Universidad de las Ciencias Informáticas

Facultad 5

“Propuesta de arquitectura para Juegos en Línea sobre plataforma web.”

Trabajo de diploma para optar por el Título de
Ingeniero en Ciencias Informáticas

Autor(es):

Anniery Martínez González

Hector Daniel Pagán Arias

Tutor(es):

Ing. Yerandi Marcheco Díaz.

Ing. Yausell Ruiz Marine.

Ciudad de la Habana

Julio, 2010

DECLARACIÓN DE AUTORÍA

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

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

Annierys Martínez González

_____

Firma del Autor

Hector Daniel Pagán Arias

_____

Firma del Autor

Ing. Yerandi Marcheco Díaz.

_____

Firma del Tutor

Ing. Yausell Ruiz Marine.

_____

Firma del Tutor

Datos de Contacto

Nombre y Apellidos: Yerandi Marcheco Díaz.

Edad: 25 años.

Ciudadanía: cubano.

Institución: Universidad de las Ciencias Informáticas.

Título: Ingeniero en Ciencias Informática.

Categoría Docente: Profesor Adiestrado.

e-mail: ymarcheco@uci.cu

Graduado en la Universidad de las Ciencias Informáticas, 2 años de experiencia.

Nombre y Apellidos: Yausell Ruiz Marine.

Edad: 25 años.

Ciudadanía: cubano.

Institución: Universidad de las Ciencias Informáticas.

Título: Ingeniero en Ciencias Informática.

Categoría Docente: Profesor Adiestrado.

e-mail: ymarine@uci.cu

Graduado en la Universidad de las Ciencias Informáticas, 2 años de experiencia.

De Annierys:

A mi mamá de todo corazón, por haber confiado en mí, aconsejarme y apoyarme en todo momento.

A mi hermanito Luis, por apoyarme, por sus consejos útiles en el momento adecuado, por su ánimo y su sonrisa siempre alentadoras, y ser un ejemplo siempre para mi, te quiero mucho.

A mis amigas de siempre, las muñes: Baby, Eve y Yele, por estar siempre ahí conmigo y permitirme aconsejarlas y tratarlas como si fueran mis hijas.

A mis abuelos, por siempre estar ahí para darme su apoyo y cariño.

A mi tío Andrés y a mi primo Pavel por haber ayudado a mí y a mi mamá cuando yo he estado aquí.

A mi tía Esther, Poso, mis primos Evelyn e Ismael por su apoyo estos 5 años de la carrera.

A mi tía Elizabet por toda su ayuda y cosejos.

A mi tía Nieves y Pipo y mis primos por su apoyo.

A mi novio Liuwis y a su familia por ayudarme tanto estos 2 últimos años y su apoyo incondicional en los momentos difíciles.

A mi cuñada Yoenia y a Blanca por la ayuda que le han dado a mi hermano y a mí en todo momento, del cual voy a estar siempre agradecida.

Y en general a toda mi familia y a todas las personas que a lo largo de la carrera me han ayudado.

De Hector:

A mi mamá que me dió la vida y desde entonces no ha dejado de apoyarme ni de alentarme a seguir adelante, por creer en mí y ayudarme a ser cada día mejor persona.

A mi papá por ayudarme a distinguir el camino correcto, por confiar en mí y por ser siempre mi mejor y más fiel amigo, por estar ahí cada vez que lo necesito.

A mi hermano por estar siempre a mi lado y por ayudarme cada vez que lo necesito.

A mi abuela, que aparte de ser mi segunda madre ha sido un ejemplo siempre para mí, a mis tías y tíos que me han apoyado ante cualquier dificultad.

A mis primos que son mis hermanos, y que siempre también me han alentado a seguir adelante, a los más pequeños les lego mi ejemplo.

A mi novia por su cariño y comprensión en todo momento. Por estar ahí cuando más la necesitaba.

A mis suegros por haber confiado en mí y apoyarme en cada momento.

A mis amigos y a todas esas personas que de manera incondicional me brindaron su apoyo y confianza. A ustedes que siempre creyeron en mí, les agradezco también este triunfo.

De Annierys:

A la memoria de mi papá, por todo el apoyo que me dio cuando yo decidí estudiar aquí en la UCI, por ser mi ejemplo y guía que siempre voy a tener presente.

A mi mamá y a mi hermano que son lo que más quiero en este mundo.

A toda mi familia por confiar en mí.

De Hector:

De manera especial a mi mamá y mi papá por ser cómplices de este logro. A ellos que me dieron la vida y han sabido sacrificarse para darme lo mejor, a ellos que son mi orgullo y ejemplo a seguir.

A mi hermano y mi abuela por ser parte de mi vida.

A toda mi familia por apoyarme y confiar en mí.

Resumen

En el presente trabajo se pretende diseñar una propuesta de Arquitectura de Software, para el desarrollo de aplicaciones de juegos en línea sobre la Web, que facilite una adecuada comprensión del sistema a sus desarrolladores y que al mismo tiempo contribuya a la construcción de juegos en línea con una alta aceptación. Para ello se realizó un amplio estudio acerca de los principales estilos arquitectónicos utilizados para el diseño de este tipo de aplicaciones, además se analizaron los distintos patrones de diseño y arquitectura, así como las herramientas, lenguaje de modelado y metodologías de desarrollo, se hizo también un estudio de los principales Gestores de Base de Datos utilizados en la actualidad para el desarrollo de aplicaciones de juegos en línea.

Luego del estudio realizado en esta investigación se hizo una propuesta de arquitectura, justificando al mismo tiempo el uso de cada uno de los patrones de diseño, estilos arquitectónicos, herramientas, lenguaje de modelado y metodologías utilizadas para el diseño de la misma, y con el objetivo de lograr un mejor entendimiento de esta, se realizó la descripción de la arquitectura propuesta, donde quedaron bien definidas las metas y restricciones arquitectónicas, así como la representación arquitectónica a través el modelo 4 + 1 vista, para finalizar se realizó la evaluación de la arquitectura donde se utilizó el método ARID (Revisiones Activas para Diseños Intermedios), para esto se seleccionaron diferentes atributos de calidad con el objetivo de identificar los riesgos y fortalezas de esta propuesta.

Palabras Claves

Arquitectura de Software, Patrón, Estilo.

Índice:

Índice de figuras

Índice de tablas

Introducción

El creciente desarrollo de las tecnologías de la informática, ha incitado a que en los últimos años las aplicaciones web se hayan convertido en complejos sistemas, con interfaces de usuario cada vez más parecidas a las aplicaciones de escritorio, brindando servicio a diversos procesos de negocio de gran extensión, y así creándose sobre ellas requisitos muy estrictos de accesibilidad y respuesta.

El uso de aplicaciones web que permitan una interacción rápida y amigable con los clientes, es un tema de interés constante de las empresas, en busca de elevar los mercados, las ventas o el número de clientes, aprovechándose de sus ventajas. El aumento de la demanda de estas solicitudes ha crecido, así como las expectativas de los clientes, hecho que obliga a los desarrolladores a buscar nuevas soluciones.

Es tanto así el desarrollo en esta rama que es inevitable en cualquier sector de la vida socio-económica de un país no pensar en insertarse dentro de este mundo, por supuesto que Cuba no se ha mantenido al margen de esta situación, por lo que ha favorecido la creación de estructuras que permitan una "informatización" del país con vistas al desarrollo productivo y social que esto pueda reportar. En este marco de ideas revolucionarias se inserta la construcción de la Universidad de las Ciencias Informáticas (UCI) que vincula la formación, producción e investigación en su seno. La UCI representa en estos momentos la estructura educacional de mayor magnitud en Cuba. Ello implica que maneje un elevado número de recursos tanto materiales como humanos. Si se unen estas características estructurales a las metas internas en cuanto al plan de enseñanza y producción, así como al propósito de ser "la ciudad digital de Cuba", surge la necesidad de ejecución de numerosos proyectos informáticos que tributen a dicho fin.

La Facultad 5 de la UCI uniéndose a este marco de trabajo, consecuente a la necesidad de la producción de software se ha propuesto desarrollar varios proyectos productivos, entre los que se encuentra el proyecto Juegos Online (juegos en línea), el cual se dedica al desarrollo de juegos en línea **basado en** aplicaciones Web, debido a que el desarrollo de estos se torna complicado en cuanto a la falta de una estructura que lo soporte, lo que atrasa el trabajo del desarrollador, al no tener definido donde irán los componentes o módulos, esto hace que el código escrito en muchas ocasiones no se reutilice ni quede en

forma óptima, además de quedar totalmente desorganizado. Por lo que el proyecto carece de una arquitectura que soporte el desarrollo de este tipo de juegos.

Teniendo en cuenta la situación antes definida se plantea el siguiente **problema científico**: ¿Cómo lograr una arquitectura que soporte el desarrollo de juegos en línea?

De ahí se define como el **objeto de estudio:** Las Arquitecturas para aplicaciones Web, y específicamente la investigación se centrará en las Arquitecturas para las aplicaciones Web, relacionadas con juegos que sería **el campo de acción**.

Como consecuencia de lo antes expuesto se tiene que el **objetivo de la investigación es:** Proponer una arquitectura que soporte el desarrollo de juegos en línea.

Para dar cumplimiento al objetivo planteado se definen las siguientes tareas de investigación:

1- Realización de un estudio de las aplicaciones web para elaborar el estado del arte de la investigación.

2- Caracterización de las diferentes tecnologías empleadas a nivel mundial para el desarrollo de juegos en línea.

3- Selección de las características distintivas de la arquitectura a proponer.

4- Descripción de los componentes de la arquitectura y determinar los que serán empleados en el desarrollo.

5- Identificación de las tecnologías y métodos a usar para dar respuesta al problema planteado.

6- Evaluación de la Arquitectura Propuesta para identificar riesgos y fortalezas durante el diseño de esta.

1

Capítulo 1: Fundamentación Teórica

1.1 Introducción

Este capítulo comprenderá el estado del arte del tema tratado, así como definiciones importantes de arquitectura de software. Se analizarán además los distintos patrones y estilos arquitectónicos. También se realizará un estudio y fundamentación de las tecnologías, metodologías, y herramientas más usadas actualmente para el desarrollo de aplicaciones web.

1.2 Arquitectura de Software

1.2.1 Inicios de la arquitectura de software

La Arquitectura de Software (AS) surgió en 1968, cuando Edsgar Dijkstra, de la Universidad Tecnológica de Eindhoven en Holanda y Premio *Turing* 1972, propuso que se establezca una estructuración correcta de los sistemas de software antes de lanzarse a programar, escribiendo código de cualquier manera [1].

En 1975, Frederick Phillips Brooks Jr., diseñador del sistema operativo OS/360 y Premio *Turing* 2000, utilizaba el concepto de arquitectura del sistema para designar "la especificación completa y detallada de la interfaz de usuario" y consideraba que el arquitecto es un agente del usuario, igual que lo es quien diseña su casa [2] , también distinguía entre arquitectura e implementación, mientras aquella decía qué hacer, la implementación se ocupa de cómo.

La Arquitectura de Software siguió tomando auge con el paso de los años pero no es hasta 1992 que la Arquitectura de Software se define como disciplina de software en la publicación "*Foundations for the study of software architecture*" escrito por Dewayne Perry, Alexander Wolf donde planteaban [3]:

"La década de 1990, creemos, será la década de la arquitectura de software…"

1.2.2 Definiciones de arquitectura de software

Primeramente hay que partir de las grandes definiciones que existen de la arquitectura como:

"La arquitectura de software es, a grandes rasgos, una vista del sistema que incluye los componentes principales del mismo, la conducta de esos componentes según se le percibe desde el resto del sistema y las formas en que los componentes interactúan y se coordinan para alcanzar la misión del sistema. La vista arquitectónica es una vista abstracta, aportando el más alto nivel de comprensión y la supresión o diferimiento del detalle inherente a la mayor parte de las abstracciones" [1].

Con esta definición el autor se refiere a que la arquitectura es la estructura más importante de un sistema, puesto que está encargada de la estructura y relaciones de los componentes de un sistema, la

arquitectura representa una vista de abstracción del sistema como un todo. A la hora de realizar una arquitectura hay que tener en cuenta el funcionamiento y la interacción de los componentes, por lo que la estructura debe ser diseñada de acuerdo con los requisitos funcionales y no funcionales, para poder proporcionar una buena confiabilidad, seguridad, integridad, flexibilidad, mantenibilidad y escalabilidad del sistema, además la arquitectura de software establece los fundamentos para que analistas, diseñadores, programadores, etc. trabajen en una línea común que permita alcanzar los objetivos del sistema de información, cubriendo todas las necesidades.

Algunas arquitecturas son más recomendables de implementar con ciertas tecnologías mientras que otras tecnologías no son aptas para determinadas arquitecturas. Por ejemplo, no es viable emplear una arquitectura de software de tres capas para implementar sistemas en tiempo real.

La arquitectura de software además define, de manera abstracta, los componentes que llevan a cabo alguna tarea de computación, sus interfaces y la comunicación entre ellos. Toda arquitectura debe ser implementada en una arquitectura física, que consiste simplemente en determinar qué computadora tendrá asignada cada tarea.

Otras de las definiciones de arquitectura es que dirige el desarrollo de los sistemas software y contribuye a que estos se lleven a cabo en los límites establecidos de costos y tiempo, y fundamentalmente debe garantizar que se cumpla con los requisitos funcionales y no funcionales de los usuarios. Se podría decir además que la arquitectura de software es el resultado del trabajo durante todo el ciclo de vida del proyecto, y principalmente en las primeras iteraciones, de un grupo de trabajo encabezado por el arquitecto (o grupo de arquitectura, en dependencia de las dimensiones del proyecto).

La definición oficial de arquitectura del software que propone la IEEE [5] es:

“La arquitectura de software es la organización fundamental de un sistema encarnada en sus componentes, las relaciones entre ellos y el ambiente y los principios que orientan su diseño y evolución.”

La arquitectura del software tiene importantes decisiones en cuanto a:

- La organización del sistema software.

- Elementos estructurales del sistema
- La composición de los elementos estructurales.
- El estilo de arquitectura que guía esta organización.

Con todas estas definiciones antes mencionadas, se llega a la conclusión de que la arquitectura de software no es más que una guía para el desarrollo de aplicaciones, la cual representa la estructura principal en el desarrollo del software, sin tener en cuenta el tipo de software que se quiera desarrollar. Por otra parte, se podría decir también que brinda soluciones mediante estructuras capaces de soportar todo tipo de problema, que pueda presentarse en el desarrollo de un determinado trabajo. Asegura que los requerimientos importantes puedan ser implementados y evaluados. Además, ayuda a la planificación de recursos y la asignación de tareas, debido a que el trabajo de desarrollo puede ser dividido a través de subsistemas, de ahí que los esfuerzos de desarrollo individual pueden realizarse en paralelo.

1.2.3 ¿Por qué es necesaria la Arquitectura de Software?

A continuación se sintetizan las virtudes de la Arquitectura de Software (AS) demostrando las opiniones convergentes de los expertos [14]:

- **Comunicación mutua**. La AS representa un alto nivel de abstracción común que la mayoría de los participantes, si no todos, pueden usar como base para crear entendimiento mutuo, formar consenso y comunicarse entre sí. En sus mejores expresiones, la descripción arquitectónica expone las restricciones de alto nivel sobre el diseño del sistema, así como la justificación de decisiones arquitectónicas fundamentales.
- **Decisiones tempranas de diseño**. La AS representa la encarnación de las decisiones de diseño más tempranas sobre un sistema, y esos vínculos tempranos tienen un peso fuera de toda proporción en su gravedad individual con respecto al desarrollo restante del sistema, su servicio en el despliegue y su vida de mantenimiento.
- **Reutilización, o abstracción transferible de un sistema**. La AS encarna un modelo relativamente pequeño, intelectualmente tratable, de la forma en que un sistema se estructura y

sus componentes se entienden entre sí, este modelo es transferible a través de sistemas, en particular, se puede aplicar a otros sistemas que exhiben requerimientos parecidos y puede promover reutilización en gran escala. El diseño arquitectónico soporta reutilización de grandes componentes o incluso de *frameworks* (marco de trabajo) en el que se pueden integrar componentes.

- **Evolución.** La AS puede exponer las dimensiones a lo largo de las cuales puede esperarse que evolucione un sistema. Haciendo explícitas estas "paredes" perdurables, quienes mantienen un sistema pueden comprender mejor las ramificaciones de los cambios y estimar con mayor precisión los costos de las modificaciones. Esas delimitaciones ayudan también a establecer mecanismos de conexión que permiten manejar requerimientos cambiantes de interoperabilidad, prototipado y reutilización.

- **Administración**. La experiencia demuestra que los proyectos exitosos consideran una arquitectura viable como un logro clave del proceso de desarrollo industrial. La evaluación crítica de una arquitectura conduce típicamente a una comprensión más clara de los requerimientos, las estrategias de implementación y los riegos potenciales.

De forma general estos aspectos analizados anteriormente, demuestran en gran medida la necesidad de la AS para el desarrollo de cualquier tipo de software, primeramente porque representa el esqueleto principal de este, y luego porque sin ella se dificultarían otros aspectos importantes que aunque no tengan que ver con la distribución de las partes de un software, de alguna forma u otra deben estar presente, un ejemplo de esto pudiera ser la organización o comunicación entre las personas involucradas, etc.

1.3 Estilos arquitectónicos

Las soluciones de diseño arquitectónicas que son comunes y reusables a lo largo de años de experiencia se han ido agrupando en lo que más tarde se les llamó estilos. El éxito del diseño de la arquitectura de software depende de los estilos que se decida utilizar para el desarrollo de la misma. Los estilos expresan la arquitectura en el sentido más formal y teórico, describen entonces una clase de arquitectura, o piezas identificables de las arquitecturas empíricamente dadas. Esas piezas se encuentran repetidamente en la práctica, trasuntando la existencia de decisiones estructurales coherentes. Una vez que se han

identificado los estilos, es lógico y natural pensar en re-utilizarlos en situaciones semejantes que se presenten en el futuro [3].

Otras definiciones de estilos arquitectónicos por grandes desarrolladores de la ingeniería de software se muestran a continuación:

Según Shaw y Garlan definen estilo arquitectónico como:

*"*Una familia de sistemas de software en términos de un patrón de organización estructural, que define un vocabulario de componentes y tipos de conectores y un conjunto de restricciones de cómo pueden ser combinadas. Para muchos estilos puede existir uno o más modelos semánticos que especifiquen cómo determinar las propiedades generales del sistema partiendo de las propiedades de sus partes" [6].

*"*Una familia de sistemas de software en términos de su organización estructural. Expresa componentes y las relaciones entre estos, con las restricciones de su aplicación y la composición asociada, así como también las reglas para su construcción. Así mismo, se considera como un tipo particular de estructura fundamental para un sistema de software, conjuntamente con un método asociado que especifica cómo construirlo. Este incluye información acerca de cuándo usar la arquitectura que describe, sus invariantes y especializaciones, así como las consecuencias de su aplicación*"* [7].

Luego de haber visto algunas de las principales definiciones de estilos arquitectónicos, pasaron a analizar los estilos arquitectónicos más difundidos en la actualidad.

1.3.1 Estilos arquitectónicos más difundidos

Este epígrafe se centra en los estilos arquitectónicos más populares en la actualidad, así como los más usados en aplicaciones Web donde se enmarcan las aplicaciones de juegos en línea.

