Universidad de las Ciencias Informáticas
Vertex, Entornos Interactivos 3D, Facultad 4

Arquitectura para la plataforma de gestión de videojuegos serios Medicando

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

Autor: Gabriel Lázaro García Díaz

Tutores: Ing. Reinaldo Garcia Maturell

Ing. Juan Gabriel Valdés Díaz

Ing. Julio César Espronceda Pérez

La Habana, 2017

"En la tierra hacen falta personas que trabajen más y critiquen menos, que construyan más y destruyan menos, que prometan menos y resuelvan más, que esperen recibir menos y den más, que digan mejor ahora que mañana."

Ché

"Aprendí que el coraje no era la ausencia de miedo, sino el triunfo sobre él. El valiente no es quien no siente miedo, sino quien conquista ese miedo."

Nelson Mandela

"No existe una manera fácil. No importa cuán talentoso seas, tu talento te va a fallar si no lo desarrollas, si no estudias si no trabajas duro; si no te dedicas a ser mejor cada día."

Will Smith

Dedicatoria

Todos estos años de esfuerzo y sacrificio van dedicados a las tres personas que estuvieron siempre presentes en mi formación: Niurka, mi mamá; Carlos, mi papá y mi abuela, Maida. Para ustedes todo lo que logro hoy.

Agradecimientos

Le agradezco en primer lugar a esa persona que me lo ha dado todo en la vida sin pedir nada a cambio. Que ha sido mi soporte, mi guía, mi luz, a esa que siempre sabe que decir. La que siempre está conmigo y me apoya, la que, estando lejos, sabe cómo me siento. Gracias mamá por tu paciencia y comprensión. Nunca olvides lo mucho que te amo. Gracias mami.

A mi papá, por darme un ejemplo de hombre a seguir, por siempre preocuparse y estar al tanto de mí. Por apoyarme en todo momento. Por tus consejos que han hecho de mí una mejor persona. Gracias papá, te quiero mi viejo.

A mi abuela Maida, mi vieja, porque desde la primera vez que me puse un uniforme escolar estuvo a mi lado. Porque siempre te dedicaste a mí y a mi formación. Por tu amor y cariño, por tu preocupación en todo momento. Te estaré eternamente agradecido por todo Abu.

A mi tía Miriam, mis primos y mi familión de Artemisa, porque cuando los necesité, solo me hizo falta levantar el teléfono. Especialmente mi tía, que cuando estuve enfermo o simplemente para que no pasara hambre se las ingeniaba siempre para venir a verme. Gracias gorda.

A mi novia, porque a pesar de estar lejos este último año nunca dejó de darme su apoyo y preocuparse por cómo me iba en las pruebas y en este proceso de tesis, por su amor y paciencia. Gracias Dayi.

A mis tutores: Rei, Juan y Julio, más que tutores amigos. Este logro es gracias a ustedes, les estaré siempre agradecido. Gracias por su paciencia.

A la profe Alina: que, aunque no oficial, fue una tutora más para mí.

Gracias por su ayuda y consejos.

A los tribunales que me ayudaron a mejorar este trabajo durante todo el proceso. Gracias por sus críticas constructivas.

A esos amigos de estudios, al equipo de tantos trabajos: Jaime, Yanelis, Gloria y Lamis.

A los amigos con los que compartí el día a día, en el aula, hablando o discutiendo de fútbol o tanto molesté para que compartieran la WiFi: Oscar, Miguel, Yosvani, El Flaco y su minion, Richard, Yasiel, Almirola, Keiger, Jander y Yasser. Si alguno se me escapa, quiero que sepan que me los llevo a todos en el corazón.

A Randy, Patricia, Damian, Leyanet y Pi que pasamos estos últimos meses juntos haciendo las tesis en el laboratorio.

A Celi, mi primi, por ser mi mensajera.

A May, Esme y Rosme, esas amigas que a pesar de habernos distanciado un poco siempre me aconsejaron y ayudaron con la tesis.

A la familia que hice en los muchachos de la FEU y el Chino, Carlo y David, fue bonito el tiempo que compartí con ustedes.

A la Facultad 5 y sus profesores que me recibieron y formaron. Gracias por todos los conocimientos y valores que me enseñaron, sin ustedes nada de esto hubiese sido posible.

A todos los que contribuyeron de una forma u otra en mi formación.

A esta universidad por cumplir mi sueño

Gracias a TODOS.

Declaración de autoría

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

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

Gabriel Lázaro García Díaz
Autor

Ing. Reinaldo Garcia Maturell
Tutor

Ing. Juan Gabriel Valdés Díaz
Tutor

Ing. Julio César Espronceda Pérez
Tutor

Resumen

La Arquitectura de *Software* (AS) es una disciplina que intenta contrarrestar los efectos negativos que pueden surgir durante el desarrollo de un *software*. La continua evolución de este mundo ha demostrado la importancia de una arquitectura funcional y estable, que permita agilizar la evolución de los sistemas. El presente trabajo tiene como objetivo definir una propuesta de arquitectura para la nueva versión de la plataforma Medicando. En este se presentan los conceptos de arquitectura y su importancia para el proceso de desarrollo de *software*. También se realiza un análisis de varios estilos arquitectónicos y patrones de diseño, con los cuales se elaboró la propuesta de solución. Haciendo uso de los diagramas necesarios de la metodología Proceso Unificado Racional (RUP), por sus siglas en inglés *Rational Unified Process*, se describen los requerimientos del sistema, que lograron dar una representación de la arquitectura a través de las "4+1" vistas arquitectónicas propuestas por Kruchten. A partir de los atributos de calidad que propone la norma Organización Internacional de Normalización (ISO)/Comisión Internacional Electrónica (IEC) 25010, mediante el uso del Método de Análisis de Acuerdos de Arquitectura (ATAM), por sus siglas en inglés *Architecture Tradeoff Analysis Method*, y las técnicas de evaluación basada en escenarios y en prototipo, se efectuó la evaluación a la arquitectura propuesta. La evaluación permitió identificar los riesgos presentes en la propuesta realizada y determinar que la arquitectura satisface los atributos de calidad definidos para la presente investigación.

