez asociada a esta identidad, el cliente puede comenzar a utilizar los beans y el servidor EJB se encarga de llevar un control de cada uno de los clientes y su identidad. Cuando un cliente invoca a un método de un bean, el servidor se encarga de notificar al bean la identidad del cliente que lo invoca.

Control de acceso manejado por roles.

En EJB la identidad de seguridad está representada por un objeto del tipo java.security.Principal, el cual actúa como una representación para los usuarios, grupos, organizaciones, etc dentro de la arquitectura de control de acceso.

Los descriptores de despliegue incluyen etiquetas que permiten declarar cuales roles lógicos tienen permitido acceder a los métodos del bean en tiempo de ejecución. Este tipo de roles se consideran lógicos ya que no reflejan de forma directa a las identidades de seguridad en un ambiente de operación específico, por el contrario, los roles de seguridad son mapeados a grupos de usarios "reales" al momento en el que el bean es desplegado.

A continuación se presenta un ejemplo de un descriptor de despliegue que contiene información acerca de roles de seguridad y como es que estos se asignan a los distintos métodos. Para eso se utilizará el descriptor de despliegue del bean `ActualizaJuegoBean`.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE ejb-jar PUBLIC "-//Sun Microsystems, Inc.//DTD Enterprise
JavaBeans 2.0//EN" "http://java.sun.com/dtd/ejb-jar_2_0.dtd">
<ejb-jar>
    <enterprise-beans>
        <session>
            <display-name>ActualizaJuego</display-name>
            <ejb-name>ActualizaJuego</ejb-name>
            <Home>tonalli.servidor.ActualizaJuegoHome</Home>
            <remote>tonalli.servidor.ActualizaJuego</remote>
            <ejb-class>tonalli.servidor.ActualizaJuegoBean</ejb-class>
            <session-type>Stateless</session-type>
            <transaction-type>Container</transaction-type>
        </session>
    </enterprise-beans>
    <assembly-descriptor>
        <container-transaction>
            <method>
                <ejb-name>ActualizaJuego</ejb-name>
                <method-name>*</method-name>
            </method>
            <trans-attribute>Required</trans-attribute>
        </container-transaction>

        <security-role>
            <description>
             Este rol se le permite ejecutar cualquier metodo del bean.
            </description>
            <role-name>Administrador</role-name>
        </security-role>

        <method-permission>
            <role-name>Administrador</role-name>
            <method>
                <ejb-name>ActualizaJuego</ejb-name>
                <method-name>*</method-name>
            </method>
        </method-permission>

    </assembly-descriptor>
</ejb-jar>
```

`<security-role>...</security-role>`. Esta etiqueta se utiliza para definir los roles de seguridad que se utilizan cuando se accesa al bean. Estos roles de seguridad son utilizados en el elemento `<method-permission>`.

```
<security-role>
    <description>
        Este rol se le permite ejecutar cualquier metodo
del bean.
    </description>
    <role-name>Administrador</role-name>
</security-role>
```

`<method-permission>...</method-permission>`. Este elemento especifica cuales roles de seguridad tienen permitido llamar a uno o más métodos del bean. Con la etiqueta `<role-name>` indicamos el rol que se quiere aplicar a uno o más métodos. Con la etiqueta `<ejb-name>` indicamos el bean cuyos métodos se van a restringir a uno o más roles. Con la etiqueta `<method-name>` indicamos los métodos del bean especificado que sólo podrán se accedidos por los miembros del rol definido.

```
<method-permission>
    <role-name>Administrador</role-name>
    <method>
        <ejb-name>ActualizaJuego</ejb-name>
        <method-name>*</method-name>
    </method>
</method-permission>
```

Cabe hacer notar que a pesar de que EJB cuenta con un cierto soporte para la seguridad, este aspecto se encuentra en evolución y por el momento se basa en otras herramientas como el uso de llaves públicas y privadas. Sin embargo se espera que con futuras iteraciones de la plataforma J2EE se perfeccionen muchos de los aspectos de seguridad y EJB pueda soportarlos plenamente.

Capítulo 6.

Marco de Aplicación

CAPÍTULO 6. MARCO DE APLICACIÓN

Para responder a la necesidad que originó la iniciativa presentada en este proyecto, la presentación, revisión incorporación y aplicación de los productos y temas expuestos en el presente trabajo han aportado a la dependencia avances que a continuación se resumen:

Establecimiento, como pauta de organización de trabajo del Área de desarrollo de sistemas de la Dirección General de Televisión Universitaria, de la metodología de desarrollo de aplicaciones multicapa siguiendo el paradigma orientado a objetos, desarrollada, documentada y perfeccionada en este trabajo.

Adopción el presente proyecto como material introductorio a las tecnologías J2EE y al desarrollo de aplicaciones empresariales, enfocado a los nuevos elementos que se integren al trabajo de desarrollo de nuevos sistemas.

Recopilación y revisión del conjunto de convenciones que dan forma y orden a los productos del área, mismos que se ponen a disposición de los encargados de la creación, mantenimiento y documentación de los sistemas. Estos lineamientos enriquecen los artefactos y programas generados, y tienen por intención simplificar las tareas de revisión, organización y mantenimiento de los recursos de software del área de desarrollo.

Despliegue e instalación del producto de software generado a la par del presente proyecto. Ese caso práctico es utilizado de manera conjunta con los textos de este trabajo, como referencia adicional, a manera de guía práctica aplicada y experimentable. Los desarrolladores tienen la libertad de revisar, adicionar, modificar y reutilizar las herramientas, componentes, módulos y fragmentos de software proporcionados.

Se conserva el presente proyecto como referencia a sistemas en desarrollo, cursos internos del área y planeación de proyectos futuros, en temas relacionados con:

- Notación UML.
- Fundamentos del lenguaje Java.
- Introducción al estándar J2EE.
- Desarrollo de proyectos en TVUNAM.
- Apoyo al diseño de aplicaciones de arquitectura multicapa y distribuidos.
- Referencia a las convenciones.

Como punto final, cabe resaltar que queda como trabajo futuro la actualización y ampliación de los temas incluidos en este proyecto.

Capítulo 7.

Conclusiones

CAPÍTULO 7. CONCLUSIONES.

A lo largo de el presente proyecto se ha puesto en relieve la necesidad que, de este tipo de iniciativas, existe en el Área de Desarrollo de Sistemas de TVUNAM, como una alternativa de aprendizaje para introducir a nuevos elementos del área, a los lineamientos, convenciones y tecnologías que sustentan la creación de software adecuado a las necesidades de la dependencia; se ha explicado la importancia del aprovechamiento de la tecnología de EJB para aplicaciones distribuidas, así como de contar con una metodología de trabajo establecida, y se ha delimitado la relevancia de un videojuego como elección a caso práctico de referencia.

Además, se han incluido referencias prácticas a herramientas y tecnologías relacionadas con las actividades del área, asimiladas, comentadas, y organizadas para la mejor comprensión del lector. Entre ellas resalta la introducción al lenguaje de programación Java, como guía para los jóvenes desarrolladores en la comprensión de conceptos fundamentales, la referencia a la notación UML, la introducción a los sistemas distribuidos multicapa y el resumen teórico-práctico de los contenidos de Enterprise Java Beans y el estándar J2EE.
Como parte del desenvolvimiento de este proyecto, un resultado adicional en TVUNAM fue la organización del Área de Desarrollo de Sistemas, quedando implantada una metodología aplicable a futuros proyectos del área, que se favorezcan de las ventajas que las nuevas tecnologías adoptadas reportan a la creación de software.

Es importante notar que, como los estudios de la carrera nos han preparado, se encontró una necesidad que requería solución y, por lo tanto, se llevó a cabo la aplicación de una metodología que permitiera resolver, no sólo éste, sino los proyectos consecutivos a los que se enfrentará el área a la que pertenecemos. Con esto hemos cumplido uno de los muchos objetivos que se nos han impartido en la Facultad.

A lo largo de la investigación realizada en la institución, se ha puesto énfasis en la implantación de una metodología de desarrollo acorde con aplicaciones orientadas a objetos y al estándar J2EE. Esto incluye la revisión de buena parte de lo que la orientación a objetos implica: noción de la notación UML como lenguaje de modelado de objetos, manejo de los conceptos básicos del lenguaje de programación Java, aprendizaje de la tecnología de Enterprise Java Beans para el desarrollo de aplicaciones empresariales, entre otros. Por lo tanto, el presente documento es una guía de las principales tecnologías adoptadas, mismas que, una vez aplicadas al desarrollo de un proyecto concreto, servirá al lector para comprender la conjunción de éstas en un mismo sistema.

Se cumplió con el objetivo principal al crear una aplicación que oriente a cualquier persona, ya sea becario, estudiante o profesional, a seguir una metodología uniforme a todos los proyectos del área de desarrollo de sistemas de TVUNAM. Para esto, el presente documento es tanto manual como guía de referencia práctica, ya que conjuga apartados organizados de cada herramienta, con una aplicación desarrollada para ejemplificar cada una de ellas.

El equipo se enfrento al reto de encontrar un sistema que involucrara los aspectos comunes a los sistemas requeridos por la institución: distribución de recursos, acceso a bases de datos, comunicaciones, concurrencia, manejo de transacciones y seguridad,

por mencionar algunos, y, además, de que el caso práctico a seguir, independientemente de su complejidad, atrajera la atención del perfil antes descrito del personal promedio del área de desarrollo. Por ello, se eligió un videojuego, que cumple con ambas intenciones.

Este conjunto de documentos, ejemplos e implementaciones, proveerán al lector de literatura adicional a la ya existente, enriqueciendo las referencias accesibles actualmente, mismas que, aunque no son limitadas, son escasas y, en su mayoría, complejas o difíciles de relacionar con los problemas reales del entorno de las organizaciones mexicanas. Para ello, se aprovecho la experiencia del equipo de desarrollo de sistemas de TVUNAM, que, desde la incorporación de las tecnologías expuestas, ha dedicado esfuerzo y tiempo mayor en su aprendizaje y documentación, con lo que se responde de forma creativa al requerimiento identificado en TVUNAM de generar a una solución que permitiera a los futuros participantes de su área de desarrollo, la incorporación, simplificada y rápida, a los proyectos, arquitecturas, herramientas, tecnologías y metodología del área.

Es por lo anterior, que no sólo se deja en TVUNAM un resumen, sino toda una aplicación que el usuario podrá revisar a profundidad cuantas veces sea necesario; con los manuales necesarios para su manejo y las referencias que le orienten en el uso de todo lo que se requirió para su desarrollo, enriqueciendo y adecuando las fuentes externas a este trabajo.

Finalmente, se deja toda una estrategia de trabajo propuesta para el desarrollo de los futuros proyectos y no solo herramientas de generación de código, para las futuras generaciones de desarrolladores por venir a TVUNAM.

Apéndice 1.

Importancia de los Videojuegos como Industria

273-A

APÉNDICE 1. IMPORTANCIA DE LOS VIDEOJUEGOS COMO INDUSTRIA.