1.3.1.1 Tubería y filtros

Una tubería es una popular arquitectura que conecta componentes computacionales a través de conectores, de modo que el procesamiento de datos se ejecuta como un flujo. Los datos se transportan a

través de las tuberías entre los filtros, transformando gradualmente las entradas en salidas. Este estilo se percibe como una serie de transformaciones sobre sucesivas piezas de los datos de entrada [25].

El estilo arquitectónico tubería y filtros se puede utilizar cuando:

- Se puede especificar la secuencia de un número conocido de pasos.
- No se requiere esperar la respuesta asincrónica de cada paso.
- Se busca que todos los componentes situados corriente abajo sean capaces de inspeccionar y actuar sobre los datos que vienen de corriente arriba (pero no viceversa).

Ventajas:

- Es simple de entender e implementar.
- Es posible implementar procesos complejos con editores gráficos de líneas de tuberías o con comandos de línea.
- Fuerza un procesamiento secuencial.
- Los filtros se pueden empaquetar, y hacer paralelos o distribuidos.

Desventajas:

- El patrón puede resultar demasiado simplista, especialmente para orquestación de servicios que podrían ramificar la ejecución de la lógica de negocios de formas complicadas.
- No maneja con demasiada eficiencia construcciones condicionales, bucles y otras lógicas de control de flujo.
- Una desventaja adicional referida en la literatura sobre estilos concierne a que eventualmente pueden llegar a requerirse buffers de tamaño indefinido, por ejemplo en las tuberías de clasificación de datos.

- El estilo no es apto para manejar situaciones interactivas, sobre todo cuando se requieren actualizaciones incrementales de la representación en pantalla.
- La independencia de los filtros implica que es muy posible la duplicación de funciones de preparación que son efectuadas por otros filtros (por ejemplo, el control de corrección de un objeto de fecha).

1.3.1.2 Arquitectura en capas

Este estilo arquitectónico es un estilo de llamada y retorno, en el mismo cada capa proporciona servicios a la capa superior y se sirve de las prestaciones que le brinda la inferior, al dividir un sistema en capas, cada capa puede tratarse de forma independiente, sin tener que conocer los detalles de las demás. La división de un sistema en capas facilita el diseño modular, en la que cada capa encapsula un aspecto concreto del sistema y permite además la construcción de sistemas débilmente acoplados, lo que significa que si se minimiza las dependencias entre capas, resulta más fácil sustituir la implementación de una capa sin afectar al resto del sistema [25]. Como una variante de este tenemos:

Arquitectura en tres capas:

Una especialización muy usada en aplicaciones Web de la arquitectura en capas es la arquitectura de tres capas donde se observan muy bien delimitadas las responsabilidades de cada capa en la aplicación.

En la figura Fig. 1 se ejemplifica una arquitectura de tres capas. Una capa superior interactúa con una capa inferior mediante interfaces que definen las funcionalidades que la misma debe brindar. Es válido aclarar que todas estas capas pueden residir en un único ordenador (no es lo típico). Si bien lo más usual es que haya una multitud de ordenadores en donde reside la capa de presentación (son los clientes de la arquitectura cliente/servidor). Las capas de negocio y de datos pueden residir en el mismo ordenador, y si el crecimiento de las necesidades lo aconseja se pueden separar en dos o más ordenadores. Así, si el tamaño o complejidad de la base de datos aumenta, se puede separar en varios ordenadores los cuales recibirán las peticiones del ordenador en que resida la capa de negocio.Si por el contrario fuese la complejidad en la capa de negocio lo que obligase a la separación, esta capa de negocio podría residir en uno o más ordenadores que realizarían solicitudes a una única base de datos. En sistemas muy complejos

se llega a tener una serie de ordenadores sobre los cuales corre la capa de acceso a datos, y otra serie de ordenadores sobre los cuales corre la base de datos.

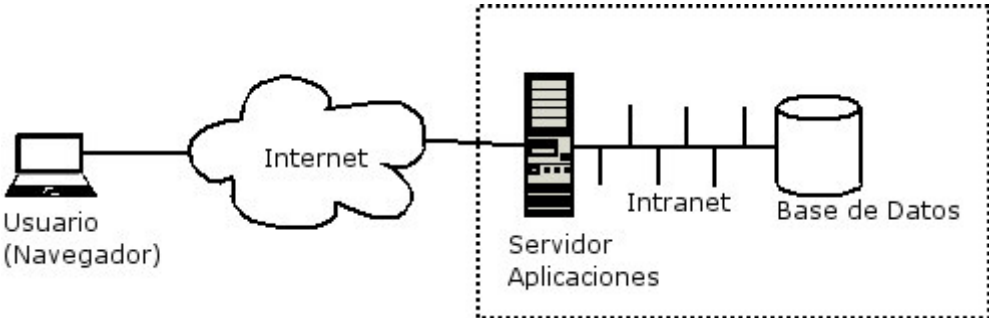

Fig. 1 Arquitectura en tres capas

Sin dudas el objetivo primordial de este estilo es la separación de la lógica de negocios y los datos de las vistas en una aplicación, aunque no se pueden descartar otras ventajas y desventajas que nos brinda este popular estilo arquitectónico y son las que se muestran a continuación.

Ventajas:

- Desarrollos paralelos (en cada capa).
- Aplicaciones más robustas debido al encapsulamiento.
- Mantenimiento y soporte más sencillo: es más sencillo cambiar un componente que modificar una aplicación monolítica.
- Mayor flexibilidad: se pueden añadir nuevos módulos para dotar al sistema de nueva funcionalidad.
- Alta escalabilidad: la principal ventaja de una aplicación distribuida bien diseñada es su buen escalado es decir, que puede manejar muchas peticiones con el mismo rendimiento simplemente añadiendo más hardware. El crecimiento es casi lineal y no es necesario añadir más código para conseguir esta escalabilidad.

Desventajas:

- Formatos, protocolos y transportes de la comunicación entre capas suelen ser específicos y propietarios.
- No todos los sistemas pueden estructurarse en capas.

1.3.1.3 Arquitecturas orientadas a servicios (SOA)

Las Arquitecturas Orientadas a Servicios (SOA: *Service Oriented Architecture*) están formadas por servicios de aplicación débilmente acoplados y altamente interoperables. Para comunicarse entre sí, estos servicios se basan en una definición formal independiente de la plataforma y del lenguaje de programación (por ejemplo WSDL: Web Services Description Language). La definición de la interfaz encapsula las particularidades de una implementación, lo que la hace independiente del fabricante, del lenguaje de programación o de la tecnología de desarrollo [21].

Esta arquitectura o estilo arquitectónico construye toda la topología de la aplicación como una topología de interfaces, implementaciones y llamados a interfaces, es una relación entre servicios y consumidores de servicios, ambos lo suficientemente amplios como para representar una función de negocio completa.

Desde el punto de vista arquitectónico, estas son las características de este estilo:

- Un servicio es una entidad de software que encapsula funcionalidad de negocios y proporciona dicha funcionalidad a otras entidades a través de interfaces públicas bien definidas.
- Los componentes del estilo (o sea, los servicios) están débilmente acoplados. El servicio puede recibir requerimientos de cualquier origen. La funcionalidad del servicio se puede ampliar o modificar sin rendir cuentas a quienes lo requieran.
- Los componentes que requieran un servicio pueden descubrirlo y utilizarlo dinámicamente. En general no se pretende que un servicio recuerde nada entre un requerimiento y el siguiente.

Entre las limitaciones, el principal problema del estilo se manifiesta en el hecho de que para poder interactuar con otro objeto a través de una invocación de procedimiento, se debe conocer su identidad [21].

Esta situación contrasta con lo que es el caso en estilos tubería y filtros, donde los filtros no necesitan poseer información sobre los otros filtros que constituyen el sistema. La consecuencia inmediata de esta característica es que cuando se modifica un objeto (por ejemplo, se cambia el nombre de un método, o el tipo de dato de algún argumento de invocación) se deben modificar también todos los objetos que lo invocan. También se presentan problemas de efectos colaterales en cascada: si A usa B y C también lo usa, el efecto de C sobre B puede afectar a A [21].

Ventajas

- Mejorar toma de decisiones.
    - Panorámica unificada. Más información con mejor calidad.
- Mejorar productividad de empleados.
    - Acceso óptimo a sistemas. No limitación de TIC
- Potenciar relación con los clientes y proveedores.
    - Mayor capacidad de respuesta a los clientes.
- Aplicaciones más productivas y flexibles.
- Aplicaciones más seguras y manejables.

Desventajas

- Los tiempos de llamado no son despreciables, gracias a la comunicación de la red, tamaño de los mensajes, entre otros. Esto necesariamente implica la utilización de mensajería confiable.

- La respuesta del servicio es afectada directamente por aspectos externos como problemas en la red, configuración, entre otros.

- Debe manejar comunicaciones no confiables, mensajes impredecibles, reintentos, mensajes fuera de secuencia, etcétera.

1.3.1.4 Arquitectura basada en componentes

Actualmente en el desarrollo de software hay una gran necesidad de hacer uso de la reutilización de partes o módulos de software existente, que podrían ser utilizadas para la generación de nuevas extensiones de las aplicaciones o las aplicaciones completas. Cuando se hable de reutilización en los procesos de ingeniería, está muy implícito el concepto de componente, pues a las partes eficientes de software que pueden ser utilizadas para la construcción de aplicaciones se les conoce como componentes software [25].

Características:

Una de las características más importantes de los componentes es que son reutilizables. Para ello los componentes deben satisfacer como mínimo el siguiente conjunto de características:

- ✓ **Identificable**: un componente debe tener una identificación clara y consistente que facilite su catalogación y búsqueda en repositorios de componentes.

- ✓ **Accesible sólo a través de su interfaz**: el componente debe exponer al público únicamente el conjunto de operaciones que lo caracteriza (interfaz) y ocultar sus detalles de implementación. Esta característica permite que un componente sea reemplazado por otro que implemente la misma interfaz.

- ✓ **Servicios son invariantes**: las operaciones que ofrece un componente, a través de su interfaz, no deben variar. La implementación de estos servicios puede ser modificada, pero no deben afectar la interfaz.

- ✓ **Documentado**: un componente debe tener una documentación adecuada que facilite su búsqueda en repositorios de componentes, evaluación, adaptación a nuevos entornos, integración con otros componentes y acceso a información de soporte.

A continuación se muestran las principales ventajas y desventajas que posee este estilo arquitectónico:

Ventajas:

- ✓ **Reutilización del software**. Lleva a alcanzar un mayor nivel de reutilización de software.
- ✓ **Simplifica las pruebas**. Permite que las pruebas sean ejecutadas probando cada uno de los componentes antes de probar el conjunto completo de componentes ensamblados.
- ✓ **Simplifica el mantenimiento del sistema**. Cuando existe un débil acoplamiento entre componentes, el desarrollador es libre de actualizar y/o agregar componentes según sea necesario, sin afectar otras partes del sistema.
- ✓ **Mayor calidad**. Dado que un componente puede ser construido y luego mejorado continuamente por un experto u organización, la calidad de una aplicación basada en componentes mejorará con el paso del tiempo.

Desventajas:

- ✓ Si no existen los componentes, hay que desarrollarlos y se puede perder mucho tiempo, así como en que estos componentes pueden tener conflictos si de estos sale una nueva versión es posible que haya que re-implementar estos componentes.
- ✓ Las actualizaciones de los componentes adquiridos no están en manos de los desarrolladores del sistema.

1.4 Patrones de diseño

Sin dudas los patrones de diseño son una herramienta muy potente para el diseño de aplicaciones Web, su objetivo fundamental es que sean reutilizados en el contexto donde el mismo se preste, puesto que

cada patrón tiene un objetivo específico a resolver. La reutilización nos permite: reducción de tiempos, disminución del esfuerzo de mantenimiento, eficiencia, consistencia, fiabilidad y protección de la inversión en desarrollos, entre otros. En este epígrafe se hará referencia a las principales definiciones de los patrones, así como se analizarán los patrones de diseño que se espera que puedan servir de ayuda durante el diseño de esta propuesta de arquitectura.

*"*Cada patrón describe un problema que ocurre una y otra vez en nuestro entorno, para describir después el núcleo de la solución a ese problema, de tal manera que esa solución pueda ser usada más de un millón de veces sin hacerlo ni siquiera dos veces de la misma forma" [8].

También se puede decir que los patrones de diseño hacen más fácil reutilizar con éxito los diseños y arquitecturas y que ayudan a los diseñadores a reutilizar con éxito diseños para obtener nuevos diseños.

Los patrones tienen una serie de elementos que los caracterizan:

- ✓ El nombre del patrón, describe el problema de diseño, su solución, y consecuencias en una o dos palabras. Tener un vocabulario de patrones permite hablar sobre ellos.
- ✓ El problema describe cuando aplicar el patrón. Se explica el problema y su contexto. Puede describir estructuras de clases u objetos que son sintomáticas de un diseño inflexible. Se incluye una lista de condiciones.
- ✓ La solución describe los elementos que forma el diseño, sus relaciones, responsabilidades y colaboraciones. No se describe un diseño particular. Un patrón es una plantilla.
- ✓ Las consecuencias son el resultado de aplicar el patrón.

A continuación se exponen los patrones de diseño que se espera puedan ser utilizados durante el diseño de esta propuesta.

1.4.1 Modelo-Vista-Control (MVC)

Este patrón propone la separación en distintos componentes de la interfaz de usuario (vistas), el modelo de negocio y la lógica de control [9]. Tal y como se muestra en la Fig. 2.

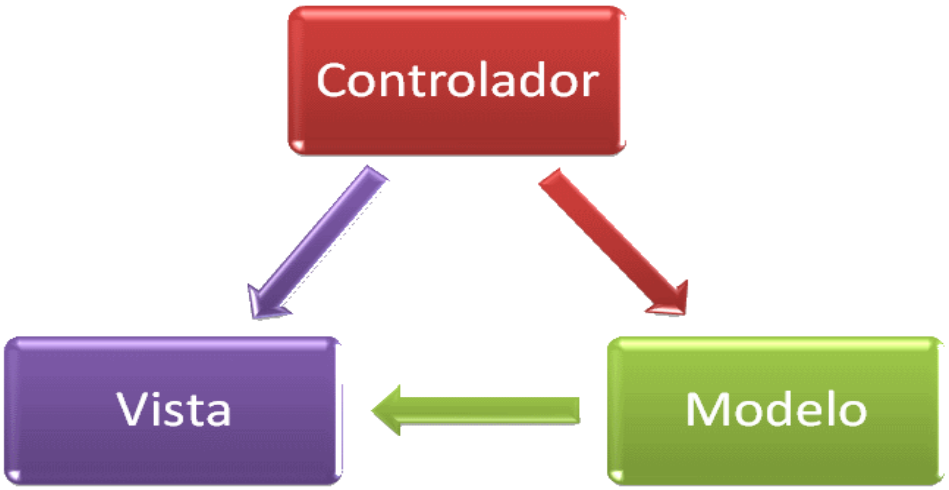

Fig. 2 Modelo-Vista-Controlador.

Estos componentes se describen a continuación:

- ✓ El modelo es la representación específica de la información con la cual el sistema opera (los datos).
- ✓ La vista transforma el modelo en una página Web que permite al usuario interactuar con ella.
- ✓ El controlador se encarga de procesar las interacciones del usuario y realiza los cambios apropiados en el modelo o en la vista.

Una vista es una "fotografía" del modelo (o una parte del mismo) en un determinado momento. Un control recibe un evento disparado por el usuario a través de la interfaz, accede al modelo de manera adecuada a la acción realizada, y presenta en una nueva vista el resultado de dicha acción. Por su parte, el modelo

consiste en el conjunto de objetos que modelan los procesos de negocio que se realizan a través del sistema [9]. ¿Cómo funcionaría en una aplicación web?

Las vistas serían las páginas HTML que el usuario visualiza en el navegador. A través de estas páginas el usuario interactúa con la aplicación, enviando eventos al servidor a través de peticiones HTTP. En el servidor se encuentra el código de control para estos eventos, que en función del evento concreto actúa sobre el modelo convenientemente. Los resultados de la acción se devuelven al usuario en forma de página HTML mediante la respuesta HTTP.

La clave está en la separación entre vista y modelo. El modelo suele ser más estable a lo largo del tiempo y menos sujeto a variaciones, mientras las vistas puede cambiar con frecuencia, ya sea por cambio del medio de presentación (por ejemplo HTML a WAP o a PDF) o por necesidades de usabilidad de la interfaz o simple renovación de la estética de la aplicación. Con esta clara separación las vistas pueden cambiar sin afectar al modelo y viceversa. Los controladores son los encargados de hacer de puente entre ambos, determinando el flujo de salida de la aplicación que se ve en cada momento. En la Fig. 3 que se muestra a continuación se puede observar bien claro la interacción del usuario a través de un navegador por medio de este patrón.

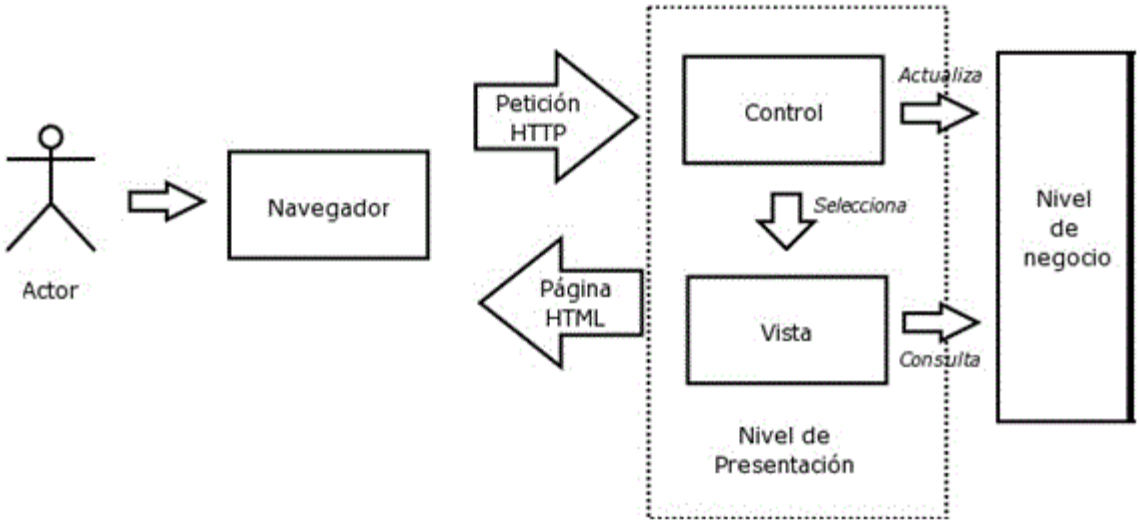

Fig. 3 MVC

Algunas de las principales ventajas de este clásico patrón de diseño Web se encuentran a continuación:

Ventajas del patrón MVC.

- ✓ Hay una clara separación entre los componentes de un programa, lo cual permite implementarlos por separado.
- ✓ La conexión entre el Modelo y sus Vistas es dinámica, se produce en tiempo de ejecución, no en tiempo de compilación.
- ✓ Al incorporar el modelo de arquitectura MVC a un diseño, las piezas de un programa se pueden construir por separado y luego unirlas en tiempo de ejecución. Posteriormente, si uno de los Componentes, se observa que funciona mal puede reemplazarse sin que las otras piezas se vean afectadas.
- ✓ La porción del programa que transforma los datos dentro del Modelo en una presentación gráfica es la vista. La vista incorpora la visión del Modelo a la escena, es la representación gráfica de la escena desde un punto de vista determinado, bajo condiciones de iluminación determinadas. El Controlador sabe que puede hacer el modelo e implementa la interfaz de usuario que permite iniciar la acción.