Palabras clave: arquitectura de *software*, ATAM, atributos de calidad, Medicando.

Índice general

Índice de figuras

Índice de tablas

Introducción

El desarrollo de la tecnología ha irrumpido en todos los ámbitos de la actualidad, trayendo consigo cambios, más específicamente en la mente del ser humano, en su forma de vivir, comunicarse, pensar y hacer [1]. La medicina no escapa a estas consideraciones; los estudios de diagnóstico, así como los recursos terapéuticos se tornan cada vez más sofisticados y numerosos [2]. Según la doctora Victoria Ramos González [3]: "La incorporación de las tecnologías al mundo sanitario está suponiendo un motor de cambio para la mejora de la calidad de vida de los ciudadanos, favoreciendo el desarrollo de herramientas dirigidas a dar respuesta en áreas como la planificación, la información, la investigación, la gestión, prevención, promoción o en el diagnóstico o tratamiento".

Cuba ha estado inmersa en el desarrollo de *software* para informatizar la sociedad, en especial el sector de la medicina. La Universidad de las Ciencias Informáticas (UCI) forma parte en el proceso de informatización de la sociedad cubana, desarrollando *software* en distintas ramas de la ciencia.

La UCI está dividida en diferentes centros de producción, especializados en diversas líneas de investigación. El Centro de Entornos Interactivos 3D (Vertex), cuenta con una línea de desarrollo de videojuegos, entre ellos, videojuegos serios con fines terapéuticos para favorecer algunos procesos de rehabilitación. Con el fin de gestionar estos videojuegos, las estadísticas que generan y los usuarios que los utilizan se desarrolló la plataforma Medicando. Esta herramienta permite proporcionar al personal de salud, un medio para llevar un control de la evolución de los pacientes en sus tratamientos con videojuegos.

El sistema actualmente se encuentra desarrollado en la versión 1.1.14 del *framework* Yii, la cual dejó de tener soporte, por lo que se quiere migrar a una versión superior. Actualmente el sistema cumple de manera funcional con las especificaciones iniciales realizadas por el Centro, sin embargo, se encontraron algunas deficiencias asociadas al diseño arquitectónico, las cuales se quieren mitigar para la nueva versión del sistema, por ejemplo: se comprobó que al adicionar nuevas funcionalidades se necesita realizar modificaciones en la mayor parte del núcleo base del sistema debido a la inexistencia de un diseño modular. Se cuenta con muy poca documentación asociada a la arquitectura, lo que dificulta el aprendizaje de nuevos desarrolladores que se incorporen; pues solo se cuenta con la lógica del negocio y los flujos de trabajo que implementa el sistema. Estas deficiencias que presenta la arquitectura actual hacen más lento el proceso de incorporación de nuevos comportamientos a la aplicación, lo cual dificulta su evolución y afecta el modelo planificado para la comercialización del producto, el cual es basado en la modularidad del sistema.

Teniendo en cuenta la situación anterior, se plantea el siguiente **problema de investigación**: ¿Cómo añadir extensibilidad y reducir los problemas de dependencia de la plataforma Medicando para adicionarle

nuevas funcionalidades? A partir del problema planteado se toma como **objeto de estudio** las arquitecturas de *software* en plataformas *web* y dentro de esta área se toma como **campo de acción** arquitecturas de *software* en plataformas *web* orientadas a la gestión de componentes.

Se define como **objetivo**, desarrollar una arquitectura de *software* que permita añadir extensibilidad y reducir los problemas de dependencia de la plataforma Medicando para adicionar nuevas funcionalidades. Para dar cumplimiento al objetivo planteado se propone el siguiente grupo de **tareas investigativas**:

1 Búsqueda bibliográfica sobre el tema AS para la elaboración del marco teórico a partir del estado del arte actual referente al tema.
2 Análisis de la arquitectura actual de Medicando para identificar los cambios necesarios.
3 Descripción de los cambios y requerimientos del sistema.
4 Diseño de la arquitectura de *software* para la plataforma Medicando 2.0.
5 Desarrollo de la arquitectura propuesta.
6 Validación de la propuesta arquitectónica.

Para la realización de la investigación y elaboración del presente trabajo se utilizan varios métodos científicos de investigación, entre los cuales se pueden mencionar:

Métodos teóricos:

Histórico-Lógico: Método teórico que se utiliza para analizar la evolución y las tendencias actuales de las arquitecturas de *software* y las plataformas virtuales.

Analítico-Sintético: Método teórico el cual permite identificar, analizar y seleccionar los conceptos y definiciones más importantes relacionadas con la presente investigación.

Métodos empíricos:

Consultas de fuente de información: Método empírico que se utiliza para la consulta de fuentes bibliográficas durante la investigación.

Pruebas: se realizan diferentes pruebas a la arquitectura de *software* propuesta para evaluarla y determinar si se lograron los resultados esperados.

CAPÍTULO 1

Fundamentación Teórica

1.1. Introducción al capítulo

En el presente capítulo se realiza una descripción de los conceptos que son utilizados en la investigación. Se expone una valoración del estado del arte de la investigación realizada referente a arquitecturas en plataformas *web* y las características que presentan varias plataformas virtuales para la gestión de videojuegos. Se realiza el análisis de diferentes estilos y patrones arquitectónicos, estableciendo las principales diferencias entre ambos términos, realizando el análisis además de varios patrones de diseño.