Cuando se habla de videojuegos se tiende a subestimar el mercado del cual estamos hablando, a tomarlo a la ligera, o incluso a hablar de los mismos con prejuicios. Una causa de esto es que se considera erroneamente que el principal consumidor son los niños. Nada más lejos de la verdad. Se puede ver de esta forma: un niño se deja llevar por una moda y convence a sus padres de adquirir el producto que desea y, por lo tanto, la consola adecuada; el adolescente quizá convenza a los padres para conseguir la consola y empezará a aprender el valor del dinero cuando tenga que trabajar para adquirir los juegos que desea; y, por último, estan aquellos adultos que cuentan con los medios suficientes para adquirir cualquiera de éstos. De una forma u otra, los videojuegos engloban a la mayor parte de la población. Y se dice que, dentro de la industria del ocio, ocupan el segundo lugar después de la televisión, superando al cine o a los deportes.

En los albores de la industria de los videojuegos, estos eran fabricados por un grupo reducido de personas y tan sólo se distribuían con sus conocidos; sin embargo, con la madurez de las nuevas tecnologías y la comercialización de consolas y PCs, estos canales de distribución se han "engrosado", de tal forma que lo que antes podía ser considerado como un negocio de cochera, se ha convertido en una industria capaz de generar ganancias que se miden en miles de millones de dólares a nivel mundial.

A continuación se presenta una cronología de algunos de los hechos más importantes en cuanto a la industria del videojuego.

Historia de los Videojuegos.

El origen de los videojuegos, a diferencia de lo que mucha gente pudiera pensar, no se encuentra en el juego de Pong (1972), sino en el juego llamado "Tennis for Two", desarrollado en 1958 por Willy Higinbotham en el centro de investigación nuclear de Brookheaven. Este juego era realmente simple y su dispositivo de salida era la pantalla de un osciloscopio.

De igual forma, dos de las tres principales compañías dedicadas a la fabricación de consolas tienen sus orígenes mucho antes de que se concibieran las computadoras como medios de diversión, tal es el caso de Nintendo, establecida en 1889 como la Marufuku Company (dedicada a la fabricación de barajas) y Sony, establecida en 1947 como la Tokyo Telecommunications Engineering Company.

En 1962 Steve Russell crea el primer videojuego interactivo: SpaceWar en una PDP-1, máquina utilizada en ese entonces para realizar simulaciones sobre energía nuclear y gravedad. Este juego consistía de caracteres ASCII que simulaban dos naves que intentaban destruirse una a la otra.

En 1972 Nolan Bushnell funda Atari, la primer compañía dedicada al desarrollo de videojuegos, siendo su primera creación el juego de Pong, desarrollado principalmente por Al Alcorn. Uno de los puntos más fuertes de este juego era su simplicidad y facilidad de control, razón por la que se hizo bastante popular en los años siguientes.

Durante la década de los 70's compañías como Magnavox y Coleco comenzaron a comercializar los primeros sistemas caseros de videojuegos, convirtiéndolo en un éxito total, sin embargo, debido a que estos eran los primeros pasos de la industria, muchas de estas máquinas no contaban con el apoyo suficiente, razón por la que muchas de estas, no soportaron el ataque de las compañías japonesas en la década de los 80's y tuvieron que cerrar sus puertas o dedicarse a otros negocios.

A pesar de que en la década de los 70's los mayores avances en cuanto a la creación de videojuegos se desarrollaron en los Estados Unidos, los japoneses rápidamente comenzaron a tomar parte en este negocio. A finales de los 70's el presidente de Nintendo, Hiroshi Yamauchi cambia totalmente el enfoque de la compañía y basa sus principales productos en formas de entretenimiento que utilicen los mayores adelantos tecnológicos, siendo sus principales productos los sistemas portátiles "Game & Watch" creados por Gumpei Yokoi[64].

Es en esta misma década cuando se comienza a dar la llamada "Guerra de las Consolas", cuando Atari con su Atari 2600, Coleco con su Colecovision y Magnavox con su Intellivision comienzan a publicitar sus consolas para atraer la atención de los consumidores. Una de las principales ventajas de estas consolas es el cartucho, medio utilizado para almacenar los juegos. Gracias a esto, las personas podían utilizar el sistema para jugar una infinidad de videojuegos.

Desafortunadamente para estas compañías, las cuales no contaban con un sistema que prohibiera a cualquier persona a fabricar sus juegos y comercializaros, cayeron en la bancarrota; este error fue aprovechado por Nintendo; el cual, con su Famicom (acrónimo de Family Computer) tomó por sorpresa tanto al mercado japonés como al mercado norteamericano, ofreciendo una consola que ofrecía mejores gráficas, un control[65] totalmente distinto al joystick utilizado hasta la fecha y una política de calidad que impedía la fabricación de juegos que no contaran con normas mínimas de calidad.

Gracias al Famicom, Nintendo estableció las reglas tanto en el desarrollo de Hardware como de Software, sentando así las bases para lo que en la actualidad es una de las mayores industrias a nivel mundial.

A principios de la década de los 90's Nintendo y Sega lanzan sus consolas de 16 bits: El Super Nintendo y el Génesis, respectivamente. Ambas consolas contaban con el apoyo tanto de los grupos internos de desarrollo de sus fabricantes como de algunas compañías formadas explícitamente con la intención de desarrollar juegos. Estas consolas protagonizaron los "Años Dorados" de esta industria.

Es en este período donde surge el tercer competidor (y actual dominador del mercado): Sony. Inicialmente Sony desarrollaría un sistema de CD que se incorporaría

[64] Gumpei Yokoi fue además el creador del GameBoy, GameBoy Color, Virtual Boy, el control pad (utilizado en todos los controles actuales). Es considerado por muchos como el verdadero innovador dentro de Nintendo.

[65] El control utilizado por Nintendo se conoce como Control Pad y en lugar del Joystick, utiliza flechas direccionales para controlar los movimientos del personaje. Esta innovación que aún en los sistemas modernos se considera como un estándar fue desarrollada por Gumpei Yokoi.

al Super Nintendo, pero debido a problemas sobre quien se quedaría con las regalías de los juegos vendidos, Nintendo cancela el contrato con Sony. Sony no desperdiciaría el trabajo realizado y lo tomó como base para el PlayStation (PSX), consola que tenía el objetivo de demostrar a Nintendo que no era el rey indiscutible.

En esta década Nintendo lanza al mercado el sistema más exitoso de todos los tiempos: el GameBoy, sistema basado en los Game & Watch, pero con la ventaja de ofrecer un sistema portátil de entretenimiento. Hasta la fecha, el GameBoy ha vendido más de 100 millones de unidades a nivel mundial.

A mediados de los 90's Sony lanza su PlayStation, consola de 32 bits, mostrándose como la principal competidora del Sega Saturn (consola de 32 bits de Sega) y el próximo a salir Nintendo64. Desde nuestro punto de vista, el principal hecho que dio el dominio de mercado a Sony sobre Sega y Nintendo fue en primera instancia la decisión de Nintendo de seguir utilizando los cartuchos como medio de almacenamiento de sus juegos, a diferencia de Sony que utilizaba CD. Este, aunado a la estricta política que imponía Nintendo a las casas desarrolladoras de software para publicar juegos en sus sistemas, obligaron a varias compañías a publicar sus juegos en la consola de Sony en lugar de hacerlo para Nintendo.

A finales de la década pasada se cumplía un ciclo más y terminaba otra guerra de las consolas en las que existió una baja. Sega lanzó en 1999 el Dreamcast, primera consola de 128 bits y la primera lista (de fábrica) para poder jugar en línea. Sin embargo, la consola no tuvo el éxito esperado, y después de lo demostrado por Sony con su PlayStation2, las ventas del Dreamcast bajaron estrepitosamente. Por esta razón, Sega anunció en el año 2000 que dejaba el negocio de la fabricación de las consolas para dedicarse exclusivamente al desarrollo de software para las tres consolas de nueva generación: PlayStation2 de Sony, GameCube de Nintendo y el XBox de la recién incorporada Microsoft.

Juegos para PC.

Una de las primeras computadoras en las que se jugaron juegos fue la Sinclair ZX Spectrum, así como el Atari 400. Estas computadoras, equipadas con un teclado, ranura para cartuchos y especificaciones irrisorias si se comparan con las características de PC's actuales. datan de la década de los 80's y tenían un costo aproximado de $300 a $400 dólares.

A estas computadoras se les unió la Commodore 64 y Commodore 128 de la empresa Commodore. Esta computadora ofrecía además del teclado, una drive para discos, accesorios para impresión y una casetera. Esta computadora fue bastante popular entre los jugadores, debido en parte a que muchos de ellos podían crear sus propios juegos y distribuirlos entre sus conocidos. Inicialmente, los juegos para la Commodore se comercializaban en formatos de casete, pero con el advenimiento del diskette, la importancia del casete como medio de almacenamiento se redujo.

En un esfuerzo para mejorar la plataforma en la que se jugaban los juegos, Commodore lanzó al mercado la computadora conocida como Amiga, la cual no sólo era capaz de desplegar gráficos impresionantes para la época, sino que fue además, una de las primeras en dar mayor énfasis al sonido.

Para la década de los 90's el impacto de la Commodore tuvo que ceder ante la creciente fortaleza de la PC de IBM, la cual hasta el momento había sido utilizada principalmente para aplicaciones de oficina. En esta década, se paso del sistema de video CGA (Computer Graphics Adapter) con 4 colores al VGA (Video Graphics Adapter) con 256 colores y al SVGA (Super VGA) con casi 16 millones de colores.

Sin embargo, no fue sino hasta el advenimiento de las tarjetas de sonido que la PC se comenzó a considerar como una plataforma sobre la que se pudiera jugar. El juego que inició la avalancha de juegos para la PC fue Wolfenstein 3D[66], el cual, además de contener gráficos espectaculares (simulaba un entorno en tres dimensiones) gracias a la utilización de la tarjeta Sound Blaster[67] ofrecía una experiencia envolvente a los jugadores.

A mediados de los noventa se comenzaron a comercializar las computadores 386 y 486, con procesadores que alcanzaban velocidades de 100 MHz. Fue en esta época donde el segundo juego más influyente fue lanzado: Dune 2, el cual ignauguraba el género de Estrategia en Tiempo Real (RTS), siendo la base para juegos como Command and Conquer (WestWood Studios) y Starcraft (Blizzard).

Para finales de la década de los 90's y del siglo XX los avances en hardware, han permitido a los desarrolladores crear juegos que rivalizan, en cuanto a calidad gráfica, con los creados para las consolas caseras. Cabe hacer mención que los géneros más populares dentro de la PC son los de simulación (SimCity, The Sims, desarrollados por Maxis), los juegos en primera persona como Quake, RTS como Starcraft, Age of the Empires (Ensemble Studios/Microsoft) y los juegos de rol como Diablo (Blizzard).

Juegos en línea.

Así como mucha gente tiene la idea errónea de que el primer videojuego es Pong, mucha gente piensa que los juegos en línea tienen su origen alrededor de 1994 con Doom (de idSoftware), sin embargo, la historia de los juegos en línea data de finales de la década de los 60's.

En 1969 Rick Blomme, un estudiante del MIT crea una versión para dos jugadores del juego Spacewar (creado en 1962 por Steve Russell) para el sistema PLATO, Logica Programada para Operaciones Automáticas de Enseñanza (Programmed Logic for Automatic Teaching Operations). Este sistema fue uno de los primeros en soportar una red de computadoras con el objetivo de dividir los datos entre múltiples terminales con fines académicos. Para 1972 este sistema contaba con más de 1,000 computadoras en los Estados Unidos.