1.4.2 Patrones GOF

En este apartado se hace un estudio de los patrones GOF (*Gang of Four*) que se esperan sean utilizados en el diseño de esta arquitectura, no obstante, más adelante durante la propuesta de solución se definirá detalladamente donde estarán involucrados dentro de la misma, aquí además se analizarán las ventajas y desventajas de los mismos así como cuando es necesario utilizar cada uno de ellos.

1.4.2.1 Patrón Singleton

Este patrón garantiza que una clase sólo tenga una instancia y proporciona un punto de acceso global a ella [15]. Singleton se usa cuando:

- Deba haber exactamente una instancia de una clase y ésta debe ser accesible a los clientes desde un punto de acceso conocido.
- La única instancia debería ser extensible mediante herencia y los clientes deberían ser capaces de usar una instancia extendida sin modificar su código.

Ventajas:

- **Acceso controlado a la única instancia:** Encapsula su única instancia, puede tener un control estricto sobre como y cuando acceden a ella los clientes.
- **Espacio de nombres reducido:** Es una mejora sobre las variables globales. Evita contaminar el espacio de nombres con variables globales que almacenan las instancias.
- **Permite el refinamiento de operaciones y la representación:** Se puede crear una subclase de la clase Singleton, y es fácil configurar una aplicación con una instancia de esta clase extendida, incluso en tiempo de ejecución.
- **Permite un número variable de instancias:** Hace que sea fácil permitir más de una instancia de la clase. Solo se necesitaría cambiar la operación que otorga acceso a la instancia del Singleton.

1.4.2.2 Patrón Observer

Este patrón define una dependencia de uno-a-muchos entre objetos, de forma que cuando un objeto cambie de estado se notifique y se actualicen automáticamente todos los objetos que dependen de este [15].

Un efecto lateral habitual de dividir un sistema en una colección de clases cooperantes es la necesidad de mantener una consistencia entre objetos relacionados, pero sin hacer a las clases fuertemente acopladas ya que eso reduciría su reutilización [15].

El patrón Observer describe como establecer estas relaciones. Los principales objetos de este patrón son el sujeto y el observador. Un sujeto puede tener cualquier número de observadores dependientes de él. Cada vez que el sujeto cambia su estado se notifica a todos sus observadores. En respuesta, cada observador consultará al sujeto para sincronizar su estado con el estado de este.

Este tipo de interacción también se conoce como publicar-suscribir. El sujeto es quien publica las notificaciones. Envía estas sin tener que conocer quienes son sus observadores. Pueden suscribirse un número indeterminado de observadores para recibir notificaciones [15].

Usar Observer cuando:

- Una abstracción tiene dos aspectos y uno depende del otro. Encapsular estos aspectos en objetos separados permite modificarlos y reutilizarlos de forma independiente.
- Cuando un cambio en un objeto requiere cambiar otros y no sabemos cuántos objetos necesitan cambiarse.
- Cuando un objeto debería ser capaz de notificar a otros sin hacer suposiciones sobre quienes son dichos objetos (no queremos que estos objetos estén fuertemente acoplados).

Ventajas

- **Acoplamiento abstracto entre Sujeto y Observador:** Todo lo que un sujeto sabe es que tiene una lista de observadores que se ajusta a la interfaz simple de la clase abstracta Observador. El sujeto no conoce la clase concreta de ningún observador. Por lo tanto, el acoplamiento entre sujetos y observadores es mínimo, pueden pertenecer a diferentes capas de abstracción de un sistema.
- **Capacidad de comunicación mediante difusión:** A diferencia de una petición ordinaria, la notificación enviada por un sujeto no necesita especificar su receptor, se envía automáticamente a todos los objetos interesados que se hayan suscripto a ella. Al sujeto no le importa cuantos objetos interesados haya, su única responsabilidad es notificar a sus observadores (libertad de añadir y quitar observadores en cualquier momento).

Desventajas

- **Actualizaciones inesperadas:** Dado que los observadores no saben de la presencia de los otros, pueden no saber el coste último de cambiar el sujeto. Una operación aparentemente inofensiva sobre el sujeto puede dar lugar a una serie de actualizaciones en cascada de los observadores y sus objetos dependientes.

En la tabla que se muestra a continuación se encuentra un resumen de todos los patrones GOF, según sus clasificaciones (Creacionales, Estructurales y de Comportamiento).

Creación	Estructurales	Comportamiento
▪ Abstract Factory ▪ Builder ▪ Factory Method ▪ Prototype ▪ Singleton	▪ Adapter ▪ Bridge ▪ Composite ▪ Decorator ▪ Facade ▪ Flyweight ▪ Proxy	▪ Chain of Responsability ▪ Command ▪ Interpreter ▪ Iterator ▪ Mediator ▪ Memento ▪ Observer ▪ State ▪ Strategy ▪ Template Method ▪ Visitor

Tabla 1 Tabla de Patrones GOF.

En fin estos patrones son una gran familia que permiten entre otros factores el ahorro del tiempo, ser exactos a la hora de darle respuesta a un problema determinado, si cada patrón se emplea en el contexto que él requiere o sea para lo cual fue destinado, la eficiencia y el resultado a obtener será elevado, y mucho más cuando en la actualidad, cada vez se hacen más necesario el uso de estos patrones, por la complejidad que va obteniendo el software y las necesidades del hombre, que cada vez necesita que el software sea más eficiente.

1.4.3 Patrones GRASP

Los patrones Grasp (Patrones de Software para la asignación General de Responsabilidades) en inglés (*General Responsability Assignment Software Patterns*), se dividen en dos grupos, 5 principales (Bajo Acoplamiento, Alta Cohesión, Experto, Creador y Controlador) y 4 de apoyo (Polimorfismo, Fabricación Pura, Indirección y No hables con extraños) [18]. En este apartado solo se pretende analizar los patrones que se esperan que sirvan de ayuda en el diseño de nuestra propuesta.

1.4.3.1 Experto

El principal objetivo de este patrón es asignar una responsabilidad al experto en información, es decir, la responsabilidad de realizar una labor es de la clase que tiene o puede tener los datos involucrados (atributos) en otras palabras “Una clase, debe contener toda la información necesaria para realizar la labor que tiene encomendada” [18].

1.4.3.2 Bajo Acoplamiento

Este patrón es un principio que asigna la responsabilidad de controlar el flujo de eventos del sistema, a clases específicas. Esto facilita la centralización de actividades (validaciones, seguridad, etc.). El controlador no realiza estas actividades, las delega en otras clases con las que mantiene un modelo de

alta cohesión. Un error muy común es asignarle demasiada responsabilidad y alto nivel de acoplamiento con el resto de los componentes del sistema [18].

1.4.3.3 Alta Cohesión

La cohesión es una medida de la fuerza con la que se relacionan las clases y el grado de focalización de las responsabilidades de un elemento, cada elemento de nuestro diseño debe realizar una labor única dentro del sistema, no desempeñada por el resto de los elementos y auto-identificable, una clase con baja cohesión hace muchas cosas no relacionadas o hace demasiado trabajo.

Un ejemplo de esto son aquellas clases que en un modelo hacen varias cosas, los ejemplos de buen diseño se producen cuando se crean los denominados "paquetes de servicio" o clases agrupadas por funcionalidades que son fácilmente reutilizables (bien por uso directo o por herencia) [18].

1.5 Gestores de Base de Datos

Los Sistemas Gestores de Bases de Datos son un tipo de software muy específico, dedicado a servir de interfaz entre la base de datos, el usuario y las aplicaciones que la utilizan. Se compone de un lenguaje de definición de datos, de un lenguaje de manipulación de datos y de un lenguaje de consulta. Permiten crear y mantener una base de datos, asegurando su integridad, confidencialidad y seguridad.

La necesidad de almacenar información en un juego en línea obliga a la selección de alguna de estas herramientas que facilitan la gestión de los datos persistentes. Ante esta necesidad se tomó la decisión de estudiar los gestores de bases de datos más utilizados en la actualidad para aplicaciones Web, los cuales se muestran a continuación:

1.5.1 MySQL

MySQL es un sistema de gestión de base de datos relacional, multihilos y multiusuario. Es muy utilizado en aplicaciones web y en plataformas (Linux/Windows-Apache-MySQL-PHP/Perl/Python), y por herramientas de seguimiento de errores como Bugzilla. Su popularidad como aplicación web está muy

ligada a PHP, que a menudo aparece en combinación con MySQL. A continuación se presentan algunas de las principales ventajas que nos brinda este potente gestor:

Ventajas de MySQL

- ✓ Aprovecha la potencia de sistemas multiprocesadores, gracias a su implementación multihilo.
- ✓ Soporta gran cantidad de tipos de datos para las columnas.
- ✓ Dispone de API's en gran cantidad de lenguajes (C, C++, Java, PHP).
- ✓ Soporta hasta 32 índices por tabla.
- ✓ Gestión de usuarios y *passwords*, manteniendo un muy buen nivel de seguridad en los datos.
- ✓ Conectividad Segura.
- ✓ Transacciones y claves foráneas.
- ✓ Búsqueda e indexación de campos de texto.
- ✓ Disponibilidad en gran cantidad de plataformas y sistemas [23].

1.5.2 PostgreSQL

PostgreSQL cuenta con sofisticadas particularidades tales como la versión multicontrol de concurrencia (MVCC), replicación asincrónica, transacciones jerarquizadas, en línea, un sofisticado plan de consulta, y un excelente optimizador. Es altamente escalable, tanto en la formidable cantidad de datos que puede administrar como en la cifra de usuarios concurrentes que puede acomodar. Trabaja en varios sistemas operativos como Linux, UNIX y Windows. También resiste el acaparamiento de grandes objetos binarios, ya sean imágenes, sonidos o vídeo. Tiene interfaces de programación originaria de C / C + +, Java, Net, Perl, Python, Ruby, Tcl, ODBC, entre otros" [23]. Además, no es controlado por ninguna compañía responde al esfuerzo de una comunidad global de desarrolladores. A continuación se exponen algunas de las principales ventajas que nos brinda este gestor:

Ventajas de PostgreSQL

PostgreSQL ofrece muchas ventajas para su compañía o negocio respecto a otros sistemas de bases de datos:

- **Instalación ilimitada:** Es frecuente que las bases de datos comerciales sean instaladas en más servidores de lo que permite la licencia. Algunos proveedores comerciales consideran a esto la principal fuente de incumplimiento de licencia. Con PostgreSQL, nadie puede demandarlo por violar acuerdos de licencia, puesto que no hay costo asociado a la licencia del software.
  Esto tiene varias ventajas adicionales:
  - Modelos de negocios más rentables con instalaciones a gran escala.
  - No existe la posibilidad de ser auditado para verificar cumplimiento de licencia en ningún momento.
  - Flexibilidad para hacer investigación y desarrollo sin necesidad de incurrir en costos adicionales de licenciamiento.
- **Estabilidad y confiabilidad:** En contraste a muchos sistemas de bases de datos, es extremadamente común que las grandes compañías reporten que PostgreSQL nunca ha presentado caídas en varios años de operación de alta actividad.
- **Extensible:** El código fuente está disponible para todos sin costo. Si usted necesita extender o personalizar PostgreSQL de alguna manera, pueden hacerlo con un mínimo esfuerzo, sin costos adicionales. Esto es complementado por la comunidad de profesionales y entusiastas de PostgreSQL alrededor del mundo que también extienden PostgreSQL todos los días.
- **Multiplataforma:** PostgreSQL está disponible en casi cualquier Unix (34 plataformas en la última versión estable), y una versión nativa de Windows está actualmente en estado veta de pruebas.
- **Diseñado para ambientes de alto volumen:** PostgreSQL usa una estrategia de almacenamiento de filas llamada MVCC para conseguir una mejor respuesta en ambientes de grandes volúmenes. Los principales proveedores de sistemas de bases de datos usan también esta tecnología, por las mismas razones.

1.6 Herramientas CASE

Las herramientas CASE (*Computer Aided Software Engineering*, Ingeniería de Software Asistida por Ordenador) son diversas aplicaciones informáticas destinadas a aumentar la productividad en el desarrollo de software reduciendo el costo de las mismas en términos de tiempo y de dinero. Sirven de apoyo en todos los aspectos del ciclo de vida de desarrollo del software, en tareas como el proceso de realizar un diseño del proyecto, cálculo de costos, implementación de parte del código automáticamente con el diseño dado, compilación automática, documentación o detección de errores entre otras.

Como una buena caja de herramientas, una buena herramienta de modelado ofrece todas las herramientas necesarias para conseguir hacer eficientemente varios trabajos, sin dejarte nunca sin la herramienta correcta [20].

Las herramientas de modelado deberían soportar las siguientes funcionalidades:

- ✓ Soporte para toda la notación y semántica de UML.
- ✓ Facilitar la captura de información en un repositorio subyacente permitiendo la reutilización entre diagramas.
- ✓ Posibilidad de personalizar las propiedades de definición de elementos subyacentes de modelos UML.
- ✓ Permitir a varios equipos de analistas trabajar en los mismos datos a la vez.
- ✓ Posibilidad de capturar los requisitos, asociarlos con elementos de modelado que los satisfagan y localizar cómo han sido satisfechos los requisitos en cada uno de los pasos del desarrollo.
- ✓ Posibilitar la creación de informes y documentación personalizados en tus diseños, y la salida de estos informes en varios formatos, incluyendo HTML para la distribución en la Internet o Intranet local.

- ✓ Posibilidad para generar y realizar ingeniería inversa (por ejemplo C++, Java, etcétera.) para facilitar el análisis y diseño interactivo, para volver a usar código o librerías de clase existentes, y para documentar el código.

- ✓ A continuación se hace un análisis de las herramientas de modelado más usadas, con vistas a seleccionar la más adecuada para el modelado visual de nuestra propuesta que será definida en capítulos posteriores.

1.6.1 Rational Rose

Esta es una de las más poderosas herramientas de modelado visual para el análisis y diseño de sistemas basados en objetos. Se utiliza para modelar un sistema antes de proceder a construirlo. Cubre todo el ciclo de vida de un proyecto: concepción y formalización del modelo, construcción de los componentes, transición a los usuarios y certificación de las distintas fases. La interfaz de Rational Rose (en adelante solo Rose), está formada por los siguientes elementos principales:

- ➢ Browser o Navegador, que permite navegar rápidamente a través de las distintas vistas del modelo y permite establecer una trazabilidad real entre el modelo (análisis y diseño) y el código ejecutable.
- ➢ Ventana de documentación, para manejar los documentos del ítem seleccionado en cualquiera de los diagramas.

Barra de herramientas, para acceder rápidamente a las acciones comunes a ejecutar para cada uno de los diagramas del modelo [13].

1.6.2 Visual Paradigm Suite

Visual Paradigm en la versión 3.4, es una poderosa herramienta CASE de modelación visual. Utiliza UML como lenguaje para el modelado, además se puede decir que es un conjunto de herramientas de modelado que permiten realizar el modelado dentro del proceso de desarrollo de software [10].

A continuación se describen las principales herramientas presentes dentro de esta suite:

- Visual Paradigm 3.4 for UML Enterprise Edition: Es una herramienta de modelado diseñada para un gran número de usuarios, incluyendo ingenieros de sistemas, analistas de sistemas, analistas de negocio, arquitectos de sistemas. Además, se integra con IDEs (Eclipse, JBuilder, NetBeans, Intelli JIDEA, JDeveloper and Web Logic Workshop) para soportar la fase de implementación de desarrollo de software. La transición desde el análisis al diseño y después a la implementación es cuidadosamente integrada dentro de la herramienta CASE, de esta manera, se reduce significativamente el esfuerzo en todas las etapas del ciclo de vida del desarrollo del software. Incluye además un conjunto de herramientas que soportan Object-Relational Mapping (ORM), como Hibernate, lo cual incluye generación completamente orientada a objetos, listo para usar librerías para obtener y modificar registros de base de datos para una gran variedad de gestores de base de datos, y la sincronización entre los diagramas de clases y los diagramas de entidad-relación (ERD). Presenta además generación de código e ingeniería inversa del código. Permite generar los EJB y así mismo los descriptores de despliegue para varios servidores de aplicación. Distinto de muchas herramientas de modelado pueden extenderse sus diagramas hechos desde el Visio y de Rational Rose. Soporta la última versión de UML 2.0.

1.7 Lenguaje unificado de modelado (UML)

El Lenguaje Unificado de Modelado (UML) es un lenguaje para visualizar, especificar, construir y documentar los artefactos de un sistema software [22]. Es un estándar del Grupo de administración de objetos (OMG por sus siglas en inglés). Está compuesto por diversos elementos gráficos que se combinan para conformar diagramas. Debido a que el UML es un lenguaje, cuenta con reglas para combinar tales elementos.

La finalidad de los diagramas es presentar diversas perspectivas de un sistema, a las cuales se les conoce como modelo. El modelo UML de un sistema es similar a un modelo a escala de un edificio junto con la interpretación del artista del edificio. Es importante destacar que un modelo UML describe lo que supuestamente hará un sistema, pero no dice cómo implementar dicho sistema [22].

Una característica que UML brinda para beneficiar a los modeladores es escoger una herramienta de modelado. Tiempos atrás, el modelador primero tenía que seleccionar una notación de metodología, y

después estaba limitado a seleccionar una herramienta que la soportara. Ahora con UML como estándar, la elección de notación ya se ha hecho para el modelador. Y con todas las herramientas de modelado soportando UML, el modelador puede seleccionar la herramienta basada en las áreas claves de funcionalidad soportadas que permiten resolver los problemas y documentar las soluciones [22].

1.8 Metodologías de desarrollo de software

El desarrollo de software no es sin dudas una tarea fácil. Como resultado a este problema ha surgido una alternativa desde hace mucho: la Metodología de Desarrollo de Software. Las metodologías imponen un proceso disciplinado sobre el desarrollo de software con el fin de hacerlo más predecible y eficiente. Lo hacen desarrollando un proceso detallado con un fuerte énfasis en planificar inspirado por otras disciplinas de la ingeniería. Las metodologías ingenieriles han estado presentes durante mucho tiempo. No se han distinguido precisamente por ser muy exitosas. Aún menos por su popularidad. La crítica más frecuente a estas metodologías es que son burocráticas. Hay tanto que hacer para seguir la metodología que el ritmo entero del desarrollo se retarda.

En cualquier ámbito de ingeniería hay una fractura entre los responsables de analizar y definir los problemas (necesidades), y los expertos en proveer soluciones (tecnología). Las metodologías nacen para intentar solucionar este conflicto. Su propósito es establecer un contrato social entre todos los participantes en un proyecto para conseguir la solución más eficaz con los recursos disponibles [11].

Con ánimos de buscar una metodología que se ajuste al medio en que se desarrollará un juego en línea se analizaron algunas de las más usadas en la actualidad, las cuales se clasifican en metodologías pesadas y metodologías ágiles.

1.8.1 Metodologías pesadas

Hoy en día existen numerosas propuestas que inciden en distintas dimensiones del proceso de desarrollo. Un ejemplo de ellas son las propuestas tradicionales centradas específicamente en el control del proceso. Estas han demostrado ser efectivas y necesarias en un gran número de proyectos, sobre todo aquellos proyectos de gran tamaño (respecto a tiempo y recursos).

1.8.1.1 Metodología RUP (Rational Unified Process)

El Proceso Unificado Racional es un proceso de desarrollo de software por lo que es el conjunto de actividades necesarias para transformar los requisitos de un usuario en un sistema software. Sin embargo, el Proceso Unificado es más que un simple proceso, es un marco de trabajo genérico que puede especializarse para una gran variedad de sistemas software, para diferentes áreas de aplicación, diferentes tipos de organizaciones, diferentes niveles de aptitud y diferentes tamaños de proyecto [4].