1.2. Plataforma Virtual

Una plataforma para el desarrollo de aplicaciones electrónicas, es un entorno informático determinado, que utiliza sistemas compatibles entre sí [4], que ejecutan o realizan un conjunto de servicios que este sistema puede ofrecer [5]. También se conoce como plataforma, a un sistema que sirve como base para hacer funcionar determinados módulos de *hardware* o de *software* con los que es compatible. Al definir plataformas se establecen los tipos de arquitectura, sistema operativo, lenguajes de programación o interfaz de usuario compatibles [6].

A partir de las definiciones anteriores, se puede entender por plataforma virtual; un sistema informático base que permite la integración de diferentes paquetes funcionales, atendiendo a su compatibilidad, entre ellos y con el sistema. Los usuarios no deben estar en un espacio físico determinado, solo necesitan de una conexión a la *web* para poder acceder a este tipo de plataformas y así poder hacer uso de los servicios que estas proveen. Para comprender mejor el funcionamiento de este tipo de sistemas se analizarán varias plataformas enfocadas en la gestión de videojuegos, donde se hará énfasis en las características comunes que estas presentan.

1.3. Análisis de plataformas virtuales para la gestión de videojuegos

Con la intención de analizar las características que presentan plataformas de este tipo y para el mejor entendimiento de estos sistemas se analizan varias plataformas virtuales relacionadas con la gestión de videojuegos.

VirtualRehab: es un novedoso sistema de rehabilitación física clínicamente validado que permite la monitorización y seguimiento de los pacientes desde cualquier lugar del mundo. Los pacientes pueden realizar complejos programas de rehabilitación a través de terapias divertidas tanto en su centro de rehabilitación como en sus propios hogares. Es una plataforma para la rehabilitación de pacientes con algún grado de discapacidad física, que combina modernas técnicas de captura de movimiento con tecnología de videojuegos [7].

NeuroAtHome: es una plataforma *software* de rehabilitación diseñada específicamente para tratar las secuelas de una lesión neurológica o de una enfermedad neurodegenerativa. Independientemente de donde esté el paciente, el equipo clínico podrá diseñar y personalizar las sesiones de rehabilitación que completarán los pacientes y modificar las siguientes sesiones en base a los resultados conseguidos. Además, los resultados recogidos durante las sesiones de rehabilitación se almacenarán y estarán disponibles para el análisis del equipo clínico. Durante cada sesión, NeuroAtHome monitoriza los ejercicios realizados y comprueba el grado y la calidad de ejecución de los mismos. De esta forma, el equipo clínico encargado de la rehabilitación de los pacientes puede comprobar su evolución -de forma objetiva- a lo largo del proceso de rehabilitación [8].

Gaikai: es un servicio de juego basado en la nube que permite a los usuarios jugar demos de juegos de computador y aplicaciones de forma instantánea desde una página *web* o dispositivo conectado a Internet. El servicio de *streaming* de Gaikai puede ser embebido en cualquier sitio *web* relacionado con juegos, sitios como redes sociales o productos específicos (como dispositivos móviles y televisión digital), según lo determinado por el distribuidor [9] .

1.3.1. Características comunes de las plataformas virtuales analizadas

Las plataformas analizadas basan su estilo de negocio en brindar información de los videojuegos y ofrecer facilidades para su comercialización.

Funcionalidades básicas

Entre las herramientas y funcionalidades más básicas y esenciales se encuentran:

- Los procesos de preinscripción e inscripción de los materiales que administran estos sistemas.
- Gestión de usuarios.
- La asignación de roles y control de acceso.
- Seguimientos de los accesos de usuarios.

Teniendo en cuenta el marco de la investigación, se realiza un breve análisis de la plataforma Medicando desarrollada por el Centro Vertex.

1.4. Análisis de la plataforma Medicando

Es una plataforma *web* que gestiona videojuegos terapéuticos y las estadísticas de los mismos. Fue desarrollada con el objetivo de facilitar el seguimiento de pacientes en los tratamientos que les son asignados en el sistema, donde cada tratamiento debe estar relacionado con un videojuego de rehabilitación. El sistema actualmente está desarrollado con el *framework* Yii en su versión 1.1.14, la cual deja de tener soporte al ser liberada la versión 2.0 por su comunidad de desarrollo, esta situación provoca que el Centro Vertex desee migrar el sistema a la nueva tecnología. El uso de este *framework* permitió el empleo de lenguajes como *HTML5*, *CSS3*, *JavaScript* y *PHP v5*. Para el almacenamiento de los datos se utilizó el sistema gestor de bases de datos *MySQL* y como servidor *web Apache Server* [10]. Aunque no es obligatorio hacer uso de estas tecnologías y herramientas para el desarrollo de la nueva versión, con el objetivo de continuar la línea de desarrollo del proyecto, se propone adoptar las mismas herramientas y tecnologías utilizadas por el equipo de desarrollo inicial considerando versiones superiores.

A continuación, se describe brevemente el comportamiento básico de las principales funcionalidades del sistema [11].