Para 1977 aparecen juegos como Zork, una aventura RPG basada en texto inspirada por la serie Calabozos y Dragones; Airfight un simulador de combate basado en Spacewar, así como la serie Wizardry! para el sistema PLATO. Juegos como Zork eran utilizados por alumnos del MIT que se conectaban a la DEC PDP-10 de la institución para poder jugar, debido a la popularidad de este juego, se restringió la hora de juego a las tardes, debido al alto consumo de recursos por parte del juego.

[66] Juego desarrollado por la compañía idSoftware.
[67] Creative Labs.

En 1979 Roy Trubshaw y Richard Bartle, estudiantes de la Universidad de Colchester en Esssex, Inglaterra, crean la primera versión de un MUD o Calabozos Multiusuarios (Multi User Dungeon), la cual duró nueve años en servicio. A pesar de que el código de este juego esta protegido este es compartido con otras Universidades para fines educativos; sin embargo el código empieza a ser distribuido de forma clandestina y proliferan las copias de este, popularizando su uso dentro de las Universidades.

En 1982 se funda Kesmai Corporation, la cual es contratada por CompuServe para crear un juego de rol basado en texto (ASCII[68]). Este juego vería la luz con el nombre de Kesmai Islands. En ese mismo año CompuServe compra los derechos de un simulador de combate llamado DECWars y Kesmai lo re-escribe y lo lanza, en 1983, con el nombre de MegaWars I. Este juego es el que ostenta el record de más tiempo funcionando ya que fue cerrado hasta 1998.

En estos años jugar juegos en línea no era barato ya que se pagaba una tarifa por hora, la cual podía llegar hasta los $12 dólares dependiendo del juego. Esta situación cambió para 1984 cuando la compañía AUSIE (posteriormente Mythic Entertainment) con su juego Aradath estableció la suscripción mensual, misma que hoy es la forma habitual de cobro.

En 1985 Bill Louden convence al Departamento de Información de General Electric para establecer un servicio de juegos basado en ASCII como el de CompuServe. Así nace Genie, acrónimo de GE Network for Information Exchange o Red de General Electric para el intercambio de información. La principal ventaja que ofrecía con respecto a CompuServe era que su tarifa era 50% menor y ofrecía una cantidad similar de juegos.

En ese mismo año, Quantum Computer Services (ahora America On Line) anuncia QuantumLink, un servicio basado en gráficos exclusivo para usuarios de computadoras Commodore.

Para 1986 Kesmai re-programa MegaWars I y lo lanza con el nombre de Stellar Warriors para GEnie, este juego se muestra ese mismo año en San Francisco sobre una red de computadoras Macintosh. Ese mismo año, Jessica Mulligan, al ser cancelada su cuenta de MegaWars I, crea un nuevo concepto conocido como PBeM o Juego por Correo Electrónico con el juego The Rim World Wars.

Quantum Computer Services y LucasFilms desarrollan Rabbit Jack's Casino, haciendo uso de las capacidades gráficas anunciadas por Quantum.

Para 1987 se lanzan los juegos Air Warriors y Stellar Emperor (basados en Stellar Warrior), así como Gemstone (juego de rol) para GEnie, siendo este último el más popular de todos, a pesar de que no hacía uso de gráficas sofisticadas y se basaba enteramente en texto.

Para principios de los 90's ya existía una gran cantidad de juegos en línea, la mayor parte de estos podían ser jugados por suscriptores de GEnie, misma que a pesar

[68] American Standard Code for Information Interchange.

de su popularidad, pronto vería el final de sus días al rechazar propuestas de creación de nuevos juegos de rol basados en texto. De la misma forma Quantum se consolida como AOL y CompuServe desaparece siendo adquirida por AOL.

En 1993 DARPA-net pasa a ser del dominio público bajo el nombre de Internet, nacen así los primeros navegadores MOSAIC[69] y Netscape. Gracias al desarrollo de mejor hardware para conexión, se empiezan a formar las primeras LAN's (Redes de Area Local) dedicadas exclusivamente a los juegos en línea aprovechando las mayores velocidades de transferencia.

1994 marca la pauta de lo que serían los siguientes juegos en línea. La compañía idSoftware lanza Doom un juego en Primera Persona (First Person Shooter o FPS) con la capacidad de conexión de diferentes usuarios en un mismo juego. A pesar de que no se puede utilizar este juego dentro de Internet (sólo en una LAN) el juego es un éxito.

Al año siguiente, los creadores de Doom lanzan Quake, siendo este el primer juego que realmente puede ser jugado a través del Internet. Uno de los principales atractivos de estos juegos (Doom y Quake) es el de sus gráficos, los cuales sin ser muy complejos si dan al jugador la sensación de un mundo más real. Sin embargo, para estas fechas, muchos MUD's siguen activos.

Ese mismo año se lanza Ultima Online, juego de rol en línea que tiene la virtud de ser el primero en presentar de forma gráfica un mundo permanente en el que los usuarios cuentan con una identidad, pueden convivir, adquirir posesiones, etc. Así como Doom y Quake marcaron el camino de los FPS en línea, Ultima Online marcó el terreno para los siguientes juegos de rol.

En cuanto a las consolas caseras, cabe hacer otra aclaración. El Dreamcast de Sega fue la primer consola que estaba fabricada con la idea de ser utilizada para juegos en línea, apoyada por una gran infraestructura; sin embargo, no fue la primera en ofrecer este servicio. Durante la década de los 80's Atari con Atari VCS, Intellivision (PlayCable) y Nintendo ofrecieron servicios de conexión.

[69] Desarrollado en el NCSA (National Center for Super Computing Applications), Universidad de Illinois por Tim Fool.

Panorama actual de la industria[70].

Actualmente, la bonanza y ganancias registradas en la industria del videojuego, se ha visto favorecida por la existencia de tres competidores (dejando atrás el mercado de la PC), situación para nada inusual en este mercado. Nintendo, Sony y Microsoft se disputan la supremacía en cuanto al mercado de los videojuegos[71], siendo hasta el momento Sony con más de 40 millones de unidades vendidas a nivel mundial de su consola PlayStation 2 la clara ganadora, dejando a Nintendo (Gamecube) y Microsoft (XBox) algo atrás. Para recalcar la importancia de las consolas y la industria en general, cabe hacer notar que Sony Computer Entertainment (SCE) nació como una filial de Sony Music Entertainment, pero a partir del año 1999, las ganancias de SCE cuentan para más del 50% de todo el corporativo Sony.

Como se mencionó anteriormente, atrás han quedado aquellos días en los que la distribución de un juego consistía en compartir el código fuente o mostrarlo a un grupo de amigos; actualmente para la comercialización de un videojuego existen complicados canales de distribución que implican, a el grupo desarrollador, un publicista, el fabricante del producto terminado y una empresa dedicada a la promoción del juego para que este cuente con el impacto que pueda hacer de este un producto exitoso.

Como cualquier otra industria del entretenimiento, los videojuegos han tenido que enfrentarse a la piratería, ya que si bien las ganancias registradas de la industria lícita son impresionantes, serían aún mayores si la piratería no existiera. Se cuenta que en Estados Unidos, 1 de cada 4 juegos que se vende es pirata, en nuestro país, 1 de cada 2 es pirata. Esto puede parecer como un ahorro para los consumidores, sin embargo, es perjudicial, ya que gracias a la piratería, México no es visto como un país viable para la inversión en este campo, lo que ha dado como resultado la cancelación de varios proyectos orientados a hacer de México un país productor de software de entretenimiento.

El crecimiento en las fuerzas de oferta y demanda, el aumento de las utilidades obtenidas, la difusión de las tecnologías de información y de esta forma de particular de entretenimiento, así como el fortalecimiento de un mercado, son algunos de los factores que han favorecido a la evolución de una importante industria.
Originalmente englobada en el ramo de los juguetes o los electrónicos, hoy en día, la industria del videojuego, representa un gran contendiente dentro del mercado del entretenimiento".

Ganancias registradas.

El punto más importante para la creación de una empresa de cualquier giro son las ganancias que se esperan obtener a partir de una inversión. Dentro de esta industria las nuevas empresas tienen un campo difícil de entrada, necesitan empezar a acaparar un mercado que le permita la estabilidad y el progreso económico. Una vez obtenido el reconocimiento, la empresa competidora debe mantener su posición por diferentes medios, haciendo un balance entre inversión y recuperación de la misma.

---

[70] Hasta marzo de 2003.
[71] A partir del año 2001.

La industria de los videojuegos en Estados Unidos había obtenido, hasta finales del 2001, una facturación de 4 300 millones de dólares, un 34% más que en el mismo periodo del año anterior, según un informe de la IDSA. Actualmente se registran ganancias anuales de más de 20 mil millones de dólares anuales sólo en México, considerado como un mercado débil si se compara con los 200 mil millones de dólares registrados anualmente dentro de los Estados Unidos. Esto ubica a la industria del videojuego como una de las principales industrias del entretenimiento[72].

Sin embargo, dada una inversión de 4 millones de dólares por juego, se requiere de una venta mínima de 100,000 unidades (a $40 dólares cada una) para recuperar la inversión inicial, y aún así, el grupo desarrollador tan sólo puede ver una pequeña parte de tales ganancias.

Relaciones entre empresas.

Se podría pensar que las relaciones entre las empresas en este mesrcado estan regidas por la competencia, aún más si tomamos en cuenta que, actualmente, ésta industria está dominada por las tres grandes empresas anteriormente mencionadas, y que, para una empresa debutante en este rubro, las opciones se reducen a entrar al mercado desarrollando para las consolas establecidas, o bien, probar suerte en el mercado que podría considerarse más libre: los juegos para PC.

Sin embargo, existen ciertos eventos que obligan a las empresas a reconocer en la competencia a un aliado para obtener un mejor producto. Cuando la inversión no es recuperada y, antes de optar por la quiebra, la empresa tiene diferentes opciones para salir adelante ya sea con productos innovadores que incluyan a otras de giro distinto o fusiones entre fabricantes y licenciatarios.

Es importante destacar un problema presente, las empresas desarrolladoras de juegos se enfrentan a la reducción (recursos humanos, inversiones riesgosas, productos), concentración del mercado y estandarización de productos, resultado de la crisis general. La industria del videojuego se cura a sí misma cancelando proyectos, cerrando estudios de desarrollo, fusionando divisiones y apostando por el tipo de productos que ha funcionado en el pasado. Aún así, empresas han estado a punto o han cerrado por causa de las pocas ganancias que reciben y que se encuentran muy por debajo de lo que esperaban como mínimo. Por lo tanto, necesitan de la aplicación de medidas que los mantenga, ya no digamos en competencia, sino a flote; siendo una de las cuales, la fusión o los acuerdos con los fabricantes de consolas.

Tomando en cuenta todos estos factores, podemos decir que la industria del videojuego, ha pasado de ser una industria improvisada, a ser una industria controlada por empresas transnacionales con enormes cuentas bancarias, mismas que les permiten ofrecer un producto que es atractivo a todo grupo de consumidores. Sin embargo, como cualquier industria, esta es cambiante, y aquellos que alguna vez estuvieron en la cima, ahora buscan asociarse con otras para poder hacer frente a los nuevos retos y permanecer (directa o indirectamente) en el negocio.