El Proceso Unificado está basado en componentes, lo cual quiere decir que el sistema software en construcción está formado por componentes software interconectados a través de interfaces bien definidas.

El Proceso Unificado utiliza el Lenguaje Unificado de Modelado (Unified Modeling Language, UML) para preparar todos los esquemas de un sistema software [4].

No obstante, los verdaderos aspectos definitorios del Proceso Unificado se resumen en tres frases clave: dirigido por casos de uso, centrado en la arquitectura, e iterativo e incremental.

1.8.2 Metodologías ágiles

En una reunión celebrada en febrero de 2001 en Utah-EEUU, nace el término **ágil** aplicado al desarrollo de software. En esta reunión participan un grupo de 17 expertos de la industria del software, incluyendo algunos de los creadores o impulsores de metodologías de software. Su objetivo fue esbozar los valores y principios que deberían permitir a los equipos desarrollar software rápidamente y respondiendo a los cambios que puedan surgir a lo largo del proyecto. Se pretendía ofrecer una alternativa a los procesos de desarrollo de software tradicionales, caracterizados por ser rígidos y dirigidos por la documentación que se genera en cada una de las actividades desarrolladas. Varias de las denominadas metodologías ágiles ya estaban siendo utilizadas con éxito en proyectos reales, pero les faltaba una mayor difusión y reconocimiento.

Tras esta reunión se creó *The Agile Alliance*, una organización, sin ánimo de lucro, dedicada a promover los conceptos relacionados con el desarrollo ágil de software y ayudar a las organizaciones para que

adopten dichos conceptos. El punto de partida fue el Manifiesto Ágil, un documento que resume la filosofía "ágil". [12].

El Manifiesto Ágil

El Manifiesto comienza enumerando los principales valores del desarrollo ágil. Se valora:

- Al individuo y las interacciones del equipo de desarrollo sobre el proceso y las herramientas.
- Desarrollar software que funciona más que conseguir una buena documentación.
- La colaboración con el cliente más que la negociación de un contrato.
- Responder a los cambios más que seguir estrictamente un plan [12].

1.8.2.1 eXtremeProgramming (XP)

La metodología Programación Extrema o Extreme Programming (XP), es una metodología ágil centrada en potenciar las relaciones interpersonales como clave para el éxito en desarrollo de software, promoviendo el trabajo en equipo, preocupándose por el aprendizaje de los desarrolladores, y propiciando un buen clima de trabajo. XP se basa en realimentación continua entre el cliente y el equipo de desarrollo, comunicación fluida entre todos los participantes, simplicidad en las soluciones implementadas y coraje para enfrentar los cambios. XP se define como especialmente adecuada para proyectos con requisitos imprecisos y muy cambiantes, y donde existe un alto riesgo técnico [12].

La principal suposición que se realiza en XP es la posibilidad de disminuir la mítica curva exponencial del costo del cambio a lo largo del proyecto, lo suficiente para que el diseño evolutivo funcione. XP apuesta por un crecimiento lento del costo del cambio y con un comportamiento asintótico [12].

1.8.3 Comparación entre las metodologías ágiles y las pesadas

A continuación, en la **Error! Reference source not found.** se muestra una comparación entre las metodologías ágiles y las metodologías pesadas o también conocidas como tradicionales.

Metodologías Ágiles	Metodologías Tradicionales
Basadas en heurísticas provenientes de prácticas de producción de código.	Basadas en normas provenientes de estándares seguidos por el entorno de desarrollo.
Especialmente preparadas para cambios durante el proyecto.	Cierta resistencia a los cambios.
Impuestas internamente (por el equipo de desarrollo).	Impuestas externamente.
Proceso menos controlado, con menos principios.	Proceso mucho más controlado, con numerosas políticas/normas.
No existe contrato tradicional o al menos es bastante flexible.	Existe un contrato prefijado.
El cliente es parte del equipo de desarrollo.	El cliente interactúa con el equipo de desarrollo mediante reuniones.
Grupos pequeños (<10 integrantes) y Trabajando en el mismo sitio.	Grupos grandes y posiblemente distribuidos.
Pocos artefactos.	Más artefactos.
Pocos roles.	Más roles.

Menos énfasis en la arquitectura del software.	La arquitectura del software es esencial y se expresa mediante modelos.

Tabla 2 Comparaciones entre las metodologías ágiles y las metodologías pesadas.

1.9 Conclusiones

En este capítulo se trataron los diferentes conceptos que son de vital importancia para dar solución al problema planteado. El concepto más importante tenido en cuenta es el de arquitectura de software, debido a que constituye el centro de la investigación.

Además, se realizó un estudio detallado de los principales estilos y patrones arquitectónicos que permitan lograr un diseño de alto nivel lo más robusto y escalable posible, pero a la vez que se ajuste a las características de este tipo de aplicaciones. Y por último se estudiaron otros aspectos entre los que se encuentran las metodologías de desarrollo, herramientas y lenguaje de modelado.

2

Capítulo 2: Propuesta de Arquitectura

2.1 Introducción

En este capítulo se realizará una propuesta de arquitectura para el desarrollo de juegos en línea en la web, con el objetivo de darle respuesta al problema planteado en el diseño teórico de la investigación. Además, se propone la metodología, herramientas y los patrones de diseño y arquitecturas más apropiados a utilizar para el desarrollo de aplicaciones de juegos en línea.

2.2 Rol del arquitecto

Indudablemente el rol del arquitecto de software es crítico y sumamente importante, puesto que requiere de una gran variedad de conocimientos, tales como: ingeniería de requerimientos, teoría de arquitecturas de software, codificación, tecnologías de desarrollo, plataformas de software y hardware etc. Por otra parte, el arquitecto de Software debe dominar la mayor cantidad de tecnologías de software y prácticas de diseño, para así poder tomar decisiones adecuadas para garantizar el mejor desempeño de las aplicaciones.

Además, requiere de saber negociar intereses encontrados de múltiples involucrados en el desarrollo de un sistema de software, promover la colaboración entre el equipo, entender la relación entre atributos de calidad y estructuras, ser capaz de transmitir claramente la arquitectura a los equipos, escuchar, y entender múltiples puntos de vista. El arquitecto de software además debe interaccionar con todos los involucrados en el desarrollo de un sistema de software, y ser capaz de dialogar con el analista para obtener los requerimientos significativos, diseñarlos y transmitirlos al programador para su correcta codificación.

2.3 Línea base de la arquitectura

La Línea Base de la Arquitectura contiene elementos de gran importancia para lograr la máxima abstracción en el diseño arquitectónico de la aplicación. En la misma se exponen los estilos arquitectónicos, así como los principales elementos de la arquitectura, se describen los principales patrones de arquitectura utilizados, las tecnologías y herramientas de software que se utilizarán en el sistema a desarrollar. El propósito de la Línea Base de la Arquitectura es: proporcionar la información necesaria para estructurar el sistema desde el más alto nivel de abstracción. En ella se describe la estructura del sistema en cuanto a los elementos, los conectores, las configuraciones y sus restricciones.

Los usuarios de la Línea Base de la Arquitectura son:

- ✓ El equipo de arquitectos del proyecto, que le sirve de guía para la toma de decisiones arquitectónicas y son los encargados del mantenimiento y refinamiento de esta línea base.

- ✓ Los miembros del equipo de desarrollo encargados de desarrollar la aplicación, la utilizan como la guía para la implementación del sistema.
- ✓ Los clientes tienen en ella una garantía de la calidad y el conocimiento sobre en qué tecnología está desarrollada su solución.

2.3.1 Alcance

El objetivo de la Línea Base de la Arquitectura es describir la estructura del sistema en un alto nivel de abstracción. Describir detalladamente el organigrama de la arquitectura según los estilos arquitectónicos que se utilizarán. También las principales herramientas de desarrollo y como se adaptarán a la solución, se propone la utilización de un conjunto de patrones que resuelven problemas que se podrían presentar a lo largo del desarrollo del sistema.

2.3.2 Estilo Arquitectónico Seleccionado

Como se describió en el capítulo anterior la arquitectura de software es la estructura más importante de un sistema, puesto que es la encargada de la estructura y relaciones de los componentes de dicho sistema. Partiendo de que los estilos arquitectónicos representan el nivel de abstracción mayor para estructurar el sistema, la elección del mismo está dada por el tipo de aplicación que se vaya a desarrollar. Teniendo en cuenta los elementos mencionados anteriormente, y conociendo además que es un estilo arquitectónico clásico para el desarrollo de aplicaciones Web donde se enmarcan los juegos en línea, la arquitectura del sistema se ha estructurado a partir del estilo arquitectónico Arquitectura en Capas (*Layers*), en la Fig. 4 Arquitectura 3 capas. que se muestra a continuación, se presenta la estructura lógica del diseño de una Arquitectura tres Capas que será la propuesta de arquitectura.

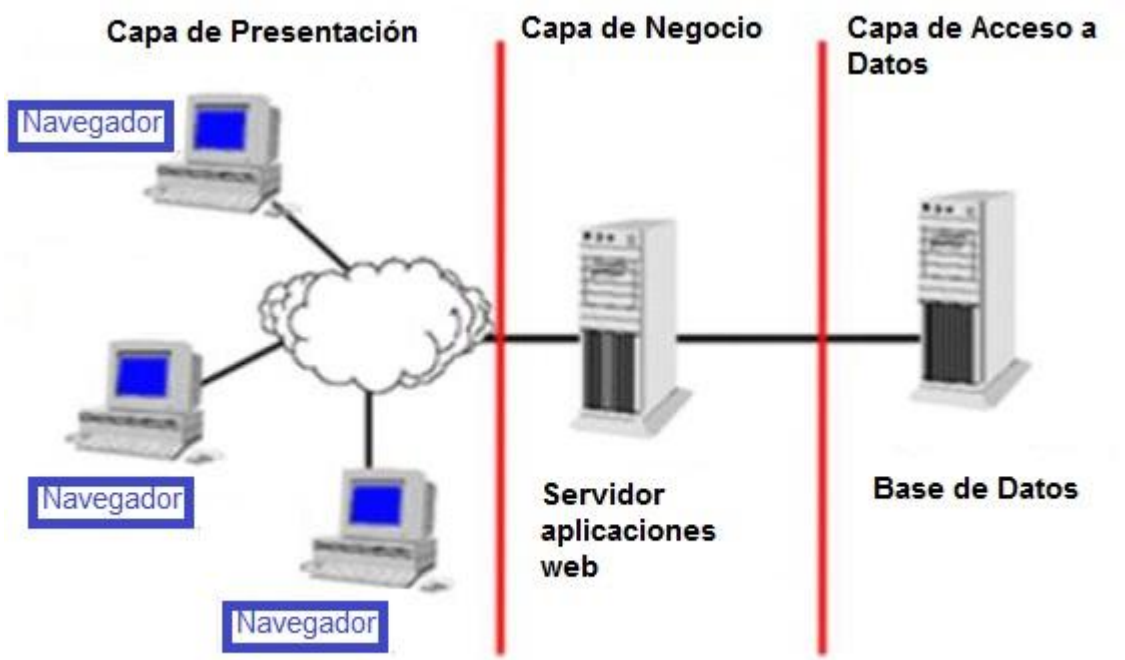

Fig. 4 Arquitectura 3 capas.

El usuario interacciona con las aplicaciones web a través del navegador. Como consecuencia de la actividad del usuario, se envían peticiones al servidor de aplicaciones, donde se aloja la aplicación y que normalmente hace uso de una base de datos que almacena toda la información relacionada con la misma. El servidor procesa la petición y devuelve la respuesta al navegador que la presenta al usuario. Por tanto, el sistema se distribuye en tres componentes: el navegador, que presenta la interfaz al usuario, la aplicación, que se encarga de realizar las operaciones necesarias según las acciones llevadas a cabo por éste y la base de datos, donde la información relacionada con la aplicación se hace persistente. Esta distribución se conoce como el modelo o arquitectura de tres capas.

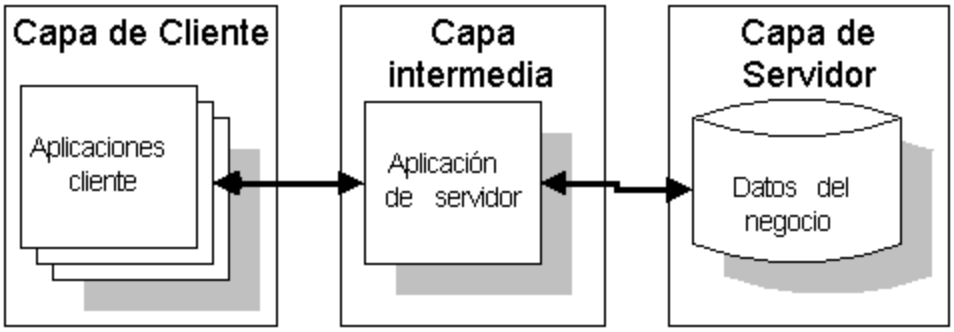

Fig. 5 Estructura de la arquitectura de tres capas.

Distribución de cada una de las capas:

Capa de presentación:

Esta capa es la que será visible para el usuario, denominada también como **capa de usuario**, presenta el sistema al usuario, le comunica la información y captura la información del usuario en un mínimo de proceso. Esta capa se comunica únicamente con la capa de negocio. También es conocida como interfaz gráfica y tendrá la característica de ser amigable, entendible y fácil de usar para el usuario.

Responsabilidades de la capa de presentación:

- ✓ Obtener información del usuario.
- ✓ Enviar la información del usuario a los servicios de negocios para su procesamiento.
- ✓ Recibir los resultados del procesamiento de los servicios de negocios.
- ✓ Presentar estos resultados al usuario.

Capa de negocio:

Aquí es donde estarán los programas que se ejecutan, se reciben las peticiones del usuario y se envían las respuestas tras el proceso. Esta capa se denomina capa de negocio e incluso de lógica del negocio porque es aquí donde se establecerán todas las reglas que deben cumplirse. Esta capa se comunicará con la capa de presentación, para recibir las solicitudes y presentar los resultados, y con la capa de datos, para solicitar al gestor de base de datos para almacenar o recuperar datos de él.

Responsabilidades de la capa de negocio:

- ✓ Recibir la entrada del nivel de presentación.
- ✓ Interactuar con los servicios de datos para ejecutar las operaciones de negocios que sean necesarias para dar respuesta a las peticiones hechas por el usuario desde la capa de presentación.
- ✓ Enviar el resultado procesado al nivel de presentación.

Capa de acceso a datos:

Es donde residen las clases que se encargan de gestionar todo el acceso a la información contenida en la base de datos. Esta además recibe solicitudes de almacenamiento o recuperación de información desde la capa de negocio.

Responsabilidades de la capa de datos:

- ✓ Almacenar los datos.
- ✓ Recuperar los datos.
- ✓ Mantener los datos.
- ✓ La integridad de los datos.

Este estilo sin dudas permite ganar en organización del sistema, además soporta un diseño basado en niveles de abstracción crecientes, lo cual a su vez permite a los implementadores la partición de un problema complejo en una secuencia de pasos incrementales, proporciona una amplia reutilización, facilita la corrección de errores y se obtiene al final una solución altamente flexible y reutilizable como la que se quiere para el desarrollo de juegos en línea.

2.3.3 Patrones de diseño utilizados

En este apartado se especifica la utilización de cada uno delos patrones de diseño estudiados en el capítulo 1 durante el diseño de la propuesta de arquitectura.

2.3.3.1 Modelo Vista Controlador (MVC)

Un propósito que sin dudas es de gran importancia en cualquier sistema es el de tomar datos de un almacenamiento y mostrarlos al usuario o cliente que de alguna forma interactúe con la aplicación. Después de que el usuario introduzca cualquier tipo de modificaciones, las mismas se reflejarán en el almacenamiento. Dado que el flujo de información ocurre entre el almacenamiento y la interfaz, una tentación común, un impulso espontáneo (hoy se llamaría un anti-patrón) seria unir ambas piezas para reducir la cantidad de código y optimizar el rendimiento. Sin embargo, esta idea es antagónica al hecho de que la interfaz suele cambiar, además de que la programación de interfaces de aplicaciones Web donde se enmarcan las aplicaciones de juegos en línea, requiere habilidades muy distintas de la programación de lógica de negocios. Pues está claro que con la presencia de uno de los patrones clásicos de diseño Web se evitaría cualquier inconveniente que se pueda ocasionar a este tipo de aplicaciones, aun mucho más cuando nos encontramos en presencia de un sistema que implique un gran manejo y almacenamiento de información como son los propios juegos en línea, en este caso nos referimos al patrón Modelo-Vista-Controlador (MVC), el cual será utilizado para el diseño de la capa de presentación, a continuación se muestra en la **Fig. 6 Modelo Vista Controlador** (**MVC**). la representación lógica de este [16].

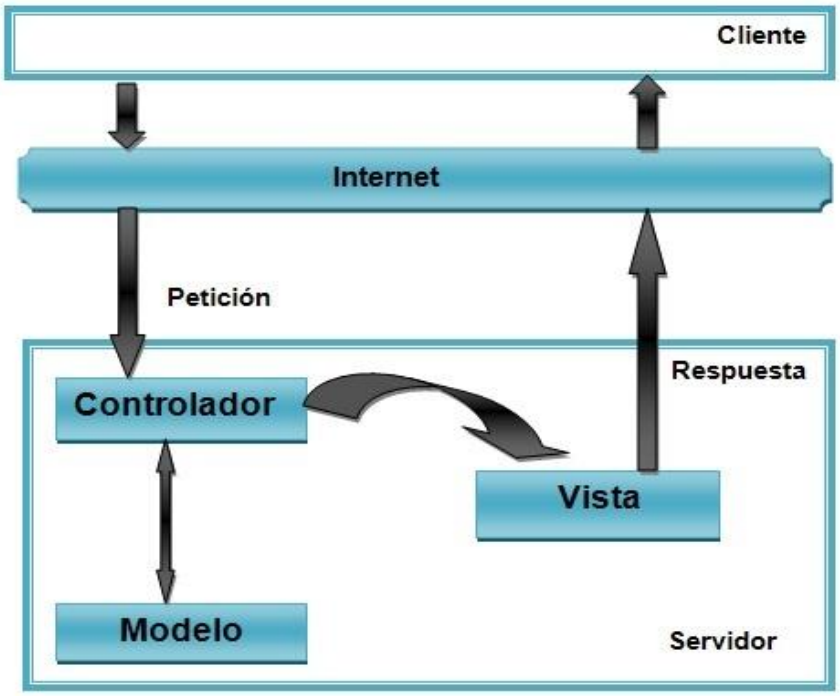

Fig. 6 Modelo Vista Controlador (MVC).

Aplicando el MVC.

Aplicación en la capa de presentación:

Esta capa será la encargada de la comunicación entre el jugador y las demás capas. Aquí se hace uso del patrón MVC.

Controladores:

Estos objetos son responsables de procesar las entradas del usuario en forma de peticiones HTTP.

Modelo:

Estos objetos contienen los datos resultantes de la ejecución de la lógica de negocio del juego.

Vistas:

Estas serán las responsables de mostrar el modelo resultante al usuario en respuesta de una petición.

2.3.3.2 Patrones GOF

Después del análisis realizado en el estado del arte acerca de los patrones de diseño que van a estar presentes en esta propuesta de arquitectura, a continuación se especifica la utilización de cada uno de ellos durante el diseño de la misma.