- **La gestión de videojuegos**: se encarga de la integración, actualización y eliminación de videojuegos en la plataforma, así como la sincronización de los datos y garantiza el CRUD (*Create*, *Read*, *Update* y *Delete*) de los videojuegos. Controla la posibilidad de descargar o no los videojuegos y permite ejecutarlos directamente en el navegador. Posibilita cargar videojuegos que estén compactados en formato Zip[1]. Cuando se ejecuta el videojuego online, sincroniza los datos cuando el doctor entra a la sección Consultar Estadísticas. Además, brinda la opción de subir un fichero compactado que contenga el JSON[2] (o salva) y ejecutar la sincronización.
- **La salva de estadísticas de los videojuegos**: encargada de la generación de las estadísticas grupales e individuales de la aplicación. A partir de los datos almacenados en el sistema, genera un grupo de estadísticas grupales e individuales asociadas a los videojuegos y a los pacientes asociados a estos.
- **La gestión de tratamientos**: encargada de que los doctores puedan crear y actualizar tratamientos en el sistema. Además, permite asignar uno o varios tratamientos a los pacientes.
- **La gestión de usuarios**: Encargada de la creación, actualización y eliminación de usuarios en el sistema. Además, se encarga del proceso de asignación de roles y permisos en la aplicación. Los usuarios son identificados en correspondencia con el rol que desempeñen. Los roles que se definen son Paciente, Doctor y Administrador.

[1] Es un formato de compresión sin pérdida, muy utilizado para la compresión de datos como documentos, imágenes o programas.
[2] Es un formato ligero para el intercambio de datos.

Con la intención de que el sistema sufra una evolución en su nueva versión, se realiza un análisis a la versión actual de la plataforma. Se detectan debilidades generales del sistema y se identifican deficiencias específicas en ciertas funcionalidades [10] .

- **Debilidades Generales**
  1. El sistema no cuenta con un diseño modular lo cual afecta la mantenibilidad del mismo.
  2. La base de datos del sistema no hace uso de las llaves foráneas y el tratamiento desde la aplicación se hace complejo debido a que, para eliminar un usuario, hay que realizar el proceso de eliminación en cascada de manera manual. No se utiliza el sistema ofrecido por el *framework* para la interrelación de tablas de la base de datos.
  3. Los especialistas no cuentan con los permisos para eliminar cualquier información.
  4. Los datos que consultan los especialistas se encuentran disponibles solo en la plataforma. Esta organización, no permite que los interesados puedan seleccionar la información que necesitan y extraerla para análisis posteriores.
- **Gestionar tratamientos:** no se evidencia una forma sencilla de conocer qué tratamiento está asignado a qué paciente, así como medir el nivel de cumplimiento de estas relaciones. No se encuentra definido un mecanismo para asignar un mismo tratamiento a un grupo de pacientes de manera simultánea.
- **Notificaciones:** Los especialistas no tienen la posibilidad de ser informados en tiempo real cuando un paciente ha iniciado o terminado un tratamiento, por lo que tienen que chequear constantemente si existen actualizaciones en este aspecto. Además, no se implementa un mecanismo que informe a los pacientes, en tiempo real, que se les ha asignado un nuevo tratamiento o de recomendaciones y valoraciones que les pueda dar un doctor, con respecto al tratamiento que realiza.

Teniendo en cuenta el análisis realizado a la plataforma apoyado en trabajos de desarrollo anteriores a la misma, se llega a la conclusión de que el sistema cumple con el propósito por el cual fue creado, por lo cual se mantendrán las principales funcionalidades ya definidas en el sistema, abordando mejoras de las mismas, basadas en las debilidades antes expuestas. En cuanto a las deficiencias arquitectónicas que afectan características de calidad que deberían estar presentes en la plataforma, tales como: modularidad y capacidad para ser modificado y teniendo en cuenta la necesidad de migrar la misma a una nueva versión, es objetivo de esta investigación mitigar estas deficiencias desde la base, desde el diseño y organización de los componentes del sistema, por lo cual se realiza un estudio de arquitecturas usadas en plataformas *web*.

1.5. Análisis de arquitecturas en plataformas *web*

Plataforma BizkaiSense

Es una plataforma para el manejo y la consulta de datos medioambientales recogidos de sensores desplegados en áreas metropolitanas para permitir definir y adoptar medidas estratégicas de diverso calado en relación con la sostenibilidad. A continuación, se detalla la arquitectura de la plataforma BizkaiSense [12]:

Se trata de una arquitectura multicapa. Esta arquitectura ha sido diseñada de forma modular, asegurando así, la alta escalabilidad, el bajo acoplamiento y la alta cohesión que una plataforma de estas características debe tener.