[72] Hasta marzo del 2003. Estudio realizado por la IDAS (www.idsa.com)

Tendencias tecnológicas.

De forma análoga a muchas otras industrias, el hecho de contar con tecnología "de punta" y aprovecharla de formas novedosas en los videojuegos, representa un factor de atractivo e incluso de sobre vivencia de un juego en el mercado. Público más enterado y competidores de vanguardia marcan pautas a seguir, ya sean gráficos y audio más sofisticados, hardware con capacidades superiores, mejores formas de almacenamiento, nuevas técnicas de desarrollo y en general mejores aplicaciones de herramientas y técnicas. Además, este proceso de innovación continua, "quien no se mueve, muere", promueve la creación y crecimiento de tecnologías, aplicables a otras industrias, y que representa aportaciones importantes. No por nada los juegos son proyectos que benefician la exploración y explotación de la tecnología, como en el caso del nuestro propio proyecto. Sin embargo, no todo es bueno en el uso de las tecnologías en este ramo, debemos considerar que, aplicaciones tecnológicamente muy complejas, pueden redundar en altos costos de desarrollo, gran atractivo visual y auditivo, pero poca creatividad e innovación. Fórmula, a largo plazo, de un éxito moderado o nulo, o de productos exitosos, pero poco propositivos".

La principal atracción del juego.

Actualmente, el mercado del videojuego ofrece gran cantidad de productos y esa variedad pone a los potenciales compradores en la disyuntiva de qué producto adquirir, qué propuesta preferir. Además se genera tanta información, que al final deriva en público más enterado y más consciente de sus preferencias.
Como es lógico, los juegos más atractivos serán los más solicitados, las propuestas más originales, los más exclusivos, en resumen, los más creativos, serán los que permanecerán dejando antecedente de su existencia.
La creatividad se puede expresar en muchos aspectos, en todos para ser precisos. Por mencionar algunos:

- Crear una moda o seguirla (juegos basados en películas o eventos deportivos actuales)
- Presentar una historia interesante y envolvente que tenga el interés de un texto literario (Eternal Darkness por mencionar alguno)
- Innovar en la presentación del juego, arte,, gráficos, sonido, ambientación integrados de forma original y atractiva.
- Presentar nuevos objetivos, modo de juego, retos y actividades, adecuados al público al que nos dirigimos, y qué además, nos retribuya de forma suficiente vencerlos.
- Incluir toques ingeniosos y humor en los diálogos o situaciones y detalles remarcando la actitud lúdica del programa.
- Agregar "valor de rejuego" (del termino anglosajón "replay value") es decir, motivar a que una vez terminado el juego se pueda seguir disfrutando del mismo, ya sea con nuevos retos, más opciones, minijuegos, o un modo de juego original y adictivo.
- Cuidar cada detalle de la promoción y comercialización del producto, desde su empaque hasta su difusión son campañas publicitarias novedosas y atractivas

Productos más propositivos, innovadores y creativos tienen más oportunidades en este difícil mercado.

La mercadotecnia

Los videojuegos son productos comerciales, y como tales, requieren de un respaldo mercadotécnico sólido para posicionarse dentro de la industria. Esto incluye desde seleccionar adecuadamente el público consumidor, diseñar una imagen atractiva, establecer la forma de distribución del producto, hasta determinar la forma de publicitarlo, el momento idóneo de lanzamientoy planear médios para retroalimentarnos con los comentarios de nuestros clientes.

Las empresas no esperan por los buenos comentarios de los medios, los provocan. Tal es el caso de la E3[73], una convención en la que se reúnen las empresas competidoras para hacer gala de lo que ofrecen a los usuarios, para dar una probada de lo que su producto es, enmascarado en toda una serie de eventos que provocan la sensación de euforia y satisfacción en el usuario con sus ruidos incesantes y sus colores atractivos.Es decisión de cada compañía anunciar cuándo y en dónde se lanzará su producto, esto crea un ambiente de expectación que provoca, a su vez, una compra masiva del producto en cuanto sale al mercado.

Durante las fiestas navideñas parece que se abre un periodo de veda convenientemente programada por la misma industria del entretenimiento, que inunda, especialmente a partir de noviembre, los estantes de los comercios. En esta apuesta intervienen los hábitos consumistas y la creencia de que, si un producto falla en su cita invernal será barrido por la competencia.

En otro aspecto, un buen recurso que explota Nintendo es el crear una moda de uno de sus personajes, darle toda una vida en uno o varios juegos. Se ha generado todo un fenómeno por la serie creada y esto permite que las empresas de entretenimiento hagan de todo, desde llaveros hasta construcciones, sin olvidar naturalmente, a los videojuegos.

Ahora viene un fenómeno interesante, antes los videojuegos pugnaban por ser anunciados en los anuncios de alguna marca reconocida, hoy la situación ha cambiado. La industria de los videojuegos esta compitiendo fuertemente con el mercado televisivo y Hollywood para convertirse en el rey de lo que se conoce como "*emplacement*". Así como las compañías luchan por incluir en series televisivas o en las películas de moda sus productos y marcas, actualmente buscan establecer su imagen atraves de los videojuegos, que en ocasiones, logran alcanzar a una población mucho mayor que otros medios. Actualmente, las compañías pagan más de 100 millones de dólares[74] para poder aparecer como el patrocinador de un juego. Así, las ganancias de las compañías por concepto de publicidad tenderán a aumentar conforme los videojuegos lleguen a más personas.

Esto se ha convertido en una alianza que pronto será una rama del negocio. La empresa que haga el videojuego venderá el espacio publicitario al mejor postor, sin ningún problema. Después de todo, un potencial consumidor va a estar frente al juego tres horas diarias durante cerca de tres meses.

[73] Electronic Etertainment Expo
[74] Montos estimados al año 2003.

Piratería.

La piratería es un tema que, aunque no es exclusivo de los videojuegos, resulta igualmente aterrador para los desarrolladores y fabricantes de consolas que para el resto de las empresas que sufren el problema.

No conforme con la competencia entre empresas, la disputa de quién programará los juegos para qué consola, las críticas publicitarias que tanto afectan a las ventas, se tiene la mayor pérdida que las industrias de videojuegos, CDs de música, desarrolladores de software, etc., sufren: La piratería.

Con más frecuencia cada día, las empresas se enteran que el pedido que salió para determinado lugar fue asaltado y se perdió una gran suma de consolas o juegos. En poco tiempo se sabe que un mercado negro florece por la adquisición de los nuevos juegos y consolas, esto debido a la facilidad para reproducir copias de los juegos o a lo fácil que es encubrir las consolas robadas.

O, en el caso de Internet, sólo se requiere un poco de atención y un buen módem para convertirse en un pirata cibernético que tenga las últimas películas con una calidad aceptable o las canciones de su autor favorito sin necesidad de comprar absolutamente nada. ¿Y qué pueden hacer las empresas para defenderse? Casi nada ya que lo poco que hacen es superado rápidamente. Y a eso se le suma la falta de una legislación que castigue severamente este tipo de faltas, por lo que lo convierte en un peligro sólo a boca por el riesgo tan pobre que se corre. Siendo el caso contrario, el castigo más grande para los establecimientos en materia de ley ya que se muestran severos si los juegos exhibidos en locales formales no cumplen con las indicaciones de categorías de juegos o si no cumplen con la distancia mínima hacia una escuela.

Por lo tanto, se ha desarrollado un mercado en el que puedes obtener un mismo producto hasta 70% más barato que el original en una tienda formal. Por ejemplo, si un videojuego cuesta 100 dólares en el mercado formal, en el informal es posible encontrarlo en 30 dólares o menos, ya que fue introducido al país de manera informal o fue robado.

En México, el fenómeno se ha dado desde siempre y con la entrada de tecnología en computadoras y el mejoramiento de la conexión por Internet doméstica, el problema afectará más. Por los llamados "manteros" o ambulantes, varias pequeñas empresas vendedoras de discos han tenido que cerrar porque no pueden evitar esa competencia desleal y se ven atados de manos cuando los culpables no tienen castigo. Así mismo no sólo perjudica a las grandes compañías, sino también al canal de distribución y al Estado porque deja de percibir impuestos.

Además es de mencionarse la forma en la que nuestro país es excluido como opción para inversión de los desarrolladores de juegos gracias a esta problemática y la imagen que genera. Al ver la industria del videojuego las grandes pérdidas ocasionadas por este fenómeno, éstas no se animan a abaratar los precios de sus productos, o adecuarlos a nuestro mercado, cayendo en un círculo vicioso donde el negocio de la piratería es el único ganador.

Desarrollo Independiente. ¿Hay cabida para la innovación?

Debido a la competitividad en la industria, cada vez son menos las empresas que se aventuran a patrocinar conceptos innovadores. Si un tipo de juego es exitoso, los desarrolladores se encargan de emular dicha fórmula para obtener ventas aceptables que les permita patrocinar sus futuros proyectos. Esto se ve fortalecido debido a que los consumidores generalmente siguen tendencias con lo que se reduce el apoyo a proyectos novedosos por no encontrarlos "de moda". Así, sólo aquellas compañías exitosas como Nintendo se arriesgan a intentar ideas innovadoras que ofrezcan a los jugadores nuevas formas de entretenimiento. Podemos citar a Pokémon, comercializado en 1994, que a pesar de su popularidad, se le negó una oportunidad desde 1991 que surgió la idea, y actualmente ha demostrado ser un concepto innovador y con ventas nada despreciables.

Parte de esta falta de innovación se debe a la tecnología, que en muchos casos parece matar a la creatividad, ya que los desarrolladores se concentran más en cómo ofrecer más efectos visuales (eye candy[75]) que en ofrecer nuevas ideas. Sin embargo, esto no sucede siempre. Podemos citar a la *Cero: Convención de Desarrolladores Independientes* llevada a cabo en el año 2002 que se propuso como meta la aplicación de un adelanto tecnológico como base para el desarrollo de nuevas ideas. En este caso se planteó la pregunta ¿Qué es posible hacer con 150 000 elementos bidimensionales en pantalla (lo máximo permisible con la tecnología actual[76])?. Después de 4 días se obtuvieron más de 10 juegos innovadores; así, en lugar de ver a la tecnología como un limitador de la creatividad, esta puede ser vista como trampolín para el desarrollo de nuevas ideas. De la misma forma se puede citar a Zelda The Wind Waker, que utiliza la nueva tecnología[77] para ofrecer al jugador no sólo el perfeccionamiento de un estilo utilizado, sino una mayor interacción y nuevas formas de juego dentro de un concepto conocido por los jugadores.

Desafortunadamente en muchos casos, las ideas innovadoras o no consiguen una promoción adecuada o son robadas por terceros. Esto se da cuando los diseñadores presentan proyectos a las productoras y estas rechazan dichos proyectos, para posteriormente utilizar dichas ideas como si fueran de ellos. Sin embargo, con un buen uso de patentes, es un riesgo que debe correrse si se quiere obtener un contrato.

[75] Término anglosajón utilizado para haver referencia a un elevado atractivo visual.
[76] Máquinas PentiumIV.
[77] Nintendo GameCube

México y los videojuegos.