Singleton:

Teniendo en cuenta que el patrón de diseño *Singleton* posibilita la restricción para la creación de objetos de una clase determinada garantizando que sólo exista una instancia de dicha clase además de proporcionar un punto de acceso global a ella, y teniendo en cuenta además que cualquier juego en línea que se desarrolle sobre esta propuesta de arquitectura de alguna forma tendrá que utilizar datos almacenados en una base de datos, entonces con la utilización de este patrón se garantizará que la conexión a la base de datos sea única.

Observer:

El patrón Observador también conocido como "*spider"* el cual define una dependencia del tipo uno-a-muchos entre objetos, de manera que cuando uno de los objetos cambia su estado, el observador se encarga de notificar este cambio a todos los otros dependientes, en el juego el observador sería una clase o un objeto que cuando el jugador cambie por ejemplo de nivel o realice una operación este actualizaría o informaría a todo el sistema de lo ocurrido.

2.3.3.3 Patrones GRASP

No es menos cierto que la asignación de responsabilidades es muy importante a la hora del diseño de aplicaciones de cualquier tipo, esta asignación se realiza muy a menudo en el momento de preparar los diagramas de interacción. Los patrones GRASP describen una serie de principios fundamentales para la asignación de responsabilidades a objetos, expresados en forma de patrones. Por lo que es muy necesario e importante entender y poder aplicar estos principios durante la preparación de un diagrama de

interacción. A continuación se hace una breve descripción acerca de los patrones utilizados durante esta propuesta para la asignación de responsabilidades [19].

Experto:

Este patrón consiste en asignarle una responsabilidad al experto en información, en el juego cuando el usuario interactúe con la aplicación, el experto en la información seria la clase controladora, que se encargaría de gestionar el evento correspondiente a la acción del jugador.

Alta Cohesión:

Cuando se dice que una clase tiene alta cohesión, se quiere decir que el objeto tiene bien delimitadas sus responsabilidades. En la programación orientada a objetos, cada objeto tiene una responsabilidad que ha de cumplir dentro de un programa, en este caso se pretende que todas las responsabilidades de las clases del juego estén bien definidas.

En una aplicación de juego en línea este patrón estará presente en la capa de **presentación** pues al aplicar el patrón MVC cada una de las partes (Modelo, Vista, Controlador) son independientes, donde la comunicación entre ellas será mediante interfaces, que abstraen sus estructuras internas. Esto permite desarrollar estas partes de forma independiente, así como realizar modificaciones en sus partes, sin afectar a las demás.

Incluso se puede decir que en la estructura principal de la aplicación también se pone de manifiesto ya que las capas son independientes, y cualquier cambio que se realice sobre cada una de estas no afectará en nada a las demás.

Bajo acoplamiento:

El bajo acoplamiento hace referencia a las relaciones que tienen los objetos entre si dentro de un sistema. Teóricamente, cuando una serie de objetos tienen una relación con varios objetos, cuando estos últimos son cambiados, los objetos relacionados han de verse afectados necesariamente. El alto acoplamiento se da normalmente, cuando un objeto ha de saber demasiados detalles internos de otro para su funcionamiento, es decir, se rompe el encapsulamiento de otro objeto. Por ello cuando menos

acoplamiento, mejor diseñado estará el sistema y es precisamente lo que perseguimos dentro de este tipo de aplicación, ya que cualquier juego en línea encapsula una serie de objetos que tendrán que estar actualizándose a medida que transcurra el desarrollo de por ejemplo (una partida).

2.4 Metodología de desarrollo seleccionada

Se propone RUP porque es un proceso de desarrollo de software, o sea, conjunto de actividades necesarias para transformar los requisitos de un usuario en un sistema de software. Sin embargo, RUP es más que un simple proceso, es un marco de trabajo genérico que puede especializarse para una gran variedad de sistemas software, para diferentes áreas de aplicación, diferentes tipos de organización, diferentes niveles de aptitud y diferentes tamaños de proyecto. RUP está basado en componentes, lo cual quiere decir que el sistema software en construcción está formado por componentes de software interconectados a través de interfaces bien definidas.

Además, divide en fases el desarrollo del software (Inicio, Elaboración, Construcción, Transición), cada una de estas fases es desarrollada mediante un ciclo de iteraciones, la cual consiste en reproducir el ciclo de vida en cascada a menor escala. Los objetivos de una iteración se establecen en función de la evaluación de las iteraciones precedentes. Se han agrupado las actividades en grupos lógicos, se va desarrollando el software por iteraciones e incrementos, es guiado por casos de uso, los que reflejan las necesidades de los futuros usuarios, centrado en la arquitectura, la cual muestra la visión común del sistema completo. Permite que en la medida que se vaya construyendo el software por ciclos se puedan detectar errores, una particularidad de esta metodología es que, en cada ciclo de iteración, se hace exigente el uso de artefactos, siendo una de las metodologías más importantes y utilizadas para alcanzar un grado de certificación en el desarrollo del software.

El uso de esta metodología asegura que se produzca desde las primeras fases de desarrollo del software, un producto de calidad que cumpla con las características de funcionalidad, usabilidad y fiabilidad. RUP constituye la metodología estándar más utilizada para el análisis, implementación y documentación de sistemas relativamente extensos con un gran volumen y transferencia de información que es el caso de los juegos en línea.

2.5 Lenguaje y Herramienta de modelado a utilizar

Como se mencionaba en el capítulo anterior el lenguaje de modelado a utilizar será UML y como herramienta de modelado Visual Paradigm 3.4.

¿Por qué Visual Paradigm 3.4?

Porque Visual Paradigm 3.4 es una poderosa herramienta CASE que al igual que el Rational Rose utiliza UML para el modelado, es la herramienta por excelencia para ser utilizada en un ambiente de software libre. Permite crear tipos diferentes de diagramas en un ambiente totalmente visual. Es muy sencillo de usar, fácil de instalar y actualizar. Genera código para varios lenguajes. Posibilita la representación gráfica de los diagramas permitiendo ver el sistema desde diferentes perspectivas, como el de componentes, despliegue, secuencia, casos de uso, clase, actividad, estado, entre otros. A continuación se exponen otras características que este ofrece:

- Entorno de creación de diagramas para UML 2.0.
- Diseño centrado en casos de uso y enfocado al negocio que generan un software de mayor calidad.
- Uso de un lenguaje estándar común a todo el equipo de desarrollo que facilita la comunicación.
- Capacidades de ingeniería directa e inversa.
- Modelo y código que permanece sincronizado en todo el ciclo de desarrollo.
- Disponibilidad de múltiples versiones, para cada necesidad.
- Disponibilidad de integrarse en los principales IDEs (Integrated Development Environment).
- Disponibilidad en múltiples plataformas.
- Ofrecen un mecanismo general para la organización de los modelos/subsistemas/capas agrupando elementos de modelado.

- Versión gratuita (licencia para Community Edition).

2.6 Gestor de Base de Datos escogido

¿Por qué PostgreSQL?

Se escogió como Gestor de Base de Datos a PostgreSQL 8.2 primeramente porque una de las principales características que hace que este sea tan popular es precisamente que se distribuye bajo licencia GPL. Además, provee otras características y ventajas de gran importancia para el correcto funcionamiento de una aplicación de juego en línea las cuales fueron especificadas en el capítulo anterior.

2.7 Conclusiones

En este capítulo se hace una descripción del sistema a grandes rasgos para la solución de la problemática existente así como la justificación de todas las herramientas, metodología, estilo arquitectónico y patrones que se han utilizado en la presente propuesta. Se optó por el uso de la metodología RUP, utilizando Visual Paradigm como herramienta de modelado con UML como lenguaje para el modelado de los artefactos propuestos. Y finalmente la decisión se inclinó por el uso de una arquitectura en capas (3 capas).

3

Capítulo 3: Descripción de la arquitectura

3.1 Introducción

El conjunto de modelos que describen la Línea Base de la Arquitectura se denomina Descripción de la Arquitectura que será precisamente nuestro propósito durante el desarrollo de este capítulo. El papel de la descripción de la arquitectura es guiar al equipo de desarrollo a través del ciclo de vida del sistema. En este capítulo además se especifican los requerimientos no funcionales así como las diferentes vistas fundamentales que componen una arquitectura.

3.1.1 Propósito

La descripción de la arquitectura de un sistema tiene como propósito fundamental proporcionar una comprensión arquitectónica del sistema a cada uno de los miembros del proyecto mediante el uso de las distintas vistas. Además, proporciona organización durante el desarrollo del sistema en cuestión, así como una correcta evolución del mismo. Y por último y no menos importante, fomentar la reutilización.

3.1.2 Alcance

Con este capítulo se proporciona una visión general de la arquitectura de un juego en línea. La misma será extensible a todos los involucrados en el desarrollo de este tipo de aplicaciones, debido a que influye en la toma de decisiones arquitectónicas dentro de estos sistemas.

3.2 Metas y restricciones arquitectónicas

A continuación se plantean las metas y restricciones con las que se deberá cumplir a cabalidad para la correcta puesta en funcionamiento de un juego en línea, y que además son significativas para la arquitectura. Además, se hace mención de los requerimientos no funcionales así como de los requerimientos funcionales arquitectónicamente más significativos para el desarrollo de este tipo de aplicaciones.

3.2.1 Requerimientos no funcionales

En este epígrafe se analizarán los requerimientos no funcionales, los cuales son propiedades o cualidades que el producto debe tener.

Apariencia o interfaz externa.

La aplicación de juego en línea tendrá una interfaz lo más sencilla y fácil de usar posible, permitiendo que a la hora de un usuario interactuar con el sistema no tenga que depender de muchas instrucciones para interactuar con la misma, además teniendo en cuenta que el sistema podrá ser utilizado por personas que no deberán tener grandes conocimientos de computación, por otra parte, debido al constante uso diario

que podría tener el software, la interfaz debe ser agradable, que favorezca el estado de ánimo del cliente y que combine correctamente los colores, tipo de letra y tamaño y que los iconos estén en correspondencia con lo que representan.

Requerimientos de Seguridad.

Garantizar que la información sea vista únicamente por quien tiene derecho a verla. Protección contra acciones no autorizadas o que puedan afectar la integridad de los datos. Verificación sobre acciones irreversibles (eliminaciones).

Requerimientos de rendimiento.

El tiempo de respuesta de una petición al servidor deber ser rápido teniendo en cuenta que, a pesar de que este tipo de aplicaciones no opera con grandes cantidades de datos, pero si necesita de estar actualizándose constantemente ya que en muchas ocasiones será a varios clientes a la misma vez.

Requerimientos de usabilidad.

El sistema podrá ser usado por cualquier tipo de personas, la navegabilidad no debe ser muy compleja, todas las funcionalidades deben ser rápidas y fácilmente accesibles por los usuarios.

Requerimientos de Soporte.

Para el servidor de aplicaciones:

Se requiere que esté instalado un Servidor web Apache en su versión 2.x con PHP superior a la versión 5.2.8.

Para el servidor de base de datos:

Se requiere que esté instalado el gestor de base de datos PostgreSQL 8.0 o superior.

Para el cliente:

Se requiere esté instalado cualquier navegador web conocido como podría ser: Internet Explorer 6.0 o superior, Mozilla Firefox 3.6 o superior, Opera en su versión 9 o superior, Safari 4.0 o superior, o Netscape en su versión 9.0 o superior. Se requiere además que los navegadores tengan habilitado el Javascript.

Requerimientos de Portabilidad, Escalabilidad y Reusabilidad.

- El sistema será multiplataforma.
- La aplicación se construirá utilizando patrones (de diseño como los GRASP y GOF) y estándares internacionales de implementación, documentación y diseño, para facilitar su integración futura, con componentes desarrollados por cualquiera de las partes y garantizar posibilidades de mantenimiento ágil y seguro.
- Debido a los cambios en las condiciones económicas del país, las empresas cubanas toman decisiones continuas que cambian las condiciones en que se desarrollan los procesos, por lo que el sistema deberá implementar la forma de adaptarse ante el cambio de dichas condiciones.

Requerimientos de Hardware.

Para las estaciones de trabajo:

- ✓ Se requiere tengan tarjeta de red.
- ✓ Se requiere tengan al menos 256 MB de memoria RAM.
- ✓ Procesador 800 MHz como mínimo.

Para los servidores:

- Se requiere tarjeta de red.
- Se requiere tenga al menos 1GB de RAM.

- Se requiere al menos 10 GB de disco duro.
- Procesador 2.0 GHz como mínimo.

Requerimientos de Software.

- En las computadoras de los clientes se garantizará versiones de Windows 2000 o superior, así como Linux y sus correspondientes distribuciones.
- En las computadoras de los clientes solo se requiere de un navegador (Internet Explorer versión 6.0 o superior, Mozilla Firefox versión 3.6 o superior, Opera 9 o superior, Safari 4.0 o superior).

Requerimientos de redes.

Para hacer más fiable la aplicación debe de estar protegida contra fallos de corriente y de conectividad, para lo que se deberán planificar los tiempos para realizar copias de seguridad.

Requerimientos de Confiabilidad, Integridad y Disponibilidad.

- ✓ La información manejada por el sistema está protegida de acceso no autorizado y divulgación.
- ✓ La información manejada por el sistema será objeto de cuidadosa protección contra la corrupción de los datos y accesos indebidos.
- ✓ Debe garantizarse el resguardo de la información (imágenes, documentos), así como la grabación periódica (backups) de la Base de Datos, de forma tal que se posibilite la reinstalación del sistema y los datos, en caso de fallos en el sistema o en el hardware.

3.3 Representación arquitectónica

En algunos libros y artículos se han intentado capturar los detalles de la arquitectura de un sistema usando un único diagrama. Al observar detenidamente estos diagramas, se puede ver que en ellos se ha intentado representar más de un plano de lo que realmente podría expresar esta notación. Algunas de las deficiencias que estos exponen es que generalmente los nodos representan varias cosas como

programas en ejecución, partes de código fuente, computadores físicos o agrupaciones de funcionalidad y las líneas representan dependencias de compilación o flujos de control.

La arquitectura no requiere un estilo único. A veces esta tiene secuelas de un diseño donde se particionó prematuramente el software o se hizo un énfasis excesivo en la ingeniería de los datos, la eficiencia en tiempo de ejecución, o estrategias de desarrollo y organización de equipos. A menudo, la arquitectura tampoco aborda los intereses de todos sus "clientes". El modelo de 4+1 vista fue desarrollado para remediar este problema. Este describe la arquitectura del software usando cinco vistas concurrentes, donde cada vista se refiere a un conjunto de intereses de diferentes trabajadores del sistema.

3.4 Modelo 4+1 vista

El modelo 4+1 vista describe la arquitectura del software usando cinco vistas concurrentes, cada una de las cuales trata una serie de aspectos. Este modelo permite que diversas partes involucradas puedan encontrar lo que necesitan en la arquitectura de software. Los ingenieros de sistemas pueden abordar en primer lugar la vista física y, a continuación, ver el proceso; los usuarios finales, clientes y especialistas, pueden aproximarse a los datos de la vista lógica, y los directores de proyectos y miembros del equipo de configuración del software, pueden abordar desde la visión de desarrollo [17]. Véase la Fig. 7.

Estas cuatro vistas están guiadas por la vista de casos de uso que describe las funcionalidades del sistema que más inciden sobre su arquitectura.

Fig. 7 Modelo 4+1 vista.

3.4.1 Vista de Casos de Uso

Esta vista representa un subconjunto del artefacto Modelo de casos de uso y lista los casos de usos o escenarios más significativos, con las funcionalidades centrales del sistema.

Los Casos de Uso arquitectónicamente significativos, son aquellos que describen funcionalidades imprescindibles para el sistema, y que a través de estos se valida la arquitectura propuesta para el mismo.

En esta vista se representa el diagrama de casos de uso con los casos de uso arquitectónicamente significativos para el sistema y una descripción de cada uno.

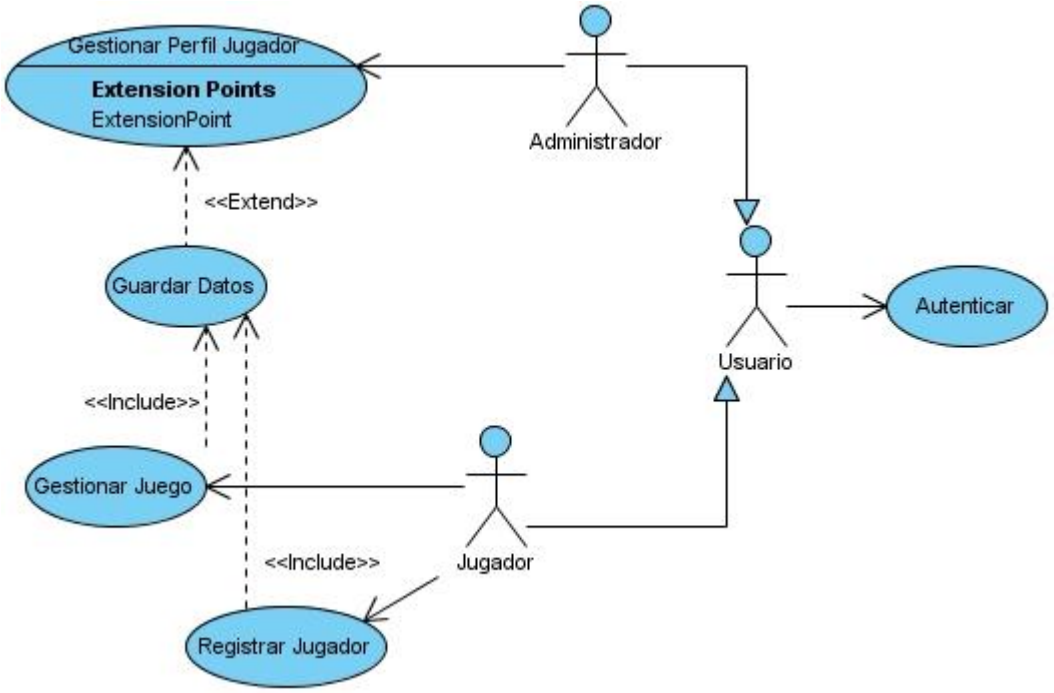

Fig. 8 Vista de Casos de Uso Arquitectónicamente significativos.

Descripción de los casos de uso arquitectónicamente más significativos:

Caso de Uso	Autenticarse
Actores:	Usuario
Resumen:	El caso de uso se inicia cuando el usuario (Administrador o Jugador) solicitan autenticarse.
Precondiciones:	El usuario tiene que estar registrado.
Referencias	
Prioridad	Crítica.

Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**
1) El usuario solicita: a) Autenticarse: Sección Autenticarse.	

Sección Autenticarse.	
Acción del Actor	**Respuesta del Sistema**
	1. El sistema muestra un formulario con los campos para la captura de datos.
2. El jugador introduce los datos de autenticación y presiona el botón	3. El sistema recoge dichos datos.

aceptar.	3.1 El sistema valida los datos. 3.2 Se entra al perfil del jugador.
Flujos Alternos	
Acción del Actor	**Respuesta del Sistema**
	Si los datos no son válidos el sistema muestra el mensaje “Datos no válidos”. Pasa a la acción 1.
Poscondiciones	El usuario accedió a su perfil.

Tabla 3 Descripción textual del Caso de Uso Autenticarse.