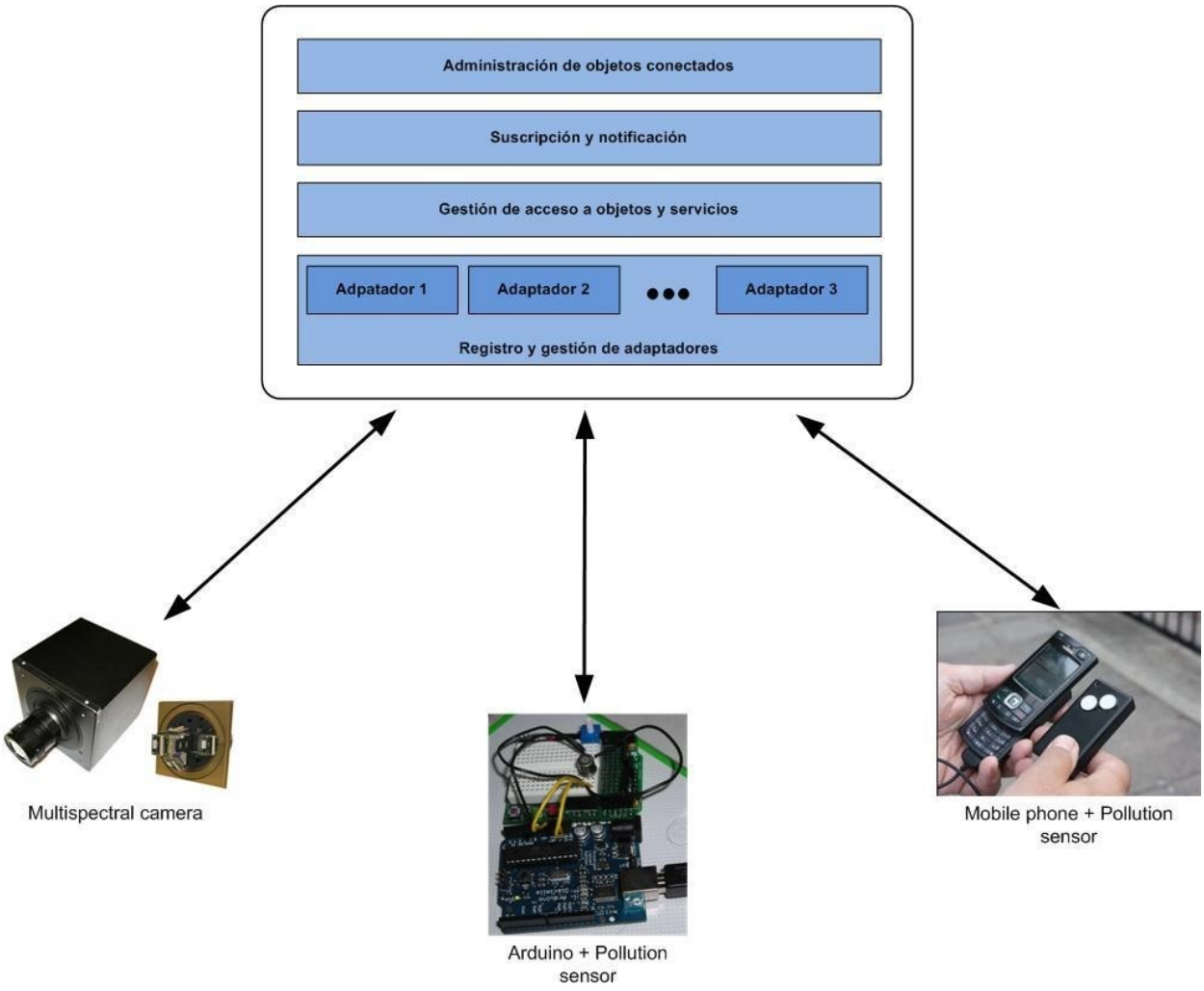

Figura 1.1. Arquitectura de BizkaiSense.

Propuesta de arquitectura para plataformas *web* de gestión de procesos desarrolladas en Django

En este trabajo desarrollado por ingenieros de la UCI, de la Facultad Regional de Granma en 2012, se propone una arquitectura para plataformas *web* de gestión de procesos. Los autores proponen una arquitectura basada en componentes reutilizables, donde estos componentes son la base fundamental de la arquitectura, logrando de esta forma una arquitectura modificable, segura y funcional, que asegure el fácil mantenimiento, amplia extensibilidad y alta reusabilidad de los componentes de la plataforma, capaz de integrar nuevas funcionalidades sin que se vea afectado el rendimiento de la plataforma. Para evaluar la arquitectura se usó el método ATAM, con el objetivo de identificar los riesgos y fortalezas de la propuesta, para mejorar la calidad

del *software* y su aplicación [13].

De las arquitecturas estudiadas, cabe destacar la escalabilidad, el bajo acoplamiento y la alta cohesión que brinda un diseño modular y la extensibilidad, la facilidad de mantenimiento y de integración o evolución de nuevas funcionalidades sin la afectación del rendimiento del sistema que se obtiene de una arquitectura basada en componentes. Además de la importancia que tiene la correcta validación de la arquitectura a partir de un método de evaluación.

Para la mejor comprensión de los elementos que presentan las arquitecturas estudiadas se hace un estudio de los conceptos y el diseño de AS.

1.6. Arquitectura de *software*

Según [14], la AS es la estructura del sistema en función de la definición de los componentes y sus interacciones. Se puede considerar como un puente entre los requisitos del sistema y la implementación. Es el plan de diseño del sistema, debido a que es usada como guía para el resto de las tareas de la etapa de desarrollo.

Del tema [15] afirma que, aunque existen abundantes definiciones del campo de la AS, existe en general acuerdo de que ella se refiere a la estructura a grandes rasgos del sistema, estructura consistente en componentes y relaciones entre ellos. Estas cuestiones estructurales se vinculan con el diseño, pues la AS es después de todo una forma de diseño de *software* que se manifiesta tempranamente en el proceso de creación de un sistema. A grandes rasgos es una vista del sistema que incluye los componentes principales del mismo, la conducta de esos componentes según se la percibe desde el resto del sistema y las formas en que los componentes interactúan y se coordinan para alcanzar la misión del sistema.

Paul Clements [16] define cinco temas fundamentales en torno de los cuales se agrupa la disciplina:

- **Diseño o selección de la arquitectura:** Crear o seleccionar una arquitectura en base de requerimientos funcionales, de rendimiento o de calidad.
- **Representación de la arquitectura:** Comunicar una arquitectura. Este problema se ha manifestado como el problema de la representación de arquitecturas utilizando recursos lingüísticos, pero el problema también incluye la selección del conjunto de información a ser comunicada.
- **Evaluación y análisis de la arquitectura:** Analizar una arquitectura para predecir cualidades del sistema en que se manifiesta. Un problema semejante es cómo comparar y escoger entre diversas arquitecturas en competencia.
- **Desarrollo y evolución basados en arquitectura:** Construir y mantener un sistema dada una representación de la cual se cree que es la arquitectura que resolverá el problema correspondiente.
- **Recuperación de la arquitectura:** Hacer que un sistema legado evolucione cuando los cambios afectan su estructura; para los sistemas de los que se carezca de documentación confiable, esto involucra

primero una arqueología arquitectónica que extraiga su arquitectura.