Desafortunadamente en México no existen filiales directas de las principales compañías desarrolladoras, esto con la excepción de Microsoft que es la única que cuenta con una división especializada para México y América Latina (además de que en Guadalajara es donde se ensambla la XBox). Tanto Nintendo como Sony, a pesar de contar con participación en México, no cuentan con filiales directas como en el caso de Europa (Nintendo Europe y Sony Computer Entertainment Europe) que dan la oportunidad a desarrolladores de esos lugares a participar en el proceso de la creación de juegos y por lo tanto fomentar el crecimiento de la industria.

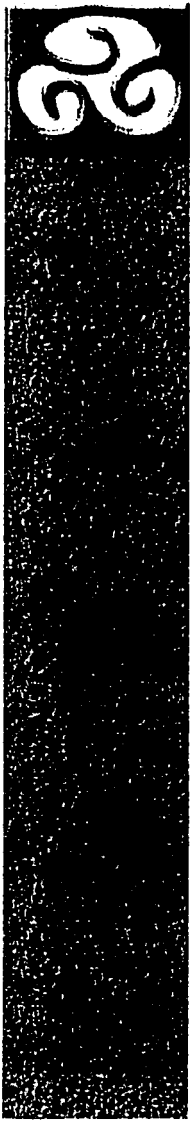
En México se han cancelado inversiones de este tipo debido a la piratería. Así, la formación de una división de Nintendo que a mediados de los noventa tenía la intención de traducir juegos japoneses y norteamericanos para su distribución nacional, como la creación de nuevos juegos nunca vió la luz del día.

Afortunadamente esto está cambiando con la entrada de Microsoft y su XBox, la escuela de desarrollo Aztec Tech y los esfuerzos de un grupo mexicano de desarrolladores llamados RadicalStudios, formados por estudiantes universitarios de la facultad de Ingeniería, cuyo juego, Eranor, ha sido uno de los principales esfuerzos por crear un juego en línea orientado al mercado nacional los que han dado un paso más para poner a México no sólo como un consumidor, sino como un productor de calidad en esta industria.

No obstante, en el caso del juego Eranor, este no ha tenido la publicidad adecuada, ya que sólo gente allegada al grupo de desarrollo o que participan de sus intereses comunes han conocido el juego. Sin embargo con la creación de eventos enfocados a la publicidad de juegos como el E3[78] pero en el mercado latinoamericano pueden dar a estos jóvenes la oportunidad de dar a conocer su trabajo y obtener el patrocinio adecuado a sus proyectos.

[78] Electronic Entertainment Expo. Se lleva a cabo anualmente en los Estados Unidos y tiene la finalidad de que las compañías demuestren sus nuevos adelantos para su futura comercialización.

Convenciones



286-A

Lineamientos de presentación de código.

Este apartado incluye los lineamientos que se siguieron al presentar el código implícito en el desarrollo del proyecto.

Convencion	Descripción
Comentarios iniciales de una clase	Todos los archivos fuente deben comenzar con un comentario que indique el programador, o programadores, la fecha, una nota de copyright y una pequeña descripción del propósito de la clase.
Sentencias import y package	Debe de especificarse cada clase utilizada en los import. En caso de que se utilice todas las clases se recurrirá al asterisco(*).
Indentación	Deben tomarse 4 espacios como unidad de indentación. La construcción exacta de la indentación (espacios o tabuladores) no es crítica. Los tabuladores deben estar fijados exactamente cada 8 espacios (no cada 4).
Longitud de líneas	Se deben evitar líneas de más de 80 caracteres, de lo contrario, no son manejadas correctamente por muchos terminales y herramientas. Los ejemplos de uso en la documentación deben ir en líneas más cortas, generalmente de no más de 70 caracteres. Por lo tanto debe existir un corte de línea: • Cuando una expresión no cabe en una sola línea, debe saltarse a la siguiente línea de acuerdo con los siguientes principios generales: • Saltar después de una coma • Saltar antes de un operador • Son preferibles los saltos de alto nivel a los de bajo nivel • Se alinea la nueva línea con el inicio de la expresión del mismo nivel de la línea inmediatamente anterior • Si las anteriores reglas dejan el código confuso, o demasiado compactado sobre el margen derecho, se colocarán indentaciones de 8 espacios en su lugar
Comentarios de Implementación	Aclaran el código o explicar una determinada implementación. Hay dos tipos: **Bloque** Los bloques se utilizar para proporcionar descripciones de archivos, métodos, estructuras de datos y algoritmos. Se deben utilizar al comienzo de cada archivo y antes de cada método. También se pueden emplear en otros lugares, como por ejemplo dentro de métodos. Los bloques que se encuentren dentro de una función o método deben estar indentados al mismo nivel que el código que describen. Un comentario de bloque siempre debe ir precedido por una línea en blanco para separarlo del resto del código. Cada una de las líneas del bloque debe comenzar siempre por un asterisco (*), excepto la primera. /* * Comentario de Bloque. */ **Línea simple** Los comentarios cortos pueden aparecer en una sola línea indentada al nivel del código que la sigue. Si un comentario no cabe en una línea, debería seguirse el formato explicado para el bloque. Es conveniente colocar una línea en blanco antes de la línea simple de

Convención	Descripción
	comentario
Comentarios de Documentación	Los comentarios de documentación describen las clases Java, interfaces, constructores, métodos y campos. Cada uno de estos comentarios está delimitado por /**...*/, con un comentario por API, que debe aparecer justo antes de la declaración del elemento de la clase a comentar.
Sentencias Simples	Cada línea solo debe contener a una sentencia.
Sentencias Complejas	• Las sentencias incluidas deben estar indentadas un nivel más que la sentencia compuesta • La llave de apertura debe colocarse al final de la línea en que comienza la sentencia compuesta; la llave de cierre debe ir al comienzo de una línea e indentarla al mismo nivel de la sentencia compuesta • Deben usarse llaves para todas estas sentencias cuando forman parte de una estructura de control, como una sentencia if-else o for. Esto hace más fácil incorporar sentencias sin que se introduzcan errores debido a llaves olvidadas.
Sentencias compleja return	Si devuelve un valor, éste no debe ir entre paréntesis a no ser que su uso haga el valor de retorno más obvio o más claro.

Sentencia compleja if-else-else if	Debe seguirse la siguiente convención.
	if (condicion) {
	sentencias;
	} else {
	sentencias;
	}else if{
	..... sentencias
	}

try-catch	Una sentencia try-catch debería tener el siguiente formato:
	try {
	sentencias;
	} catch (ExceptionClass e) {
	sentencias;
	}

Prefijos.

En adición a las convenciones listadas, cada vez que se escriba un identificador debe llevar un prefijo a su nombre, para especificar su tipo y membresía,. A continuación se muestra una lista de los principales prefijos utilizados en este proyecto:

java.sql.*	
Instancias de clases comunes	*prefijo de instancias*
Date	dt
Time	tim
Timestamp	tis
ResultSet	rs
ResultSetMetaData	rsmd
Statement	stmt
PreparedStatement	pstmt
Connection	con
CallableStatement	cstmt
DatabaseMetaData	dbmd

java.lang.*	
Instancias de clases comunes	*prefijo de instancias*
Object	obj
String	str
StringBuffer	strb
Exception	ex
Throwable	thrw
Thread	thrd

Prefijos.

java.util.*	
Instancias de clases comunes	*prefijo de instancias*
ArrayList	al
Vector	v
HashSet	hst
TreeSet	tst
HashMap	hmap
TreeMap	tmap
HashTable	htbl
Properties	props
GregorianCalendar	gclr
Date	udt
Random	ran
Timer	tmr
Properties	props
Iterator	it
StrintTokenizer	strTkn
Enumeration	enum

java.awt.*	
Instancias de clases comunes	*prefijo de instancias*
Color	clr
Font	fnt
FontMetrics	fntm
Graphics	g
Graphics2D	g2d
Image	img
MediaTracker	mt
FlowLayout	fly
GridLayout	gly
GridBagLayout	gbly
GridBagConstraints	gbc
BorderLayout	bly
CardLayout	cly
Event	ev

java.swing.*	
instancias de clases comunes	*prefijo de instancias*
Action	actn
JButton	btn
JCheckBox	chb
JComboBox	cbx
JLabel	lbl
JList	list
JMenu	mn
JMenuBar	mnb
JMenuItem	mni
JOptionPane	oppn
JPasswordField	pwf
JPopupMenu	pmn
JProgressBar	pbar
JRadioButton	rbtn
JScrollBar	scrl
JSeparator	sptr

java.swing.*	
instancias de clases comunes	*prefijo de instancias*
JTable	tbl
JTextArea	txa
JTextField	txf
JTextPane	txp
JToggleButton	tbtn
JToolBar	tb
JToolTip	tt
JTree	tr
JDialog	di
JFrame	frm
JPanel	pnl
JScrollPane	scrlpn
JSplitPane	splpn
JTabbedPane	tbpn
JWindow	wnd

Se utilizan prefijos especiales para determinar la membresía de cada variable, es decir, a quien pertenecen, esos prefijos anteceden al prefijo propio del tipo de la variable:

Variables	
membresía	**prefijo del prefijo**
de clase	s
de instancia	
local	(ninguno)

Prefijos.

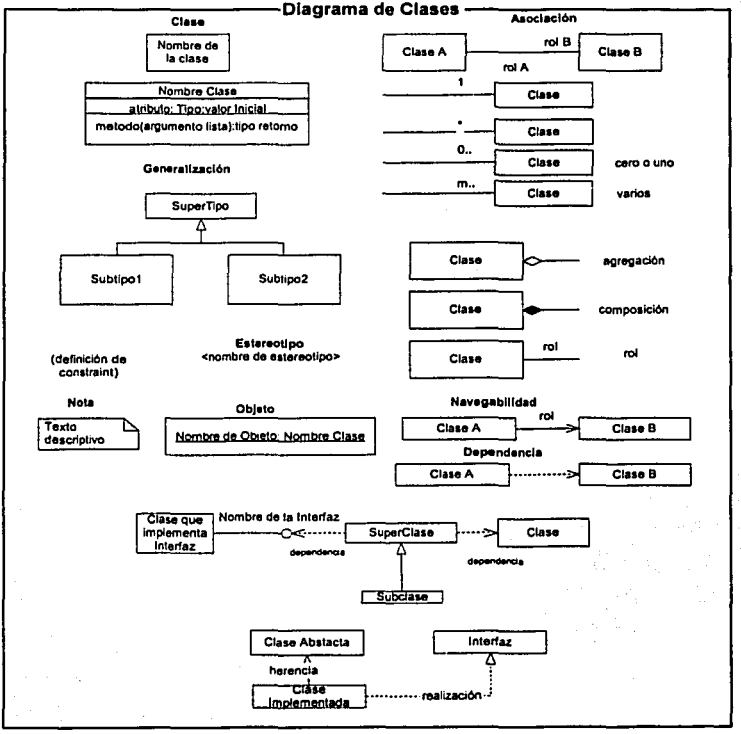
Primitivos y Clases envolventes.			
tipo primitivo	**prefijo**	**clase envolvente en java.lang**	**prefijo**
byte	y	Byte	by
short	s	Short	sh
int	i	Integer	int
long	l	Long	ln
float	f	Float	fl
double	d	Double	dl
boolean	b	Boolean	bl
char	c	Character	ch
		Number	num

Mapeo de tipos de datos de SQL a tipos Java.