Caso de Uso:	Gestionar Perfil del Jugador.
Actores:	Administrador.
Resumen:	El caso de uso se inicia cuando el Administrador desea realizar algún cambio en algunos de los perfiles existentes en la base de datos de la aplicación, o que quiera eliminar alguno de ellos.
Precondiciones:	El administrador debe estar autenticado.
Referencias:	
Prioridad:	Crítica.
Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**

1)El Administrador selecciona la opción: a) Eliminar Perfil: Sección Eliminar Perfil. b) Modificar Perfil: Sección Modificar Perfil.	
Sección Eliminar Perfil.	
Acción del Actor	**Respuesta del Sistema**
	1. El sistema muestra un formulario con el listado de todos los perfiles existentes.
2. El administrador selecciona el perfil que desea eliminar y presiona el botón aceptar.	3. El sistema muestra un mensaje de confirmación de que si realmente desea eliminar el perfil.
4. El administrador confirma la eliminación de este.	4. El sistema elimina el perfil y actualiza la base de datos.
Flujos Alternos	
Sección Modificar Perfil.	
Acción del Actor	. **Respuesta del Sistema**
	1. El sistema muestra un formulario con el listado de todos los perfiles

	existentes.
2. El administrador selecciona el perfil que desea modificar y presiona el botón aceptar.	3. El sistema muestra los datos del perfil seleccionado en un formulario.
4. El administrador selecciona el campo que desea modificar y presiona el botón aceptar.	5. El sistema muestra el campo a modificar.
6. El administrador realiza los cambios necesarios en dicho campo y presiona el botón aceptar.	7. El sistema valida los cambios y actualiza el perfil en la base de datos.
Flujos Alternos	
	5.1 Si los datos no son válidos el sistema muestra el mensaje “Datos no válidos”. Pasa a la acción 5 del flujo normal de eventos.
Poscondiciones	El administrador realizó los cambios necesarios.

Tabla 4 Descripción textual del Caso de Uso Gestionar Perfil del Jugador.

Caso de Uso:	Gestionar Juego
Actores:	Jugador
Resumen:	El caso de uso se inicia cuando el jugador entra al juego.
Precondiciones:	El jugador debe estar autenticado en el sistema.

Referencias:	
Prioridad:	Critica.

Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**
1)El jugador solicita: a) Jugar: Sección Jugar. b) Ver Datos del Juego: Sección Ver Datos del Juego.	

Sección Jugar.	
Acción del Actor	**Respuesta del Sistema**
	1. El sistema muestra el juego.
2. El jugador realiza su partida.	2. El sistema va guardando los datos del juego.

Flujos Alternos	

Sección Cargar Datos del Juego.	
Acción del Actor	**Respuesta del Sistema**
	1. El sistema le muestra los datos del juego.

Flujos Alternos	
Poscondiciones	

Tabla 5 Descripción textual del Caso de Uso Gestionar Juego.

Caso de Uso:	Guardar Datos
Actores:	CU Gestionar Perfil del Jugador, CU Gestionar Juego, CU Registrar Jugador.
Resumen:	El caso de uso se inicia cuando se esté realizando alguna partida y el juego necesite guardar datos en la base de datos, cuando algún jugador se registre o que el administrador realice algún cambio sobre alguno de los perfiles existentes en la aplicación.
Precondiciones:	
Referencias:	
Prioridad:	Crítica.
Flujo Normal de Eventos	
Sección Guardar Datos.	
Acción del Actor	**Respuesta del Sistema**
1- Cuando alguno de los actores genera nuevos datos o cambia alguno de estos que deben ser registrados en la base de datos.	2. El sistema toma los datos que se generaron o cambiaron.

	3. Selecciona las tablas que se actualizaran.
	4. Actualiza los datos.
Flujos Alternos	
	1. Si se produce un error en la actualización de los datos.
	2. Se muestra un mensaje de error.
Poscondiciones	Los datos quedaron guardados en la base de datos después de cualquier cambio efectuado.

Tabla 6 Descripción textual del Caso de Uso Guardar Datos.

Caso de Uso	Registrar
Actores:	Jugador
Resumen:	El caso de uso se inicia cuando el usuario algún Jugador solicite registrarse en la aplicación.
Precondiciones:	
Referencias	
Prioridad	Crítica.
Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**

1) El usuario selecciona la opción: a) Registrarse como Nuevo: Sección Registrarse como Nuevo.	
Sección Registrarse como Nuevo.	
Acción del Actor	**Respuesta del Sistema**
	1. El sistema muestra un formulario con los campos a completar para registrarse.
2. El usuario introduce los datos correspondientes y presiona el botón aceptar.	3. El sistema valida los datos del registro. 3.1 El sistema guarda dichos datos.
Flujos Alternos	
Acción del Actor	**Respuesta del Sistema**
	3.1 Si los datos no son válidos el sistema muestra el mensaje “Datos no válidos”. Pasa a la acción 1.
Poscondiciones	El jugador queda registrado en la aplicación.

Tabla 7 Descripción textual del Caso de Uso Registrar.

3.4.2 Vista lógica

Esta vista representa un subconjunto del artefacto Modelo de Diseño, representando los elementos de diseño más importantes para la arquitectura del sistema, aquí se describen las clases más importantes, su organización en paquetes y subsistemas. Esta descripción se realiza a través de diagramas de clases para ilustrar la relación entre las clases arquitectónicamente significativas, subsistemas y paquetes.

Teniendo en cuenta, que el objetivo del trabajo que no persigue describir los elementos del diseño de una aplicación en específico, se centra en dar una vista lo más abstracta posible de cómo quedaría distribuida lógicamente una aplicación de juego en línea de tal manera que sea posible utilizarla en cualquiera de los diseños orientados a este tipo de aplicaciones, por supuesto siempre dejando claro donde se encontrará cada una de las clases involucradas en las mismas.

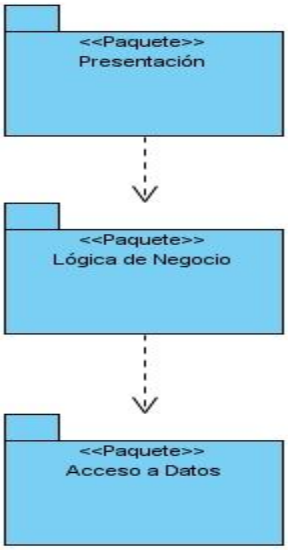

Fig. 9 Estructura en Capas.

Nombre	Estereotipo	Descripción
Presentación	Paquete	Aquí se encontrarán el conjunto de clases que componen la interfaz del sistema.
Lógica de Negocio	Paquete	Aquí estarán las clases encargadas de gestionar todos los procesos de la lógica de negocio.
Acceso a Datos	Paquete	Aquí estará ubicado el conjunto de clases que controlan el tratamiento de los datos.

Tabla 8 Descripción de los paquetes.

Estructura de la capa de Presentación.

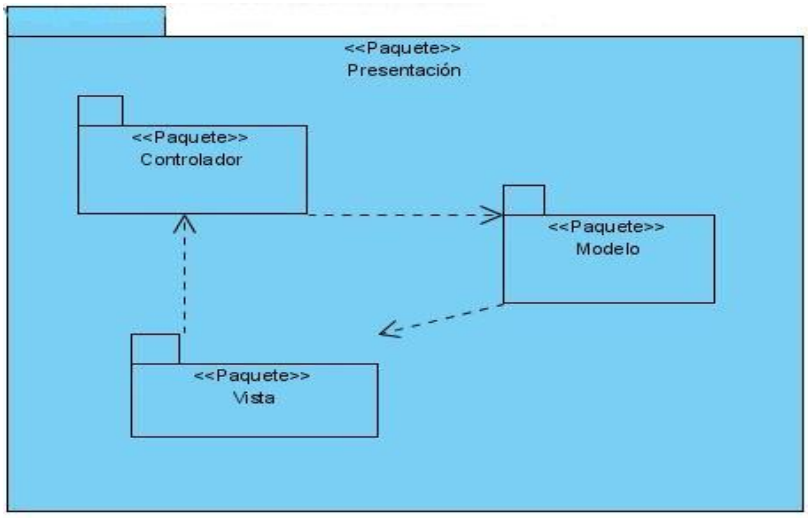

Fig. 10 Estructura lógica de la capa de presentación.

La figura anterior muestra la forma en que estará estructurada la capa de presentación haciendo uso del patrón MVC, en el paquete **Modelo** se encontrarán las clases que se encargarán de contener los datos resultantes de la ejecución de la lógica de negocio del juego, en el paquete **Vista** estarán las clases que implementan la interfaz de usuario encargadas de mostrarle el modelo al usuario en respuesta de una petición y el paquete **Controlador** contendrá las clases responsables de procesar las entradas del usuario en forma de peticiones.

3.4.3 Vista de despliegue

Esta vista suministra una base para la comprensión de la distribución física de un sistema a través de nodos. Suele utilizarse cuando el sistema está distribuido. Y hay una traza directa del modelo de implementación, puesto que cada componente físico debe estar almacenado en un nodo.

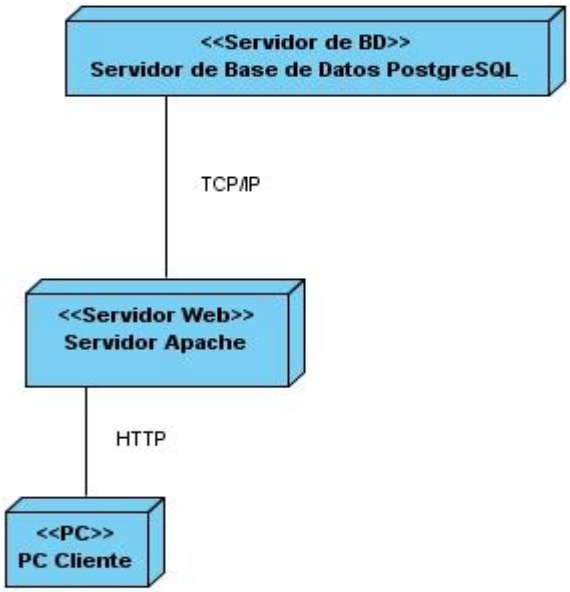

Fig. 11 Diagrama de Despliegue.

Descripción de Nodos:

Nodo PC Cliente

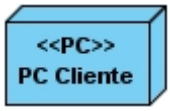

Fig. 12 PC Cliente.

- ✓ Donde estará la PC cliente.

Nodo Servidor Apache

Fig. 13 Servidor Apache.

- ✓ Donde estará el Servidor Web

Nodo Servidor de base de datos

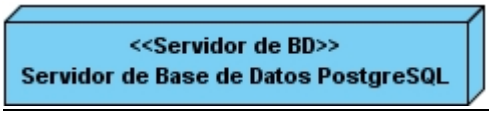

Fig. 14 Servidor de base de datos.

- ✓ Un servidor donde estará el SGBD PostgreSQL

3.4.4 Vista de implementación

En esta vista se describe la descomposición del software en capas y subsistemas de implementación. Además, la misma provee una vista de la trazabilidad de los elementos de diseño de la vista lógica ahora para la implementación.

Muy similar a la vista lógica ocurre a la hora de describir los componentes que conformarán cada uno de las capas o subsistemas de implementación dentro de esta vista, ya que no perseguimos describir la descomposición de un software en específico, sino que tratamos de dejar bien claro donde se encontrará cada uno de los componentes generados en la conformación de un juego en línea, haciéndolo lo más abstracto posible de tal forma que pueda ser utilizado en este tipo de aplicaciones.

En la figura que se muestra a continuación, se muestra la descomposición de la vista de implementación en subsistemas de implementación, notar que la relación entre los componentes no está reflejada en el mismo puesto que dicha relación depende del tipo de componente que se genere de acuerdo con el objetivo que se persiga en la aplicación.

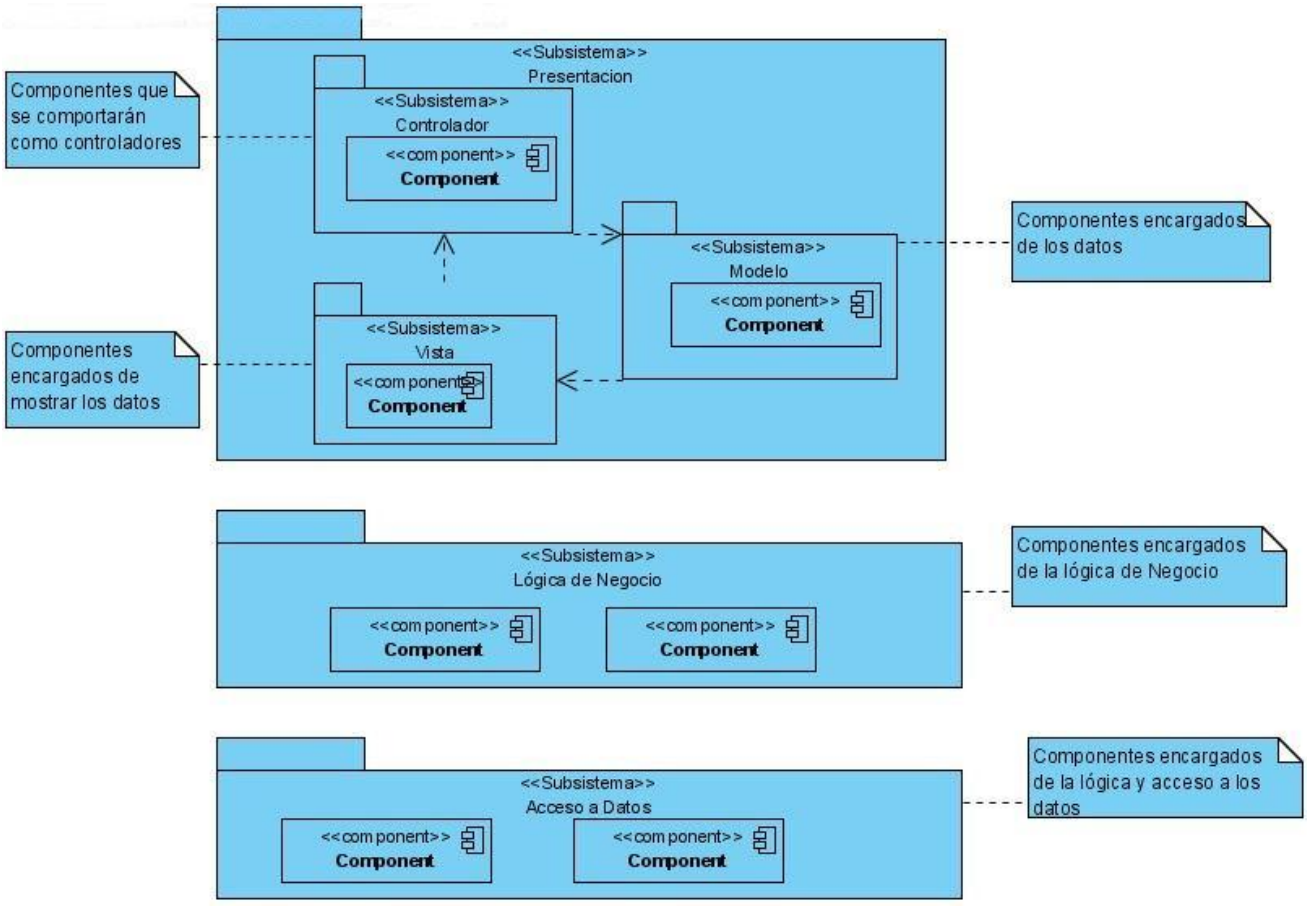

Fig. 15 Vista de implementación.

3.4.5 Vista de Procesos

Esta vista suministra una base para la comprensión de la organización de los procesos de un sistema, ilustrados en el mapeo de las clases y subsistemas en procesos e hilos. Solo suele usarse cuando el sistema presenta procesos concurrentes o hilos.

En nuestro caso la solución propuesta no presenta procesos concurrentes por tanto no se hace representación de esta vista.

3.5 Conclusiones

En este capítulo se planteó la descripción de la arquitectura detallada en el capítulo anterior, mostrándose las metas y restricciones arquitectónicas, además de la representación arquitectónica a través del modelo 4+1 vista, y de forma general dando una explicación concreta de cómo formular completamente la arquitectura propuesta para el desarrollo de un juego en línea.

4

Capítulo 4: Evaluación de la arquitectura propuesta

4.1 Introducción

Al final del desarrollo del software se conoce si éste cumplió o no con al menos un atributo de calidad que se especificaron en los requerimientos no funcionales, lo que implica tomar demasiados riesgos innecesarios. Para evitar estos riesgos es recomendable realizar evaluaciones a la arquitectura durante su diseño.

En este capítulo se describe a manera de síntesis, la importancia de la evaluación arquitectónica dentro del desarrollo de sistemas de software, sus costos y beneficios. De esta manera, se posibilita la comprensión y acercamiento a cuestiones relacionadas con el tema antes mencionado, mediante el análisis de diferentes etapas en las cuales la arquitectura puede ser evaluada, así como las técnicas y métodos del proceso.

4.2 Evaluando la Arquitectura de Software

Dentro del proceso de desarrollo de software caracterizado por su envergadura y complejidad juega la arquitectura de software un papel primordial. Es entonces en este sentido la arquitectura, vehículo para la comunicación entre las partes interesadas como de representación abstracta del sistema. Por ello y antes de evaluar determinado diseño arquitectónico resulta importante definir qué se desea evaluar y qué puede ser evaluado, pues esta contempla o excluye la mayoría de los atributos de calidad. Si la arquitectura ha sido bien diseñada, entonces, garantiza que el sistema cumpla con varios atributos de calidad, como por ejemplo confiabilidad, seguridad, etc.

4.2.1 ¿Por qué es necesario evaluar una arquitectura de software?

Evaluar una arquitectura de software sirve para prevenir todos los posibles desastres de un diseño que no cumple con los requerimientos de calidad y para saber que tan adecuada es la arquitectura de software diseñada para el sistema. Una evaluación de una arquitectura no da un sí o un no, si es buena o mala, o una calificación, expresa donde está el riesgo, es decir, fortalezas y debilidades identificadas de la arquitectura.

Después de la evaluación de una arquitectura de software, se pueden tomar algunas decisiones como: si se puede seguir el proyecto con las áreas de debilidad dadas en la evaluación, si hay que reforzar la arquitectura de software o si hay que comenzar de nuevo toda la arquitectura [28].

“El propósito de realizar evaluaciones a la arquitectura, es para analizar e identificar riesgos potenciales en su estructura y sus propiedades, que puedan afectar al sistema de software resultante, además de verificar que los requerimientos no funcionales estén presentes en la arquitectura, así como determinar en qué grado se satisfacen los atributos de calidad. Cabe señalar que los requerimientos no funcionales también son llamados atributos de calidad” [32].

4.2.2 Atributos de calidad

Los atributos de calidad pueden definirse como aquellas características deseadas en determinado sistema, definen las propiedades de los servicios que se brindan. Se pudiera entonces y de acuerdo con este criterio considerar la calidad de un sistema de software teniendo en cuenta la combinación de dichos atributos, los cuales se clasifican en dos grupos fundamentales [29]:

- **Observables vía ejecución.**

  Se determina del comportamiento del sistema en tiempo de ejecución.

  - Disponibilidad: Medida de disponibilidad del sistema para el uso.
  - Confidencialidad: Es la ausencia de acceso no autorizado a la información.
  - Funcionalidad: Habilidad del sistema para realizar el trabajo para el cual fue concebido.
  - Desempeño: Es el grado en el cual un sistema o componente cumple con sus funciones, dentro de ciertas restricciones dadas, como velocidad, exactitud o uso de memoria.
  - Confiabilidad: Es la medida de la habilidad de un sistema a mantenerse operativo a lo largo del tiempo.
  - Seguridad Externa: Ausencia de consecuencias catastróficas en el ambiente. Es la medida de ausencia de errores que generan perdida de información.
  - Seguridad Interna: Es la medida de la habilidad del sistema para resistir a intentos de uso no autorizados y negación de servicio, mientras se sirve a usuarios legítimos.