Luego de la investigación realizada, se puede entender entonces que la AS es la organización y diseño de los componentes del sistema, para que este al implementarse cumpla con todos los reuerimientos funcionales y no funcionales; así como con los estándares de calidad, rendimiento, facilidad de mantenimiento, flexibilidad para la incorporación de funcionalidades y reutilización de componentes.

1.6.1. Diferentes vistas de una arquitectura

La AS se enfrenta a la abstracción, descomposición y composición usando estilos y patrones. Para describirla se usa un modelo compuesto de múltiples vistas o perspectivas. Con el fin de resumir estos modelos, Philippe Kruchten [17] propone:

- **Vista lógica:** es el objeto del diseño.
- **Vista de procesos:** captura la concurrencia y sincronización como aspectos del diseño.
- **Vista física:** describe el mapeo del *software* en el *hardware* y refleja su aspecto distribuido.
- **Vista de desarrollo:** describe la organización estática del *software* en su entorno de desarrollo.
- **Escenarios:** es donde se ensamblan los elementos mostrados en las vistas anteriores.

La descripción de una arquitectura puede ser organizada alrededor de las vistas lógica, de procesos, física y de desarrollo, mostrándose en los escenarios. La arquitectura, de hecho, evoluciona a partir de estos escenarios.

A la hora de diseñar una AS se crean y representan componentes que interactúan entre sí, con responsabilidades específicas, y se organizan de forma tal que se logren los requerimientos establecidos. Se puede partir con patrones de soluciones probados que se conocen con el nombre de estilos arquitectónicos, patrones arquitectónicos y patrones de diseño.

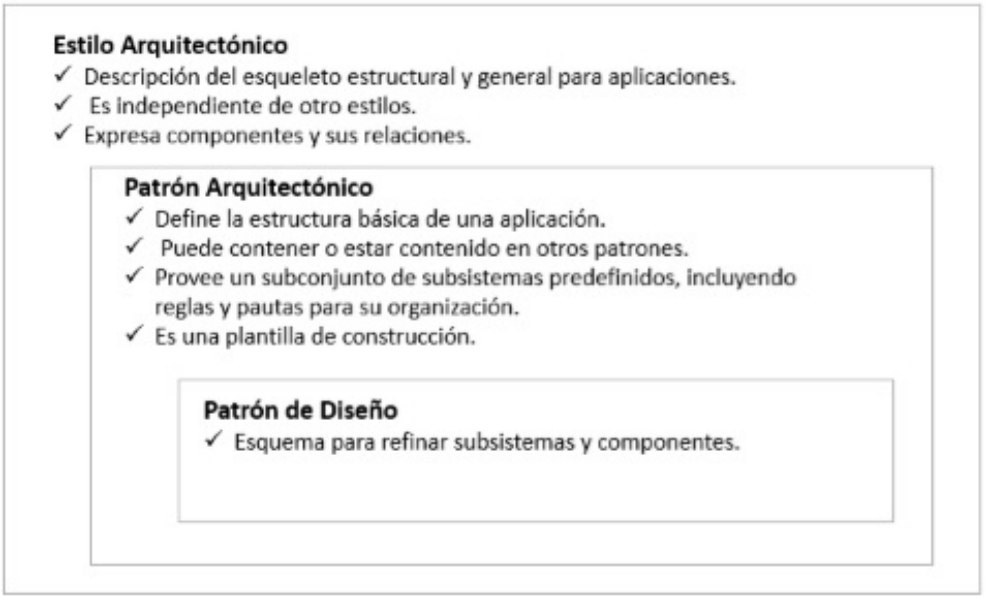

Figura 1.2. Relación entre estilo arquitectónico, patrón arquitectónico y patrón de diseño.

1.7. Estilos y Patrones Arquitectónicos

Los estilos arquitectónicos son un conjunto de reglas de diseño que identifica las clases de componentes y conectores que se pueden utilizar para componer en sistema o subsistema, junto con las restricciones locales o globales de la forma en que la composición se lleva a cabo. Se definen en algunas bibliografías como una abstracción de tipos de elementos y aspectos formales a partir de diversas arquitecturas específicas. Un estilo arquitectónico encapsula decisiones esenciales sobre los elementos arquitectónicos y enfatiza restricciones importantes de los elementos y sus relaciones posibles [16].

Reynoso y Kicillof categorizan lo estilos arquitectónicos de la siguiente manera [18]:

Tabla 1.1. Categorización de los estilos arquitectónicos.

Flujos de datos	**Centrados en datos**	**Llamada y retorno**	**Código móvil**	**Heterogéneos**	**Peer-to-Peer**
Tubería y filtros	Arquitectura de pizarra o repositorio	Modelo-Vista-Controlador (MVC)	Arquitectura de máquinas virtuales	Sistema de control de procesos	Arquitectura basada en eventos
		Arquitectura en capas		Arquitectura basada en atributos	Arquitectura orientada a servicios
		Arquitectura orientada a objetos			Arquitecturas basadas en recursos
		Arquitecturas basadas en componentes			

A continuación, se hace énfasis en los estilos de Llamada y Retorno, dado a que esta familia de estilos enfatiza el uso de la modificabilidad y la escalabilidad.