La siguiente tabla resume los tipos de datos compatibles entre SQL y Java, util para adecuar aplicaciones que envían y reciben datos desde y hacia bases de datos.

Tipos de Dato SQL	Tipos de Dato Java	
	Primitivos	Clases Envolventes
CHARACTER		String
VARCHAR		String
LONGVARCHAR		String
NUMERIC		java.math.BigDecimal
DECIMAL		java.math.BigDecimal
BIT	boolean	Boolean
TINYINT	byte	Integer
SMALLINT	short	Integer
INTEGER	int	Integer
BIGINT	long	Long
REAL	float	Float
FLOAT	double	Double
DOUBLE PRECISION	double	Double
BINARY		byte[]
VARBINARY		byte[]
LONGVARBINARY		byte[]
DATE		java.sql.Date
TIME		java.sql.Time
TIMESTAMP		java.sql.Timestamp

Convenciones de notación para diagramas UML.



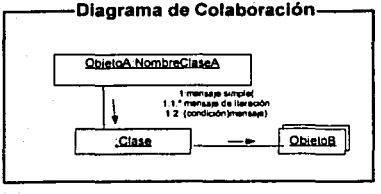


Diagrama de Secuencia

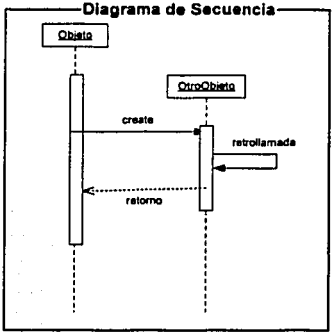


Diagrama de Casos de Uso

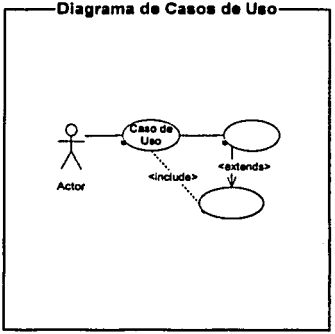


Diagrama de Estado

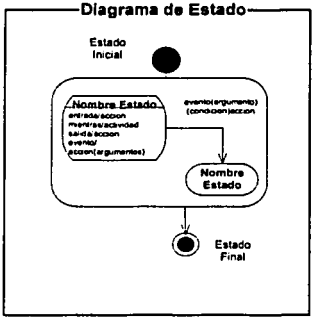


Diagrama de Actividad

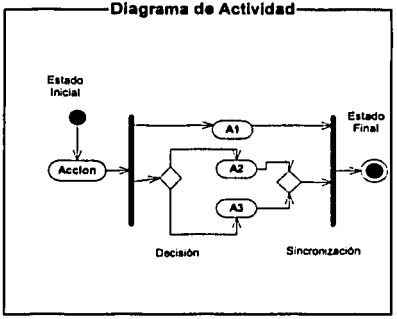


Diagrama de Distribución

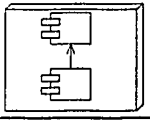


Diagrama de Componentes

Índice de Tablas e Ilustraciones

INDICE DE TABLAS E ILUSTRACIONES

Referencias

REFERENCIAS

Bibliograficas.

BOOCH ,G.;JACOBSON, I. **The UML Specificacionl Documents**. Editorial Rational Software Corp. Estados Unidos, 1997.

BOOCH, G.; RUMBAUGH, J.; JACOBSON, I. **The Unified Modeling Languaje User Guide**. Editorial Adison-Wesley, Estados Unidos, 1999.

BOOCH ,G. **Object-Oriented analysis and Desing**. Editorial Benajmin/Cummings. Estados Unidos, 1994.

FROUFE, Agustín. **Java 2 Manuel de Usuario y Tutorial**, Editoral Alfaomega, 2ª Edición,2000.

JACOBSON I. **Object-Oriented Software Engineering: A Use Case Driven Approach.** Editorial Addion-Wesley, 1992.

LARMAN, Craig.**UML y Patrones**; Prentice Hall, 1ª edición, México, 1999.

MARINESCU,Floyd. **EJB Design Patterns**. Editorial Wiley,Canadá, 2002.

PRESSMAN S, Roger. **Ingeniería del Software, un enfoque práctico.** Editorial McGraw-Hill, 4ª edición,México 1997.

MONSON-HAEFEL, Richard. **Enterprise JavaBeans.** Editorial O'Reilly, 2001.

QUATRANI,Terry.**Visual Modeling With Rational Rose 2000 and UML.** Editorial Addison Wesley,Estados Unidos, 2000.

RUMBAUGH , J. **Object Oriented Modelling and Design**. Editorial Prentice-Hall, Estaods Unidos,1991.

ROMAN, Ed. **Masterig Entreprise Java Beans**. Editorial Wiley Computer Publishing, Canadá, 2002.

TRACZ'S ,Will. **Confessions of a Used Program Salesman: Institutionalizing Software Reuse** .Addison-Wesley.

Hemerograficas.

ANDREW Burger, Ananova. SIGGRAPH. "*VIP's: Virtually Invented People*". ©ACM/SIGGRPAH Estados Unidos, Agosto 2001.

DOEGNES, Meter K. SIGGRAPH. "*Computer Games and Viz, if you can't beat them, join them*". ©ACM/SIGGRPAH Estados Unidos, Agosto 2001.

ENTIS, Glenn. SIGGRAPH Conference abstracts and applications. "*Games: the dominant medium of the future*". Estados Unidos 2002.

HERZ , J.C. SIGGRAPH. "*Video Game Play and Design: Procedural Directions*". ©ACM/SIGGRPAH Estados Unidos, Agosto 2001.

IGDA. *Online Games Whitepaper 2nd Edition*. ©IGDA, 2003.

KUSIC, Kirk. ACM Queue tomorrow's computing today. "*A Conversation with Adam Bosworth*". Columen 1. No. 1, Marzo 2003.

NUSSBAM, Miguel, *et al.* Revista Ciencia al Dia Internacional "*Diseño, Desarrollo y Evaluación de Videojuegos Portátiles Educativos y Autorregulados*". Septiembre 2001.

SUHAYA, George. SIGGRAPH. "*Game Stories: Simulation, Narrative, Addiction*". ©ACM/SIGGRPAH Estados Unidos, Agosto 2001.

Referencias de Red.

ABDOH, Akira; Nash Simon.*RMI over IIOP*. Diciembre1999; © JavaWorld. http://www.javaworld.com/javaworld/jw-12-1999/jw-12-iiop.html

ADOLPH, Steven *Whatever Happened to Reuse*?. Copyright © Gamasutra; http://www.gamasutra.com/features/19991213/adolph_01.htm

AGULLO, Soliveres Pedro. *Desarrollo cliente servidor*.Revista Profesional para Programadores.
http://www.etv.es/USERS/pagullo/arti/csbr/csbr.htm

AHUNA ,Cindy.*Online Game communities are social in nature*. 2001.
http://switch.sjsu.edu/v7n1/articles/cindy02.html

AMORY ,Alan.*Computer Games as a Learning Resource*.1998.
http://www.und.ac.za/und/biology/staff/amoy/edmedia98.html

ANDERSON Eve. *Arquitectura de sistemas distribuidos*.
http://www.galileo.edu/wp/presentation-html-download?presentation_id=2213.

ASTOR ENTERPRISE. *Tipos de aplicaciones*.
http://www.astortango.com.ar/Doc/Articulos/Tipos%20De%20Aplicaciones/Tipos_De_Aplicaciones.htm

BRYCE, Jo. *In The Game – In the Flow: Presence en Public Computer Gaming*. Marzo 2001;
http://www.digiplay.org.uk/Game.php

BARBERO, Angel .*Introducción a XML*. 1999.
http://www.openresources.com/es/magazine/xml-tutorial.

BEL Puchol, *Antonio. Apuntes Lenguaje Java*.España;2001.
http://www.arrakis.es/~abelp/ApuntesJava/IntroduccionAPI.htm

BELLAS Permuy , Fernando. *Introducción a la Orientación a Objetos*. Universida de Coruña, España.
http://www.tic.udc.es/~fbellas/teaching/joo/IOO.pdf.

BUADES, Gabriel. *Sistemas Distribuidos: ingeniería del Software III*. Enero 1999.
http://dmi.uib.es/~bbuades/sistdistr/sistdistr.ppt

CANO Araceli. *Piratería gana la partida al negocio de videojuegos*. ©El Financiero, Julio 2001;
http://www.videojuegosamazing.com/pirateria.htm, Araceli Cano, Martes, 31 de julio de 2001

CERVERA Paz, Angel. *El modelo de Mccall como aplicación de la calidad a la revisión del software de gestión empresarial.* Depto. Lenguajes y Sistemas Informáticos. Universidad Cádiz.
http://www.monografias.com/trabajos5/call/call.shtml .

COMPUWARE. *TrueTime Java Edition Preview.*©CompuWare;
http://www.compuware.com/products/devpartner/previews/truetime/ttjava/index.htm

DE LA CRUZ Salas ,Luis; CABRERA FLORES ,Eduardo C.*Introducción a la Programación Orientada a Objetos.* Noviembre 2000.
http://www.mcc.unam.mx/~lmd/Documentos/cplusplus/notas/node28.html

ETXEBERRIA, Félix. *Videojuegos y Educación.*Universidad del País Vasco;
http://www.quadernsdigitals.net/articles/quadernsdigitals/quaderns12/q12videojuegos.html

El lenguaje Java, Interpretado y Dinámico;México; 2002
http://www.linti.unlp.edu.ar/catedras/Laboratorio/Teorias/clase1.pdf.

FIGUEROA Pablo. *Metodología de desarrollo de software Orientado por Objetos.* Julio 1999.
http://www.cs.ualberta.ca/~pfiguero/soo/metod/

FONTELA, Carlos.*Programación Orientada a Objetos y Temas a Fines.*Argentina,2002.
http://www.fi.uba.ar/materias/7507F/Clases/ApuntePOOFontela.pdf

GAME RESEARCH. *Education.* Enero 2002 ©game-research.com.
http://www.game-research.com/education.asp

GARCÍA ZAVALA, Angel.*Orientación a Objetos* . México 2001.
http://paidoteca.dgsca.unam.mx/neopaidoteca/cursos/becas-java/apuntes.html

Guía de Patrones de Diseño.© Teleprogramadores, España
http://webs.teleprogramadores.com/patrones/

GÓNZALEZ ORDAS, Javier. *Desarrollo de una herramienta CASE Orientada a Agentes en Java.*Octubre 1999.
http://turing.gsi.dit.upm.es/~jgo/doc/desarrollo.html

GROS SALVAT, Begoña. *La dimensión socioeducativa de los juegos.* Revista Electrónica de Tecnología Educativa, Junio 2000.
http://edutec.rediris.es/Revelec2/Revelec12/gros.html

HAIM. *Optimizations Corner: Sorry... But Size Does Count!.* Copyright © 2000-2001 CMP Media Inc.
http://www.gamasutra.com/features/19991102/barad_01.htm

HERMAN, Leonard; KENT ,Steven; ET AL. *VideoGames History*; ©Gamespot.com 1999-2002;
http://gamespot.com/gamespot/features/video/hov/

HOPSON, John. *Behavioral Game Design.* 2002. Copyright © Gamasutra.com.
http://www.gamasutra.com/features/20010427/hopson_01.htm

HOWLAND, Geoff. *10 Things gamers need to know about game development.* 2003. Copyright © Lupine Games.
http://www.lupinegames.com/articles/10gamerknow.html

HOWSTUFFWORKS.COM; *How Videogames Work*; ©Howstuffworks.com.