- **No observables vía ejecución.**

  Aquellos atributos que se establecen durante el desarrollo del sistema.

- Configurabilidad: Posibilidad que se otorga a un usuario experto a realizar ciertos cambios al sistema.
- Integrabilidad: Es la medida en que trabajan correctamente componentes del sistema que fueron desarrollados separadamente al ser integrados.
- Integridad: Es la ausencia de alteraciones inapropiadas de la información.
- Interoperabilidad: capacidad del producto para interactuar con más sistemas.
- Modificabilidad: Es la habilidad de realizar cambios futuros al sistema.
- Mantenibilidad: Es la capacidad de someter a un sistema a reparaciones y evolución.
- Portabilidad: Es la habilidad del sistema para ser ejecutado en diferentes ambientes de computación. Estos ambientes pueden ser hardware, software o una combinación de los dos.
- Reusabilidad: Es la capacidad de diseñar un sistema de forma tal que su estructura o partes de sus componentes puedan ser reutilizados en futuras aplicaciones.
- Escalabilidad: Es el grado con el que se pueden ampliar el diseño arquitectónico, de datos o procedimental.
- Capacidad de prueba: Es la medida de la facilidad con la que el software, al ser sometido a una serie de pruebas, puede demostrar sus fallas. Es la probabilidad de que, asumiendo que tiene al menos una falla, el software fallará en su próxima ejecución de prueba.

Modelo de calidad ISO/IEC 9126.

Varias han sido las propuestas de modelos de calidad desde 1970 hasta la actualidad, donde se encuentra el modelo ISO/IEC 9126. Esta plantea que la calidad de un software debe realizarse sobre la base de sus atributos, tanto internos como externos. Es decir, la calidad interna, que trata la estructuración

de las propiedades del software, influye directamente sobre la externa, o sea, cualidades observables sin conocer como está construido el producto.

Estos atributos se expresan en un conjunto de características bien definidas en la norma ya mencionada:

Funcionalidad:

- Educación: capacidad del producto de software para proporcionar un conjunto de funciones para tareas específicas.
- Interoperabilidad: capacidad del producto para interactuar con más sistemas.
- Seguridad de acceso: capacidad para proteger la información y los datos.

Fiabilidad:

- Madurez: capacidad del producto para evitar fallas.
- Tolerancia a fallos: capacidad de mantener un nivel de prestaciones en caso de fallos.
- Capacidad de recuperación: capacidad del software para restablecer las prestaciones y recuperar los datos dañados.

Usabilidad:

- Capacidad para ser entendido: capacidad que permite que el usuario pueda entender si el producto es el adecuado.
- Capacidad para ser aprendido: capacidad que permite al usuario operar y controlar el software.
- Capacidad de atracción: capacidad del producto de ser atractivo al usuario.

Eficiencia:

- Comportamiento temporal: capacidad del producto para proporcionar tiempos de respuestas bajo condiciones determinadas.

- Utilización de recursos: capacidad del software para utilizar la cantidad y tipo de recurso más adecuado.

Mantenibilidad:

- Capacidad para ser analizado: capacidad del producto para diagnosticar deficiencias o fallos.
- Capacidad de ser cambiado: capacidad de permitir hacerle modificaciones.
- Estabilidad: capacidad del producto para evitar efectos inesperados.
- Capacidad para ser probado: capacidad que debe permitir que el software modificado sea válido.

Portabilidad

- Adaptabilidad: capacidad del producto para adaptarse a diferentes entornos.
- Instalabilidad: capacidad del producto de ser instalado en un entorno determinado.
- Coexistencia: capacidad que tiene el producto de poder coexistir con otros software en un entorno determinado.
- Capacidad para remplazar: capacidad que tiene el producto para remplazar a otro software.

4.2.3 ¿Cuándo una Arquitectura puede ser evaluada?

Una arquitectura de software puede ser evaluada en cualquier momento, partiendo siempre de que haya sido especificada con anterioridad al menos de manera parcial. Existen dos variaciones útiles para realizar esta evaluación, la evaluación temprana y la evaluación tardía [26].

La evaluación temprana

Para realizar esta evaluación no es necesario que la arquitectura se encuentre completamente especificada. Esta evaluación permite efectuar decisiones sobre la arquitectura en cualquier nivel, puesto

que se pueden imponer cambios arquitectónicos producto de una evaluación en función de los atributos de calidad esperados.

La evaluación tardía

Se realiza cuando la arquitectura del sistema se encuentra establecida y se ha terminado su implementación, es decir, en el momento de adquisición de un sistema ya terminado. Se considera por parte de los autores que la evaluación en este punto es muy importante y útil, puesto que puede observarse el cumplimiento de los atributos de calidad asociados al sistema y cómo será su comportamiento general.

La evaluación de la arquitectura de software debe realizarse cuando esta contiene suficientes elementos como para justificarla. Un buen momento para determinar cuando realizar la evaluación es cuando el equipo de desarrollo comienza a tomar decisiones que dependen de la arquitectura y que el costo de no tenerlas en cuenta son mayores que el costo de realizar una evaluación. Para la evaluación de la arquitectura se utilizó la evaluación temprana.

4.2.4 Resultado de la evaluación

La evaluación de la arquitectura produce un informe, el cual varía en dependencia del método utilizado. Este informe produce respuesta a dos tipos de preguntas.

- ¿Es esta arquitectura adecuada para el sistema para el cual fue diseñada?
- ¿Cuál de las arquitecturas propuestas es la más adecuada para el sistema?

Una arquitectura es adecuada cuando cumple dos criterios

- El sistema resultante cumple con los atributos de calidad.
- El sistema puede ser construido con los recursos disponibles, es decir, es construible.

La evaluación de la arquitectura no produce resultados cuantitativos, no es de interés conocer la cantidad de transacciones por segundos debido a que el sistema aún no está implementado. Lo que interesa es aprender como un atributo de calidad es afectado por una decisión de diseño arquitectónico.

4.2.5 ¿Por qué cualidades puede ser evaluada una arquitectura?

Los atributos de calidad son requerimientos adicionales del sistema que hacen referencia a características que éste debe satisfacer.

Algunos de los atributos por los cuales puede ser evaluada una arquitectura son:

- Disponibilidad (Availability). Es la medida de disponibilidad del sistema para el uso.
- Desempeño (Performance). Es el grado en el cual un sistema o componente cumple con sus funciones designadas, dentro de ciertas restricciones dadas, como velocidad, exactitud o uso de memoria. (IEEE 610.12).
- Confiabilidad (Reliability). Es la medida de la habilidad de un sistema a mantenerse operativo a lo largo del tiempo
- Seguridad (Security). Es la mediada de la habilidad del sistema para resistir a intentos de uso no autorizados y negación de servicios, mientras se sirve a usuarios legítimos.
- Portabilidad (Portability). Es la habilidad del sistema de ser ejecutado en diferentes ambientes de computación. Estos ambientes pueden ser hardware, software o una combinación de los dos.

4.3 Técnicas de evaluación

Existen varias técnicas que permiten realizar evaluaciones a la arquitectura, estas se clasifican en cualitativas y cuantitativas.

En las técnicas de evaluación cualitativas se pueden utilizar escenarios, cuestionarios o listas de verificación. Mientras que en las técnicas de evaluación cuantitativas se pueden emplear métricas, simulaciones, prototipos, experimentos o modelos matemáticos [28].

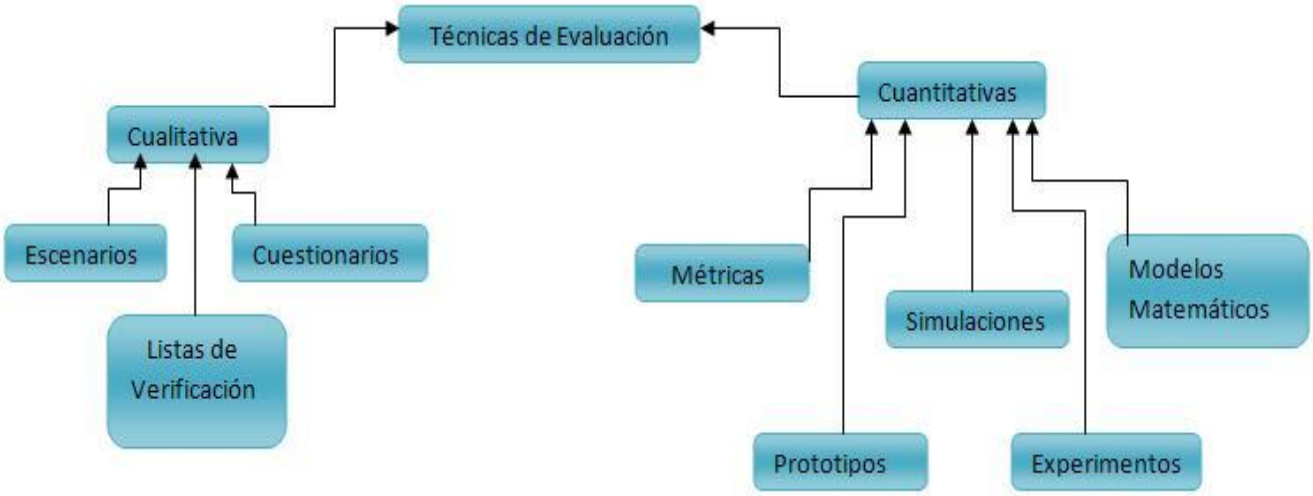

Fig. 16 Técnicas de evaluación.

La mayoría de los métodos de evaluación utilizan escenarios, que son secuencias específicas de pasos que involucran el uso o la modificación del sistema. Por lo regular, las técnicas de evaluación cualitativas son usadas cuando la arquitectura se encuentra en construcción, mientras que las técnicas de evaluación cuantitativas, se usan cuando la arquitectura ya ha sido implantada.

4.4 Métodos de Evaluación de Arquitectura de Software

4.4.1 SAAM (Software Architecture Analysis Method)

El Método de Análisis de Arquitecturas de Software (Software Architecture Analysis Method, SAAM) es un método basado en escenarios que permite la evaluación de atributos de calidad (portabilidad, modificabilidad, extensibilidad, integrabilidad) y la relación del diseño arquitectónico con los requerimientos del sistema. SAAM posibilita evaluar múltiples arquitecturas aunque en ningún caso emite un valor absoluto referente a la calidad del sistema. Los pasos que sigue este método son los siguientes [30]:

- Desarrollo de escenarios.
- Descripción de la arquitectura.

- Clasificación y asignación de prioridad de los escenarios.
- Evaluación individual de los escenarios indirectos.
- Evaluación de la interacción entre escenarios.
- Creación de la evaluación global.

4.4.2 ATAM (Architecture Trade-Off Analysis Method)

El Método de Análisis de Acuerdos de Arquitectura (Architecture Trade-off Analysis Method, ATAM) revela la forma en que una arquitectura satisface ciertos atributos de calidad, provee una visión de cómo estos atributos interactúan con otros. ATAM se inspira en tres aéreas distintas (estilos arquitectónicos, análisis de atributos de calidad, y el método de evaluación SAAM) [27]. El método de evaluación ATAM comprende nueve pasos, agrupados en cuatro fases:

- **Fase 1: Presentación**
  - Presentación del ATAM.
  - Presentación de las metas de negocio.
  - Presentación de la Arquitectura.
- **Fase 2: Investigación y análisis**
  - Identificación de los enfoques arquitectónicos.
  - Generación de Utility Tree.
  - Análisis de los enfoques arquitectónicos.
- **Fase 3: Pruebas**
  - Lluvia de ideas y establecimiento de prioridad de escenarios.

- Análisis de los enfoques arquitectónicos.

➢ **Fase 4: Reporte**

- Presentación de los resultados.

4.4.3 ARID (Active Reviews for Intermediate Designs)

El método de Revisiones Activas para Diseños Intermedios (Active Reviews for Intermediate Designs, ARID) permite realizar evaluaciones a diseños parciales en etapas tempranas dentro del desarrollo del proyecto. Tanto ADR (Active Design Review) como ATAM (Architecture Trade-off Analysis *Method*) proveen características útiles para el problema de la evaluación de diseños preliminares, dado que ninguno por sí solo es conveniente. El método ARID comprende nueve pasos agrupados en dos fases [31]:

➢ **Fase 1: Actividades Previas**

- Identificación de los encargados de la revisión.
- Preparar el informe de diseño.
- Preparar los escenarios base.
- Preparar los materiales.

➢ **Fase 2: Revisión**

- Presentación del ARID.
- Presentación del diseño.
- Lluvia de ideas y establecimiento de prioridad de escenarios.
- Aplicación de los escenarios.

- Resumen.

4.4.4 Comparación entre Métodos de Evaluación

A continuación en la Tabla 9 se hace una conparación entre los métodos de evaluación SAAM, ATAM y ARID [32].

	ATAM	**SAAM**	**ARID**
Atributos de Calidad Contemplados	Modificabilidad Seguridad Confiabilidad Desempeño	Modificabilidad Funcionabilidad	Conveniencia del diseño evaluado
Objetos Analizados	Estilos Arquitectónicos, Documentación, Flujo de Datos y Vistas Arquitectónicas	Documentación, y Vistas Arquitectónicas	Especificación de los componentes
Etapas del Proyecto en las que se Aplica	Luego que el diseño de la arquitectura ha sido establecido	Luego que la arquitectura cuenta con funcionalidad ubicada en módulos	A lo largo del diseño de la arquitectura
Enfoques Utilizados	Árbol de Utilidad y lluvia de ideas para articular los requerimientos de calidad.	Lluvia de ideas para escenarios y articular los requerimientos de	Revisiones de diseño, lluvia de ideas para obtener

	Análisis arquitectónico que detecta puntos sensibles, puntos de balance y riesgos.	calidad. Análisis de los escenarios para verificar funcionalidad o estimar el costo de los cambios.	escenarios.

Tabla 9 Comparación entre Métodos de Evaluación.

4.5 Evaluación de la arquitectura de software propuesta

Con vista a evaluar el diseño arquitectónico propuesto, se decide utilizar el método ARID, ya que permite realizar la evaluación de la arquitectura de diseños parciales en etapas tempranas del desarrollo, por otro lado, es sencillo de utilizar y el costo de evaluación es realmente bajo.

Para ello se seleccionaron algunos atributos de calidad enmarcados en diferentes escenarios de acuerdo con un perfil específico. Cada escenario, aparece vinculado a varios atributos los cuales pueden ser afectados por decisiones entorno al diseño. Los atributos que se analizarán para esta estrategia de evaluación temprana son los no observables vía ejecución.

4.5.1 Metas que se persiguen

Facilitar la comunicación entre los interesados en el proceso de desarrollo del software, así como definir una estructura que soporte de manera aceptada los procesos que se desarrollan dentro del marco del proyecto.

A continuación se detalla con claridad cada uno de los atributos de calidad seleccionados para la evaluación de la arquitectura propuesta, además del perfil y escenario donde estos se enmarcan. Aclarar que teniendo en cuenta la comparación realizada en la Tabla 9 entre los métodos de evaluación de

arquitecturas, se determinó que la selección de estos atributos de calidad se realiza a conveniencia de los evaluadores.

Atributo de calidad	Perfil	Escenario
Mantenibilidad	Mantenimiento	Migración del Sistema Gestor de Base de Datos
Relación atributo-escenario		
La **Mantenibilidad** es un atributo de calidad que se establece durante el proceso de desarrollo. Este atributo se refiere a la capacidad con que cuenta el sistema para enfrentar reparaciones y evolucionar. En este sentido, situándonos en el escenario antes planteado y de acuerdo con el diseño en capas con funcionalidades claramente definidas, se asegura de ser necesaria la migración con carácter temporal o definitivo de PostgreSQL a otro SGBD, para lo cual solo tendrían que efectuarse cambios dentro de la capa de acceso a datos, en la clase encargada de gestionar la conexión con el sistema gestor de Base de Datos.		

Tabla 10 Evaluando el Atributo de Calidad Mantenibilidad.

Atributo de calidad	Perfil	Escenario
Integridad	Integridad	Acceso a Datos
Relación atributo-escenario		
En los sistemas informáticos, un atributo de suma importancia es la **Integridad** de la información. Con el objetivo de impedir cualquier tipo de alteración en la misma, se definen durante el proceso de desarrollo diferentes estrategias. En este sentido, el sistema posibilita, teniendo en cuenta el diseño e implementación de la base de datos, qué usuario, de acuerdo con el rol que desempeña accede a		

una determinada información. De esta manera, y en estrecha relación con la seguridad interna entre base de datos se asegura la integridad de los datos almacenados, un ejemplo claro de esto es el Administrador del sistema el cual puede realizar cambios dentro de este, en cambio, el usuario que se loguearse como jugador no tiene este privilegio.

Tabla 11 Evaluando el Atributo de Calidad Integridad.

Atributo de calidad	**Perfil**	**Escenario**
Configurabilidad	Configuración	Gestionar Perfil Usuario
Relación atributo-escenario		
En la capa de presentación se gestiona la autenticación de los usuarios, teniendo en cuenta la forma en que estos se autentiquen en la aplicación serán los permisos que le son concedidos a los mismos dentro de esta. A partir de que la **Configurabilidad** se refiere a la posibilidad que tiene un usuario a realizar cambios en el sistema, cuando este se loguea como administrador tiene permisos suficientes para modificar, eliminar y crear nuevos perfiles en la aplicación.		

Tabla 12 Evaluando el Atributo de Calidad Configurabilidad.

Atributo de calidad	**Perfil**	**Escenario**
Escalabilidad	Ampliación	Base de Datos
Relación atributo-escenario		
Teniendo en cuenta que la **Escalabilidad** es el grado con el que se puede ampliar el diseño arquitectónico de datos, el gestor de bases de datos brinda la posibilidad de formar clústeres de bases de datos. Esto permite que se puedan agregar más servidores de bases de datos en caso que		

el espacio para la persistencia de los datos que se manejan en la aplicación este agotándose.

Tabla 13 Evaluando el Atributo de Calidad Escalabilidad.

Atributo de calidad	**Perfil**	**Escenario**
Portabilidad	Portabilidad	Migración de Sistema Operativo
Relación atributo-escenario		
Partiendo de que la **Portabilidad** es la habilidad del sistema para ser ejecutado en diferentes ambientes de computación. Estos ambientes pueden ser hardware, software o una combinación de los dos, este sistema podrá ser ejecutado en cualquiera de los sistemas operativos Windows, GNU/Linux, Mac, etc, lo antes expuesto demuestra que el mismo es escalable en gran medida.		

Tabla 14 Evaluando el Atributo de Calidad Portabilidad.

Atributo de calidad	**Perfil**	**Escenario**
Integrabilidad	Integrabilidad	Integración de los Componentes.
Relación atributo-escenario		
Teniendo en cuenta que la **Integrabilidad** es la medida en que trabajan correctamente componentes del sistema que fueron desarrollados separadamente al integrarlos, en este sistema la integración no tendrá problemas ya que todos los programadores utilizarán un mismo estándar de codificación.		

Tabla 15 Evaluando el Atributo de Calidad Integrabilidad.

Atributo de calidad	Perfil	Escenario
Modificabilidad	Modificabilidad	Realizar cambios al sistema.
Relación atributo-escenario		
Partiendo de que la **Modificabilidad** es la habilidad de realizar cambios futuros al sistema, a este sistema se le realizarán los cambios que sean necesarios en un futuro siempre que sean necesarios para el bien funcionamiento de la aplicación.		