1.7.1. Estilo basado en componentes

El estilo de arquitectura basada en componentes, describe un acercamiento al diseño de sistemas como un conjunto de componentes que exponen interfaces bien definidas y que colaboran entre sí para resolver el problema [19].

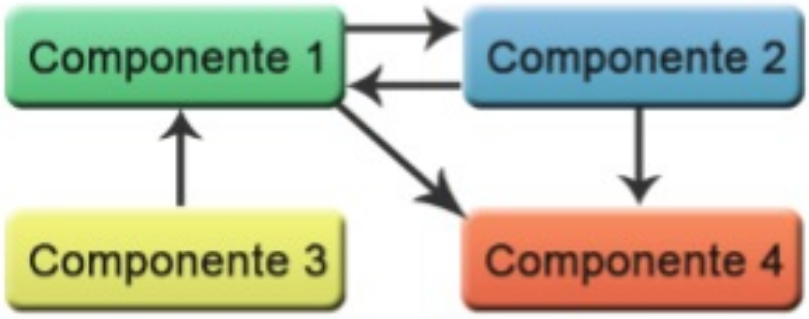

Figura 1.3. Arquitectura basada en componentes.

Este estilo se utiliza para diseñar aplicaciones a partir de componentes individuales, enfatiza la descomposición del sistema en componentes lógicos o funcionales y define una aproximación al diseño a través de

componentes que se comunican mediante interfaces que contienen métodos, eventos y propiedades. El uso de este estilo facilita el despliegue, pues permite sustituir un componente por su nueva versión sin afectar a otros componentes o al sistema y favorece la reusabilidad de los componentes independientes del contexto, permitiendo que se empleen en otras aplicaciones y sistemas [20].

Las ventajas y desventajas de este estilo son [21]:

Ventajas

- **Facilidad de Instalación:** Cuando una nueva versión esté disponible, se podrá remplazar la versión existente sin impacto en otros componentes o el sistema como un todo.
- **Costos reducidos:** El uso de componentes de terceros permite distribuir el costo del desarrollo y del mantenimiento.
- **Facilidad de desarrollo:** Los componentes implementan interfaces bien definidas para proveer la funcionalidad permitiendo el desarrollo sin impactar otras partes del sistema.
- **Reusable:** El uso de componentes reutilizables significa que ellos pueden ser usados para distribuir el desarrollo y el mantenimiento entre múltiples aplicaciones y sistemas.
- **Facilidad de prueba:** El uso de compontes en la aplicación permite probarlos a cada uno como una unidad independiente facilitando más tarde las tareas de pruebas.

Desventajas

- **Evolución de los componentes:** La gestión de la evolución es un problema, pues en los sistemas grandes han de poder coexistir varias versiones de un mismo componente.

1.7.2. Estilo basado en Capas

El estilo basado en capas funciona como una organización jerárquica tal que cada capa proporciona servicios a la capa inmediatamente superior y se sirve de las prestaciones que le brinda la inmediatamente inferior [22].

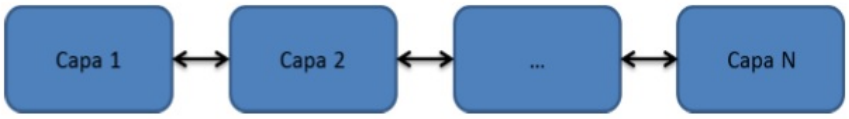

Figura 1.4. Estilo basado en capa.

Una de las principales características de este estilo se encuentra en la descomposición de los servicios de forma que la mayoría de interacciones ocurre solo entre capas vecinas. Los componentes de cada capa se comunican con los componentes de otras capas a través de interfaces conocidas y cada nivel agrega las responsabilidades y abstracciones del nivel inferior [20] .

El uso de este estilo trae ventajas, por ejemplo:

- **Abstracción:** los cambios se realizan a alto nivel y se puede incrementar o reducir el nivel de abstracción que se usa en cada capa del modelo.
- **Aislamiento:** se pueden realizar actualizaciones en el interior de las capas sin que esto afecte el resto del sistema.
- **Rendimiento:** distribuyendo las capas en distintos niveles físicos se puede mejorar la escalabilidad, la tolerancia a fallos y el rendimiento.
- **Facilidad de pruebas:** cada capa tiene una interfaz definida sobre la que realizar las pruebas.

Algunas desventajas que representa la utilización de este estilo son: [18]:

- Muchos problemas no se adaptan naturalmente a una estructura de capas.
- Los cambios en las capas de bajo nivel tienden a filtrarse hacia las de alto nivel.
- La arquitectura en capas ayuda a controlar y encapsular aplicaciones complejas, pero puede complicar las aplicaciones simples.

1.7.3. Estilo Orientado a Objetos

En este estilo los componentes se basan en los principios orientados a objetos. En una forma simple, un diseño orientado a objetos permite diseñar sistemas para encapsular los datos y los objetos proveen explícitamente interfaces a otros objetos, las interfaces están separadas de las implementaciones. Pero la principal característica de este estilo es que se puede modificar la implementación de un objeto sin afectar la interfaz. Hay muchas variantes en este estilo ya que algunos sistemas permiten que los objetos sean tareas concurrentes y otros admiten que los objetos contengan diferentes interfaces [18].

Las ventajas y desventajas del uso de este estilo son:

Ventajas

- La integración de un conjunto de rutinas de acceso con los datos que manipulan permite a los diseñadores descomponer los problemas en colecciones de agentes que interactúan.

Desventajas

- Para que un objeto interactúe con otro (mediante la invocación a un procedimiento) debe conocer la identidad del otro objeto. Luego, cuando la identidad de un objeto cambie es necesario modificar todas las invocaciones a tal objeto.