http://www.howstuffworks.com/video-game1.htm

INEI.Arquitecturas., © Instituto Nacional de Estadística e Informática, Perú 1997.
http://www.inei.gob.pe/cpi-mapa/bancopub/libfree/lib616/index.htm

INTERWARE. *Para Compender Java: Una Historia* © Interware, Febrero 2000.
http://www.interware.com.mx/secciones/java/iwejava_historia04.html

KRAMER, Wolfgang. *What Is a Game?* 2002. Copyright © The Games Journal.
http://www.thegamesjournal.com/articles/WhatIsaGame.shtml

LARAMÉE, Francois-Dominique. *The Game Industry and the Economics of Failure*. 2000. Copyright © Gamedev.net
http://www.gamedev.net/reference/articles/article867.asp

LAURENT (laurent.roucairol@wanadoo.fr). *Le jeu online: son histoire, le pour, le contre.* ©Grospixels.com 2002.
http://www.grospixels.com/site/online.html

MARONEY, Kevin. *My Entire waking life*.Copyright © The Games Journal.
http://www.thegamesjournal.com/articles/MyEntireWakingLife.shtml

MARSELAS, Herb. *Profiling, Data Analysis, Scalability, and Magic Numbers, Part 1: Meeting the Minimum Requirements for Age of Empires II: The Age of Kings*. Copyright © 2000-2001 CMP Media Inc.
http://www.gamasutra.com/features/20000809/marselas_01.htm

MCCONNELL, Steve. *Software Engineering is not Computer Science*. Copyright Gamasutra.com.
http://www.gamasutra.com/features/19991216/mcconnell_01.htm

MOLYNEUX, Peter; PERRY ,Dave, ET AL. *Computer And Video Games Com E Of Age, A National Conference To Explore The Current State Of An Emerging Entertainment Medium.* Program in Comparative Media Studies, MIT 2000.
http://web.mit.edu/cms/games/future.html

MULLIGAN, Jessica. *History of Online Games*. Imaginary Networks 2000 ©Jessica Mulligan.
http://imaginaryrealities.imaginary.com/volume3/issue2/history.html

PC POWERPLAY. *A Brief History of PC Gaming*. ©PCPowerPlay.com 2002.
http://www.pcpowerplay.com.au/retro_content.cfm?id=172

PETTY, Mike. *Serious Gaming*. 2002. Copyright © The Games Journal.
http://www.thegamesjournal.com/articles/SeriousGaming.shtml

PONS, Juan de Pablos. *Las Tecnologías de la información y la comunicación: Un punto de vista educativo*.1999.
http://www.ucm.es/info/multidoc/multidoc/revista/num8/jpablos.html

Programación II: Programación Orientada a Objetos, 2002.
http://jeee.udistrital.edu.co/concurso/programacionII/Programacion2/html/c27.html

RODRIGUEA, Alberto .Revista FoxPress; "Diseño de aplicaciones Three Tier". 1997.
http://www.fpress.com/revista/Num9711/Nov97.htm

SADOSKI , Darleen. *Client/Server Software Architectures – An Overview.* Copyright © 2000 Carnegie Mellon University.
http://www.sei.cmu.edu/str/descriptions/clientserver_body.html

SAINT-JEAN, Felipe. *Introducción a EJB.*
http://www.elcod.cl/publicaciones/IntroductionEJB/IntroductionEJB.html .

SHIN. *Informe detallado de las Ventajas de las aplicaciones XML para la empresa.* © InvSight 2003.
http://www.holisticas.com/full_ventajasXML.htm

SUHAYDA, George. *Video Game Play and Design: Procedural Directions.* ©ACM/SIGGRPAH 2001.
www.siggraph.org/s2001/conference/panels/panels1.html

SUN. TRADUCTOR PALOS, JUAN ANTONIO; Java Beans Enterpise;1999-2003.
http://programacion.com/java/tutorial/javabeans/2.

SUN; TRADUCTOR PALOS JUAN ANTONIO; Tutorial de programación en java. ,1999-2003.
http://programacion.com/java.

SUN. TRADUCTOR : PALOS, JUAN ANTONIO; Introducción a XML ;1999-2003..
http://programacion.com/java/tutorial/apis_xml/1/

STAMATOPOLOUS, Steve. *A History of Gaming spotlight.* ©ConsumerREVIEW 2002.
http://www.pcgamereview.com/spotlight/History/

STIBBE, Mathews. *A Brief History of ComputerGames.* ©Matthew Stibbe, 1999-2002.
http://www.stibbe.net/History/History.htm

SWEENY, Tim. *Unreal Networking Architecture.* Copyright © Unreal.com
http://unreal.epicgames.com/Network.htm

THOMASSON, Michael. *Lineage of Electronic Entertainment.*
http://www.gooddealgames.com/articles/lineage.html

SIKORA, Drew. *You Got Game.* 2001.Copyright © Gamedev.net
http://www.gamedev.net/reference/design/features/gotgame1/

WROBLEWSKI, Monica. *Quick Facts About Video Game Consoles and Software.* ©IDSA.
http://www.idsa.com/consolefacts.html

YEE , Nicholas. Facets: *5 Motivations for Why People Play MMORPG's.* 2002.
http://www.nickyee.com/facets/home.html

Glosario

GLOSARIO

Acceso remoto.- Solicitud de datos por medio de un cliente a un servidor.

Actor.- No son parte del sistema, si no que representan roles que un usuario puede jugar, por lo que un actor puede ser un ser humano, una máquina u otro sistema.

Ambiente.- Contiene las colecciones de ítems y personajes relacionados con una casilla. Representa la información variable de una casilla.

Análisis.- Fase del ciclo de vida de un sistema en la que se conocen las necesidades de la organización y se comprende a fondo el problema identificado, revisando sus antecedentes, requisitos específicos y contexto, para comenzar a dar forma a los conceptos de la solución sistematizada que se propondrá.

API.- Aplication Programming Interface. Conjunto de reglas de programación que determinan como una aplicación debe acceder a un servicio.

Applet .- Aplicación de Java que se despliega en un navegador web.

Artefactos.- Accesorios que complementan el funcionamiento de un programa o proceso.

Arquitectura.- Se refiere a la estructura general de un procesador, sistema operativo, computadora, línea de sistemas, etc.

Arquitectura distribuida.- Conjunto de componentes diseminados en una red que, como un todo, responden a un requerimiento.

Atributo. Característica descriptiva de un objeto.

AWT.- Abstract Windows Toolkit. Proporciona el entorno base para todas las clases de manipulación de la interfaz gráfica del usuario.

Bean. Unidad de software reutilizables y auto-contenidas que pueden unirse para formar aplicaciones más complejas.

Beans de entidad.- Clase de EJB que representan al objeto de negocio persistente, objetos de negocio cuyos datos se almacenan en una base de datos, es decir, datos del negocio.

Beans de sesión.- Clase de EJB que realizan un servicio, utilizan a los beans de entidad para realizar sus tareas.

BMP.- Persistencia Manejada por el Bean. Se refiere a los beans de entidad.

Browser.- Navegador de Internet que se utiliza para desplegar el contenido de páginas web.

ByteCode.- Código de datos que utiliza Java para poder hacer código portable.

Caballo de Troya, aplicación. Aplicación descargada generalmente por medio un virus, un correo electrónico el cual tiene código que provoca daño o robo de datos.

Canal.- Medio por el cual viajan los datos a través de la red.

Capa de Datos.- Capa en la que se define la arquitectura de la base de datos así como los componentes que harán posible el acceso a la misma.

Capa de Negocio.- Capa en la que se definen los componentes que resolverán las reglas y funciones de negocio.

Capa de Presentación.- Capa en la que se definen los componentes que comprenderá la interfaz del usuario.

Casilla.- Información que describe al espacio mínimo casilla del tablero y sus características. Pueden contener tanto ítems como personajes.

Caso de uso.- Flujo de eventos completo y con significado para el usuario. Modela el diálogo entre actores y el sistema.

Clase.- Agrupación de objetos con características y comportamientos iguales.

Cliente (Arquitectura Cliente-Servidor).- Programas con los que el usuario interactúa de forma gráfica.

Cliente (Aplicación).- Programa que llama a otro que se encuentre en un servidor.

CMC.- Comunicación Mediada por Computadora, formación de una comunidad mediante la convivencia de un videojuego.

CMP.- Persistencia Manejada por el Contenedor. Se refiere a los beans de entidad.

Colector de Basura.- Utilidad de Java que recoge las instancias de objetos que no se utilizan.

Collection.- conjunto de clases para la manipulación de conjuntos de objetos.

Comportamiento. Manera en que reacciona el objeto ante mensajes recibidos.

Concurrencia.- Solicitud de varios elementos a la vez de un mismo objeto.

Conocimiento Mágico.- Conjunto de magias válidas en el juego.

Conocimiento Mágico (Clase).-Conjunto de magias que el personaje conoce y es capaz de utilizar.

Construcción.- Fase del ciclo de vida de sistemas en la que se traducen los esquemas y conceptos generados en código, planeando antes la forma en que se atacará la programación de los módulos identificados.

Contenedor.- Entorno especial donde se ejecutan los EJB, contiene y maneja al bean de igual forma que un Servidor Web Java contiene un servlet.

Contrato.- Documentación para los diagramas de interacción que ayuda a la lectura de los mismos.

CORBA.- Common Object Request Broker Arquitecture. Modelo de programación de objetos distribuidos que soporta varios lenguajes, principalmente C. Tecnología de objetos distribuidos milti-lenguaje y multi-plataforma.

Descriptor.- Archivo XML que utiliza el contenedor para encontrar las instrucciones de manejo de transacciones, persistencia y control de acceso.

Despliegue.- Fase incluida en la implementación en la que se evalúa la distribución del software en los equipos de la organización.

Diseño.- Fase del ciclo de vida de un sistema en la que se comienza la cimentación de los conceptos de análisis en elementos más concretos y cercanos a la construcción.

Diseño Orientado a Objetos.- Documento de diseño que incluye versiones detalladas de los modelos de objeto.

DTD.- Document Type Definition de XML. Describe al tipo de bean y las clases utilizadas para la interfaz remota, home y la clase del bean.

EJB .- Enterprise Java Beans. Define una arquitectura para el desarrollo y despliegue de aplicaciones basadas en objetos distribuidos transaccionales, software de componentes del lado del servidor.

Elemento.- Clase padre de todos aquellos objetos animados o inanimados del juego con los que es posible interactuar.

ENC.- Environment Naming Context es un sistema de nombrado especial controlado por el contenedor y los beans acceden a él utilizando JNDI.