Tabla 16 Evaluando el Atributo de Calidad Modificabilidad.

Atributo de calidad	Perfil	Escenario
Reusabilidad	Reusabilidad	Reutilizar código.
Relación atributo-escenario		
Partiendo de que la **Reusabilidad** es la capacidad de diseñar un sistema de forma tal que su estructura o partes de sus componentes puedan ser reutilizados en futuras aplicaciones, en este sistema la reutilización es un factor fundamental, ya que se puede reutilizar algunos componentes en otras aplicaciones del mismo tipo de la nuestra.		

Tabla 17 Evaluando el Atributo de Calidad Reusabilidad.

Atributo de calidad	Perfil	Escenario
Interoperabilidad	Interoperabilidad	Interoperabilidad con otros sistemas.

Relación atributo-escenario
Teniendo en cuenta que la **Interoperabilidad** es la medida de la habilidad de que un grupo de partes del sistema trabajen con otro sistema, con este sistema lo que se quiere es que sea reutilizable, es decir, que se pueda utilizar con otros sistemas de juegos en línea, por lo que todos los cambios que se hagan en el sistema van a estar en función de esto.

Tabla 18 Evaluando el Atributo de Calidad Interoperabilidad.

A modo de resumen y analizando lo anteriormente ilustrado, pudiera inferirse que el sistema es mantenido valorando la relativa independencia con determinado SGBD. Su integridad ante las restricciones y políticas de seguridad en el acceso a la información almacenada. Teniendo en cuenta además que en la configurabilidad el usuario puede realizar cambios en el sistema, cuando este se loguea como administrador tiene permisos suficientes para modificar, eliminar y crear nuevos perfiles en la aplicación. En cuanto a la escalabilidad se puedan agregar más servidores de bases de datos en caso que el espacio para la persistencia de los datos que se manejan en la aplicación este agotándose. Teniendo en cuenta su portabilidad el sistema podrá ser ejecutado en cualquiera de los sistemas operativos. No habrá problemas con la integrabilidad de sus componentes, mucho menos con realizar cambios futuros en el sistema, ni tampoco con reutilizar alguna estructura o componentes en algún otro sistema parecido, y el sistema se podrá integrar con otros sistemas de juegos en línea.

Como resultado de la combinación de los atributos de calidad contemplados en el diseño arquitectónico evaluado de manera parcial, se puede concluir que la arquitectura propuesta es adecuada ya que como se mencionaba anteriormente:

Una arquitectura es adecuada cuando cumple dos criterios:

- **El sistema resultante cumple con los atributos de calidad**, que es lo analizado anteriormente.
- **El sistema puede ser construido con los recursos disponibles, es decir, es construible**, en este caso los recursos y las condiciones están creadas dentro del proyecto.

4.6 Conclusiones

En este capítulo se analizó la importancia de evaluar la arquitectura de software, los costos en la realización del proceso y los beneficios que de él se obtienen. Se analizaron además, las técnicas y métodos para la evaluación arquitectónica. Y por último se evaluó de manera parcial, en una etapa temprana, el diseño arquitectónico propuesto.

Conclusiones Generales

El desarrollo de la arquitectura de software es una de las etapas fundamentales en el desarrollo de software, pues es aquí donde los profesionales aportan todos sus conocimientos, creatividad y experiencia para crear la mejor propuesta de solución que se dará al cliente que cumpla con los requerimientos funcionales y no funcionales establecidos para el sistema en desarrollo, así como sus preocupaciones principales de lo que se espera del sistema. Es de vital importancia que todo sistema de software esté respaldado por una arquitectura sólida que facilite el entendimiento del mismo, que sea capaz de organizar el desarrollo, fomentar la reutilización y hacer evolucionar el sistema.

Luego de culminada la realización del presente trabajo y analizado todo el contenido expuesto anteriormente, se arriban a las siguientes conclusiones:

- La arquitectura por capas, implementando el patrón modelo-vista-controlador en la capa de presentación, es un diseño arquitectónico que cumple con los requerimientos de una aplicación de juego en línea, además de que provee gran flexibilidad a este tipo de aplicaciones.

- Para la evaluación de la propuesta se utilizó el método de evaluación de arquitecturas ARID. Este método permitió evaluar determinados atributos de calidad no observables vía ejecución, requeridos para aplicaciones de juego en línea. En esta evaluación se llegó a la conclusión de que el diseño arquitectónico propuesto permite el cumplimiento de los requerimientos de una aplicación de juego en línea.

Recomendaciones

De forma general, el diseño arquitectónico propuesto, describe toda una serie de aspectos significativos referentes al futuro desarrollo de una aplicación de juegos en línea. No obstante, como parte de un proceso de mejoras continuas, se proponen las siguientes recomendaciones:

- Desarrollar una aplicación de juego en línea a través de la arquitectura propuesta.
- Mantener esta propuesta en constante refinamiento.
- Una vez implementada una aplicación de juego en línea, someterla a evaluaciones a través de métodos que se utilizan para evaluar una arquitectura después de su implementación.

Referencias Bibliográficas

[1] Dijkstra, Edsger. *The Structure of the THE Multiprogramming system.* s.l. : Communications of the ACM, 1983.

[2] Brooks Jr, Frederick. *The mythical man-month.* s.l. : Addison-Wesley, 1975.

[3] Perry, Dewayne and Wolf, Alexander. *Foundations for the study of software architecture.* s.l. : ACM SIGSOFT Software Engineering Notes, 1992.

[4] Jacobson, Ivar, Booch, Grady y Rumbaugh, James. *El Proceso Unificado de Desarrollo de Software.* La Habana : Editorial Félix Varela, 2004. ISBN: 978-84-7829-036-9.

[5] IEEE. (2000). "IEEE Std 1471-2000 IEEE Recommended Practice for Architectural Description of Software-Intensive Systems -Description." from http://standards.ieee.org/reading/ieee/std_public/description/se/1471-2000_desc.html

[6] Shaw, M. (1996). Introduction to Software Architectures New perspectives on an emerging discipline.

[7] Buschmann, F., R. Meunier, et al. (1996). Pattern – Oriented Software Architecture. A System of Patterns.

[8] Alexander, C., S. Ishikawa, et al. (1977). A Pattern Language: Towns, Buildings, Construction.

[9] Salvador, Juan y Garrido, Castejón. Arquitectura y diseño de sistemas web modernos

[10] FREE DOWNLOAD MANAGER. 2007. Visual Paradigm for UML . [En línea] 2007. [Citado el: 13 de 2 de 2009.] http://www.freedownloadmanager.org/es/downloads/Paradigma_Visual_para_UML_(M%C3%8D)_14720_p.

[11] ATI. *Asociación de Técnicos de Informática.* 2009. [Citado el: 9 de Abril de 2009.] http://www.ati.es/spip.php?article1136.

[12] Letelier, Patricio y Mª Carmen, Penadés. Métodologías ágiles para el desarrollo de software: eXtreme Programming (XP). 2003. [Citado el: 9 de Abril de 2009.] http://www.willydev.net/descargas/masyxp.pdf.

[13] IBM. Rational Rose. [Citado: 03 06, 2009] Rational Rose. Linea de productos. Disponible en: http://www-01.ibm.com/software/awdtools/developer/rose/

[14] Reynoso Billy, Carlos. *Introducción a la Arquitectura de Software.* Buenos Aires : s.n., 2004.

[15] Blasi, Emanuel. Resumen de Patrones de Diseño.

[16] Catalani, Exequiel. [En Línea] 20.08.2007. [Citado: 06.06.2009]. Arquitectura Modelo/Vista/Controlador. Disponible en: http://exequielc.wordpress.com/2007/08/20/arquitectura-modelovistacontrolador/.

[17] Kruchten, Philippe. Planos Arquitect´onicos: El Modelo de "4+1" Vistas de la Arquitectura del Software.

[18] Canales Mora, Roberto. Patrones GRASP. Madrid: s.n., 2006. Disponible en: http://www.adictosaltrabajo.com/tutoriales/pdfs/grasp.pdf

[19] Visconti, Marcello y Astudillo, Hernán. Fundamentos de Ingeniería de Software. Disponible en: http://www.inf.utfsm.cl/~visconti/ili236/Documentos/16-PatronesGRASP.pdf

[20] Cueva Lovelle, **Juan Manuel. Introduccion a UML**, Disponible en: http://gidis.ing.unlpam.edu.ar/downloads/pdfs/IntroduccionUML.PDF

[21] R. E. Johnson, and B. Foote en 1988 en su publicación "Designing Reusable Classes (R. E. JOHNSON, B. FOOTE 1988).

[22] Mora, Francisco. UML: Lenguaje Unificado de Modelado. *DCCIA, Universidad de Alicante, 2002.*

[23] PECOS, Daniel. 2009. PostgreSQL vs. MySQL. [En línea] 2009. [Citado el: 20 de 3 de 2009.] http://www.netpecos.org/docs/mysql_postgres/index.html.

[24] Billy Reinoso, Carlos y Kicillof, Nicolás. 2004. *Estilos y Patrones en la Estrategia de Arquitectura de Microsoft.* [doc] Buenos Aires : Universidad de Buenos Aires, 2004.

[25] Garlan, David y Shaw Mary. An Introduction to Software Architecture. (1994)

[26] Kazman, R., M. Klein, et al. (2000). ATAM: Method for Architecture Evaluation.

[27] Carrascoso Puebla, Yoan Arlet., Chaviano Gómez Enrique y Céspedes Vega, Anisleydi, Procedimiento para la Evaluación de Arquitecturas de Software basadas en Componentes. (2009). [Citado el: 4 de Mayo de 2010.] http://www.gestiopolis.com/administracion-estrategia/procedimiento-para-la-evolucion-de-las-arquitecturas-de-software.htm

[28] Carlos Pelaez, Juan. Definiciones - Atributos de Calidad para Aplicaciones Distribuidas y de Alta Disponibilidad. (2009). [Citado el: 4 de Mayo de 2010.] http://geeks.ms/blogs/jkpelaez/archive/2009/05/29/definiciones-atributos-de-calidad-para-aplicaciones-distribuidas-y-de-alta-disponibilidad.aspx

[29] Gregory Abowd, Len Bass, Rick Kazman, Mike Webb (Texas Instruments). SAAM: A Method for Analyzing the Properties of Software Architectures. (2007)

[30] Paul C. Clements. Active Reviews for Intermediate Designs. (2009).

[31] Gómez, Omar Salvador Gómez. *Evaluando Arquitecturas de Software. Parte 1. Panorama General.* 01, México : Brainworx S.A, 2007. 1870-0888.

[32] Erika Camacho, Fabio Cardeso, Gabriel Nuñez. *Arquitecturas de Software.* 2004.

Bibliografía Consultada

Jacobson, Ivar, Booch, Grady y Rumbaugh, James. *El Proceso Unificado de Desarrollo de Software.* La Habana : Editorial Félix Varela, 2004. ISBN: 978-84-7829-036-9.

Garlan, David y Shaw Mary. An Introduction to Software Architecture. (1994)

Salvador, Juan y Garrido, Castejón. Arquitectura y diseño de sistemas web modernos

Blasi, Emanuel. Resumen de Patrones de Diseño.

IBM. Rational Rose. [Citado: 03 06, 2009] Rational Rose. Linea de productos. Disponible en: http://www-01.ibm.com/software/awdtools/developer/rose/

Catalani, Exequiel. [En Línea] 20.08.2007. [Citado: 06.06.2009]. Arquitectura Modelo/Vista/Controlador. Disponible en: http://exequielc.wordpress.com/2007/08/20/arquitectura-modelovistacontrolador/.

Visconti, Marcello y Astudillo, Hernán. Fundamentos de Ingeniería de Software. Disponible en: http://www.inf.utfsm.cl/~visconti/ili236/Documentos/16-PatronesGRASP.pdf

Carlos Pelaez, Juan. Definiciones - Atributos de Calidad para Aplicaciones Distribuidas y de Alta Disponibilidad. (2009). [Citado el: 4 de Mayo de 2010.] http://geeks.ms/blogs/jkpelaez/archive/2009/05/29/definiciones-atributos-de-calidad-para-aplicaciones-distribuidas-y-de-alta-disponibilidad.aspx

Paul C. Clements. Active Reviews for Intermediate Designs. (2009).

Gómez, Omar Salvador Gómez. *Evaluando Arquitecturas de Software. Parte 1. Panorama General.* 01, México : Brainworx S.A, 2007. 1870-0888.

Glosario de Términos y Siglas

Términos

"A"

Actividad: conjunto de operaciones o tareas propias de una persona o entidad que permite que el trabajo a realizar sea descrito y entendido de manera precisa por aquellos que tienen que ejecutarlo.

Aplicaciones web: aplicaciones que los usuarios pueden utilizar accediendo a un servidor web a través de Internet o de una Intranet mediante un navegador.

Arquitectura: es el arte de construir, de acuerdo con un programa y empleando los medios diversos de que se dispone en cada época, tiene un fundamento científico y obedece a una técnica compleja.

Arquitectura de software: es la organización fundamental de un sistema encarnada en sus componentes, las relaciones entre ellos y el ambiente y los principios que orientan su diseño y evolución.

Artefacto: es un término general, para cualquier tipo de información creada, producida, cambiada o utilizada por los trabajadores en el desarrollo del sistema.

"C"

Calidad: conjunto de propiedades y características de un producto o servicio que le confieren su aptitud para satisfacer unas necesidades explícitas o implícitas.

Capa de presentación: generalmente se identifica como la capa web. Esta capa debe ser tan fina como sea posible. Además, debe permitir diferentes capas de presentación, tales como una capa web y/o fachadas de servicios web remotos, sobre una simple y bien diseñada capa de negocio.

Capa de negocio: es la capa responsable de delimitar las transacciones y proveer un punto de entrada para las operaciones sobre el sistema. Esta capa no debería tener conocimiento sobre lo concerniente a la presentación y debería ser reutilizable.

Capa de acceso a datos: esta capa presenta un conjunto de interfaces independientes de la tecnología de acceso a datos, que son usadas para buscar y persistir los objetos persistentes. La capa de acceso a datos no debería de contener ningún tipo de lógica de negocio.

Caso de uso: Conjunto de secuencia de acciones que un sistema ejecuta y que produce un resultado observable para un actor.

"D"

Despliegue: disciplina del proceso de desarrollo de software que tiene como objetivo implantar la aplicación desarrollada en las instalaciones del cliente.

Diseño: disciplina del proceso de desarrollo de software que tiene como objetivo describir la aplicación partiendo de los desarrolladores, identificando los elementos que la componen en clases y sus relaciones para satisfacer las necesidades funcionales y la calidad de un sistema.

"E"

Eficiencia: capacidad de alcanzar los objetivos y metas programadas con el mínimo de recursos disponibles y tiempo, logrando su optimización.

Equipo de desarrollo: es un grupo de trabajo constituido por una serie de profesores, investigadores, colaboradores y alumnos unidos en la ilusión de acometer un determinado proyecto o avanzar en el conocimiento y en la investigación teórica y aplicada.

"F"

Fiabilidad: conjunto de atributos relacionados con la capacidad del software de mantener su nivel de prestación bajo condiciones establecidas durante un período de tiempo establecido.

Funcionalidad: conjunto de atributos que se relacionan con la existencia de un conjunto de funciones y sus propiedades específicas. Las funciones son aquellas que satisfacen lo indicado o implica necesidades.

"H"

Herramientas: utensilios o provisiones necesarias para poder emprender un proyecto de software. Soportan los procesos de desarrollo de software modernos.

"I"

Implementación: disciplina del proceso de desarrollo de cuya finalidad es implementar los componentes y productos para satisfacer las necesidades funcionales y la calidad de un sistema.

Ingeniería de Software: Se puede definir como el tratamiento sistemático de todas las fases del ciclo de vida del software.

"M"

Mantenibilidad: conjunto de atributos relacionados con la facilidad de extender, modificar o corregir errores en un sistema software.

Metodologías de desarrollo de software: conjunto de procedimientos que imponen una serie de pasos sobre el desarrollo del software que permiten producir y mantener un producto garantizando su fiabilidad y calidad.

"N"

Navegador web: es un programa que permite visualizar la información que contiene una página web (ya esté esta alojada en un servidor dentro de la WWW o en uno local).

"P"

Portabilidad: conjunto de atributos relacionados con la capacidad de un sistema software para ser transferido desde una plataforma a otra.

Proceso: conjunto de actividades y resultados asociados que producen un resultado.

Proceso de desarrollo de software: es la definición del conjunto de actividades que guían los esfuerzos de las personas implicadas en el proyecto para transformar los requisitos de usuario en un producto.

Procesos: conjunto de pasos parcialmente ordenados con el propósito de alcanzar una meta.

Producto: conjunto de artefactos que se crean durante la vida del proyecto, como los modelos, código fuente, ejecutables y documentación.

Proyecto: combinación de recursos humanos y no humanos reunidos en una organización temporal para conseguir un propósito, tiene un punto de comienzo definido y con objetivos definidos mediante los que se identifican.

Prueba: disciplina del proceso de desarrollo de software cuyo propósito es integrar y probar el sistema.

"R"

Recursos: conjunto de elementos disponibles para resolver una necesidad o llevar a cabo una tarea.

Requerimiento: son capacidades o características que debe tener el sistema o modelo desarrollo para satisfacer la demanda y/o necesidad del cliente.

Requisitos: una condición o capacidad que tiene que tener un sistema para satisfacer al cliente.

Requisitos funcionales: son el conjunto de Capacidades o funciones que el sistema debe cumplir.

Reutilización: es la acción de volver a utilizar los bienes o productos ya elaborados y probados. Puede venir propiciada por una mejora o restauración o sin modificarse, usarlo en la creación de un nuevo producto.

"S"

Soporte: disciplina del proceso de desarrollo de software que su objetivo es brindar los servicios de mantenimiento y corrección post venta de un producto.

Técnicas: sucesión ordenada de acciones que se dirigen a un fin concreto, conocido y que conduce a unos resultados precisos.

Tecnología: característica propia del ser humano consistente en la capacidad de éste para construir, a partir de materias primas, una gran variedad de objetos, máquinas y herramientas, así como el desarrollo y perfección en el modo de fabricarlos y emplearlos con vistas a modificar favorablemente el entorno o

conseguir una vida más segura. El ámbito de la Tecnología está comprendido entre la Ciencia y la Técnica propiamente dichas.

"U"

Usabilidad: conjuntos de atributos relacionados con el esfuerzo necesitado para el uso, y en la valoración individual de tal uso, por un establecido o implicado conjunto de usuarios.

"W"

Web: es un sistema de documentos de hipertexto y/o hipermedios enlazados y accesibles a través de Internet. Con un navegador web, un usuario visualiza páginas web que pueden contener texto, imágenes, videos u otros contenidos multimedia, y navega a través de ellas usando hiperenlaces.

Siglas

SGBD: Sistema Gestor de Base de Dato.

CASE: Computer Aided Software Engineering, Ingeniería de Software Asistida por Computadoras.

HTML: Lenguaje de Marcas de Hipertexto o HyperText Markup Language.

IEEE: The Institute of Electrical and Electronics Engineers, el Instituto de Ingenieros Eléctricos y Electrónicos, una asociación técnico-profesional mundial dedicada a la estandarización, entre otras cosas.

MVC: Modelo Vista Controlador.

RUP: Proceso Unificado de Racional o Rational Unified Process.

XP: Programación Extrema o eXtreme Programming.

UCI: Universidad de Ciencias Informáticas.

UML: Lenguaje Unificado de Modelado