Encapsulamiento.- Información de un objeto empaquetada, reutilizable como una especificación o componente de un programa.

Estado. Propiedad de un objeto que define su situación en un momento determinado.

Estado del personaje.- Información que define la situación del personaje con respecto a los niveles que posee en atributos de variables relacionados con las acciones del juego.

Estereotipos.- Mecanismos para extender el significado de los elementos UML.

Excepción.- Evento que ocurre durante la ejecución de un programa y detiene el flujo normal de la secuencia de instrucciones de ese programa.

Expresión.- Utilizadas para calcular y asignar valores a variables y control de flujo de un programa.

Factibilidad.-Disponibilidad que se tiene de recursos para llevar a cabo los objetivos y metas planteados en términos operativo, económico y técnico.

FTP.- File Transfer Protocol, protocolo de transferencia de archivos a través de la red.

Herencia.- Propiedad de las clases de utilizar los métodos de otra como padre de ésta.

HTTP.- Hyper Text Transfer Protocol, protocolo de transferencia de hiper texto a través de la red.

Identidad. Propiedad del objeto que lo identifica de los demás objetos.

Imagen del juego.- Vista continua que tiene el usuario del juego, es decir la pantalla principal de juego.

Implementación.- Fase del ciclo de vida de un sistema que comprende la instalación y puesta en marcha siguiendo la arquitectura elegida.

Implementación de un método .- Especificación del comportamiento de un método en código.

Instancia .- Relación que existe entre Clase y objeto, objeto particular de la clase.

Interbloqueo.- Solicitudes de los usuarios a un mismo registro o datos, de esta forma, se evita que la misma información sea alterada simultáneamente, dando lugar con esto a inconsistencia o mal funcionamiento del sistema.

Interfaz.- Clase que proporciona un mecanismo para abstraer los métodos a un nivel superior, ya que es como un molde donde sólo permite declarar nombre de métodos, lista de argumentos, los cuales sobrescribirán las clases que lo implementen.

Inventario.- Conjunto de ítems existentes en el juego.

Inventario (Clase).- Conjunto de ítems relacionados con el personaje, es decir, los objetos que porta actualmente el personaje.

Ítem.- Contiene la información que describe las características de un ítem en el juego.

J2EE.- Java 2 Enterprise

J2ME.- Java 2 Micro Edition.

J2SE.- Java 2 Standard Edition.

***JAR.**- Java Archive. Extensión del formato ZIP que permite empaquetar clases java compiladas para su ejecución*

JAVA.-Lenguaje de programación Orientado a Objetos creado por Sun Systems.

JAXP.- Java API for XML Parsing

JAXB.- Java Arquitecture for XML Binding

JAXM.- Java Arquitecture for XML Messaging

JAXR.- Java API for XML Registries

JDBC.- Java Data Base Conectivity, API integrada en Java para el manejo de bases de datos mediante la ejecución de instrucciones en lenguaje estándar de acceso de base de datos.

JDK.- Java Development Kit. Colección de componentes básicos para ayudar a los usuarios de software a construir aplicaciones Java.

JMS.- Java Message Service. API que facilita escribir aplicaciones de negocios que envían y reciben, de manera asíncrona datos y eventos críticos.

JNDI.- Java Naming and Directory Interface, Interfaz de Nombres y Directorios de Java es una extensión estándar para acceder a sistemas de nombrado.

***JSP.**- Java Server Pages. Tecnología que permite mezclar HTML estático con HTML dinámico.*

JVM.- Java Virtual Machine, Máquina Virtual de Java necesaria para correr programas basados en Java.

Magia.- Contiene la información que describe las características de una magia en el juego.

Mantenimiento.- Fase incluida en la implementación en la que se evalúan los planes de mantenimiento a sistemas, ya sea preventivo o correctivo.

Mapeo.- Determinación de la forma en que la información será gestionada por un bean de entidad.
Metalenguaje. Lenguaje formal que emplea símbolos especiales utilizado para describir la sintaxis de un lenguaje de computación.

Método. Procedimientos que permiten manipular a los atributos

Metodología.- Serie de pasos o métodos utilizados para la solución de un problema que tiende a ser repetitivo.

Método de retrollamada.- Métodos que permiten ubicar al bean en el contexto de beans, la conexión con la base de datos dentro de un contenedor.

Middleware.- Herramienta o conjunto de herramientas que nos permiten gestionar y coordinar los mecanismos de comunicación independizando el servicio y la implementación del Sistema Operativo y protocolos de comunicación, permitiendo la convivencia de distintos servicios en una misma máquina.

MMORPG.- Massive Multiplayer Online RPG. Juego de Rol Masivo, Multijugador en Línea. Es aquel RPG jugado través de la red donde interactúan una gran cantidad de jugadores dentro de un mundo virtual.

Modelo Conceptual.- Modelo donde se presenta una realidad por medio de conceptos (sus atributos y asociaciones entre ellos) del dominio un problema.

Modelo de capas.- Modelo de desarrollo que divide a la aplicación en capas delimitadas por funcionamiento.

Multicapa.- Esquema de desarrollo de modelos Cliente-Servidor en donde hay varias capas de software con las responsabilidades bien delimitadas. El esquema más utilizado es el de tres capas donde son : datos, negocio y presentación.

Multi-plataforma.- Característica de las aplicaciones de ejecutarse en ambientes distintos sin necesidad de reescribir código. Característica de una aplicación la cual le permite ejecutarse en diferentes ambientes.

Objeto. Es cualquier cosa real o abstracta que conforma un sistema.Esta definido por un *estado* y un *comportamiento*

Objeto distribuido.- Clase de implementación real de un EJB para su distribución en red.

Objeto valor.- Representación de la información.

ORB.- Object Request Broker. Componente de software que permite que objetos CORBA operen entre sí, aún estando en distintas computadoras.

Orientación de Objetos. paradigma ,conjunto de disciplinas de ingeniería de software que facilitan el desarrollo de sistemas complejos a partir de componentes individuales. donde un sistema se descompone en objetos.

Partida.- Conjunto de personajes activos en el juego.

Plataforma.- Arquitectura de hardware. También se refiere al sistema operativo que se instale en el equipo.

Perfil del personaje.- Información para identificar las características de un personaje

Persistencia de objeto.- Cuando un objeto no se destruye y puede ser utilizado entre máquinas en un entorno distribuido se dice que el objeto es persistente.

Personaje.- Representación del usuario como ser ficticio dentro del juego.

Polimorfismo.- Propiedad de los métodos para mantener una respuesta de manera unificada, con la misma semántica pero con diferente implementación.

Portabilidad.- Se dice que hay código portable cuando se compila el código a un formato independientemente de la arquitectura de la máquina.

Puntero.- Referencia a locaciones de memoria.

RDBMS.- Servidor de administración de bases de datos relacionales. Lugar en donde residen todos los datos que son solicitados por la capa de negocio y la de presentación.

Registro.- Serie de datos almacenados correspondientes a la ocurrencia en una entidad.

Reglas del negocio.- Políticas de la empresa que son aplicadas a los procesos que realizan los programas o aplicaciones.

Reusabilidad.- Capacidad del software de tener componentes que distintas aplicaciones puedan utilizar. Posibilidad de reutilizar el código existente en una aplicación similar.

RMI.- Invocación Remota de Métodos, sistema de Java que permite a un objeto que se está ejecutando en una Máquina Virtual de Java, llamar a métodos de otro objeto que se encuentran en otra máquina virtual diferente.

RMI-IIOP.- RMI / InternetInter-ORB Protocol. Protocolo que combina características de RMI y permite la comunicación entre varios protocolos en la red.

RPG.- Acrónimo de Role Playing Game o Juego de Rol. Es aquel en el que el jugador toma la personalidad de un personaje artificial y lo desarrolla como parte de la acción de jugar.

RUP.- Rational Unified Process. Metólogia. Metodología propuesta por Rational. Conjunto de procesos que comprenden todas las actividades involucradas en el ciclo de vida del software.

Seguridad.- Factor que se toma en cuenta para salvaguardar la información de las entidades que no se encuentran autorizadas a verla. Medidas que se implementan para resguardar el producto del software.

Serialización.- Herramienta de Java que proporciona una solución para salvar objetos en archivos y transmitirlos a través de la red.

Servicio de Nombres.- Permite que un servidor publique una referencia a un objeto remoto asociándole un nombre jerárquico, y que el cliente obtenga la referencia a partir del nombre.

Servidor (Arquitectura Cliente-Servidor).- Proporcionan un servicio al Cliente, devolviéndole los resultados solicitados. Dispositivo de un sistema que resuelve las peticiones de otros elementos del sistema, llamados clientes.

Servidor de aplicaciones.- Servidor que ejecuta los programas de negocio en lugar del cliente, situándose éste y los datos empresariales y otras aplicaciones.

Servlet.- Programas que corren en un servidor Web y construyen páginas web. Son la respuesta de Java a los CGI.

Skeleton. Conjunto de clases que sirve como simulador en el servidor para recibir parámetros enviados por una aplicación cliente a través de la red en RMI.

SOAP.- Simple Object Access Protocol

SobreCarga.- Definición de un método cuyo nombre sea igual al del método de la clase que hereda, alerando la firma, cambiando los parámetros.

SobreEscritura.- Definición de un método cuya firma sea igual a la del método de la clase que hereda, dándole un comportamiento en particular.

Socket.- Clase de Java que permite la comunicación entre máquinas a través de la red.

Stub. Conjunto de clases que sirve como simulador en el cliente para enviar los parámetros a la red y recibe los resultados enviados por el servidor.

Subclase.- Clase que hereda de otra. También llamada clase "hija".

Superclase.- Clase de la cual heredan otras clases. También llamada clase "padre".

SWING.- Pertenece al API extendido. Conjunto extendido de clases para la configuración de la interfaz gráfica del usuario. Reemplaza parcialmente al AWT.

Tablero.- Conjunto de casillas que representan el mundo o ambiente en el juego.

TCP/IP.- Transmisión Control Protocol/Internet Protocol. Se trata de un estándar de comunicaciones muy extendido y de uso muy frecuente para software de red basado en Unís con protocolos Token-Ring y Ethernet,e ntre otros. Es conforme a los niveles 3 y 4 de los modelos OSI. Desarrollado originalmente para el Departamento de Defensa de Estados Unidos.

Telepresencia.- Conocimiento de los acontecimientos en un lugar debido a la intervención de medios electrónicos y de red.

Transacción.- Colección de acciones que hacen transformaciones consistentes de los estados de un sistema, preservando la consistencia del sistema, asegurando que la base de datos regresa a un estado consistente al fin de su ejecución.

UML.- Lenguaje de Modelado Unificado es el lenguaje que permite especificar, visualizar y construir los artefactos de los sistemas de software.

Valor de Retorno: valor que regresa un método al ejecutar sus sentencias.

Videojuego.- Nombre genérico con el que se conocen ciertos programas de carácter lúdico que pueden ser ejecutados en computadoras o en otros dispositivos, también de base informática, llamados consolas.

XML.- Extensible Markup Language es un estándar para representar datos independientemente del sistema.