Para la mejor comprensión de las AS, es necesario profundizar sobre los patrones arquitectónicos, considerando que estos se utilizan para expresar una estructura de organización para un software.

1.7.4. Patrones arquitectónicos

Los patrones arquitectónicos resuelven un problema de aplicación específica dentro de un contexto dado y sujeto a limitaciones y restricciones. Estos patrones proponen una solución arquitectónica que sirve como base para el diseño de la arquitectura. La mayor parte de aplicaciones pertenecen a un dominio o género específico, y para éstas son apropiados uno o más estilos de arquitectura, dentro de ese estilo se encontrará un conjunto de problemas comunes que se abordan mejor con patrones arquitectónicos específicos [23] .

Un patrón arquitectónico expresa una organización estructural para sistemas de *software*. Se definen sobre aspectos fundamentales de la estructura del sistema, donde se especifican una serie de recomendaciones para organizar los distintos componentes.

Uno de los patrones que ha demostrado ser fundamental a la hora de diseñar aplicaciones *web* es el MVC [24]:

- **Patrón arquitectónico MVC:** Surge con el objetivo de reducir el esfuerzo de programación, necesario en la implementación de sistemas, a partir de estandarizar el diseño de las aplicaciones. Separa el modelado del dominio, la presentación y las acciones basadas en datos ingresados por el usuario en tres clases diferentes, garantizando así la actualización y mantenimiento del *software* de forma sencilla y en poco tiempo: El modelo, en el que se administra el comportamiento y los datos del dominio de aplicación, responde a requerimientos de información sobre su estado y da respuestas a instrucciones de cambiar el estado. La vista, la cual maneja la visualización de la información. Y el controlador, que interpreta las acciones del usuario, informando al modelo y/o a las vistas para que cambien según resulte apropiado.

1.7.5. Diferencia entre Estilos y Patrones Arquitectónicos

Los conceptos de Estilos y Patrones arquitectónicos son similares y no siempre es sencillo identificar un límite nítido entre ellos [25]:

Tabla 1.2. Diferencia entre estilos y patrones arquitectónicos.

	Estilos Arquitectónicos	**Patrones Arquitectónicos**
Alcance	Aplican a un contexto de desarrollo.	Aplican a problemas de diseño específicos.
Abstracción	Son muy abstractos para producir un diseño concreto del sistema.	Son fragmentos arquitectónicos parametrizados que pueden ser pensados como una pieza concreta de diseño.
Relación	Un sistema diseñado de acuerdo a las reglas de un único estilo puede involucrar el uso de múltiples patrones.	Un único patrón puede ser aplicado a sistemas diseñados de acuerdo a los lineamientos de múltiples estilos.

Reynoso y Kicillof [18] expresan: "En cuanto a los patrones de arquitectura, su relación con los estilos arquitectónicos es perceptible, pero indirecta y variable incluso dentro de la obra de un mismo autor", sobre las diferencias entre ambos conceptos señalan:

- Existen claras convergencias entre ambos conceptos, aun cuando se reconoce que los patrones se refieren más bien a prácticas de reutilización y los estilos conciernen a teorías sobre la estructura de los sistemas a veces más formales que concretas.
- Quienes trabajan con estilos favorecen un tratamiento estructural que concierne más bien a la teoría, la investigación académica y la arquitectura en el nivel de abstracción más elevado, mientras que quienes trabajan con patrones se ocupan de cuestiones que están más cerca del diseño, la práctica, la implementación, el proceso, el refinamiento y el código.

Partiendo de lo dicho anteriormente se puede concluir entonces que los estilos son guías para crear un diseño más general del sistema, mientras que los patrones se usan para darle solución a problemas más pequeños y más específicos dentro de un estilo dado. Luego de analizados los estilos y patrones arquitectónicos se decide usar para la propuesta de solución el estilo arquitectónico basado en componentes, por la modularidad que brinda las características de este estilo; y como patrón el MVC, patrón muy usado en los marcos de trabajo *web*, Yii entre ellos, por la reusabilidad de código y por la organización que este presenta al separar responsabilidades.

Para obtener una correcta implementación se hace necesario tener en cuenta las mejores soluciones conocidas, con el objetivo de diseñar las clases y las funcionalidades a codificar. Para ello se analizan patrones de diseño que responden ante las necesidades de desarrollo.

1.8. Patrones de diseño

Son patrones de un nivel de abstracción menor que los patrones de arquitectura. Están por lo tanto más próximos a lo que sería el código fuente final [26] .

Los patrones de diseño trabajan a una escala intermedia y son independientes del lenguaje de programación que se utilice. Su aplicación no tiene efectos en la estructura fundamental del sistema (arquitectura), pero puede tener una fuerte influencia sobre la arquitectura de un subsistema. Definen un esquema de refinamiento de los subsistemas o componentes dentro de un sistema, o las relaciones entre estos [27].

En resumen, los patrones de diseño representan esquemas para solucionar problemas comunes y conocidos de programación. Yii hace uso de algunos patrones de diseño, estos son:

Experto: Asigna la responsabilidad a la clase que tiene la información necesaria para cumplir la responsabilidad.

En Yii toda la información de los datos la manejan las clases modelos, pues son estas las que acceden a la base de dato en respuesta de una petición de las clases controladoras para satisfacer un evento iniciado por el usuario.

Controlador: Asignar la responsabilidad de manejar los eventos del sistema a una clase. Un evento de entrada al sistema es algún evento desde algún actor externo. La clase controlador es responsable de decidir qué hacer con el evento. El controlador actúa como una fachada entre las interfaces y la aplicación.

Yii tiene bien definidas las clases controladoras dentro de su patrón MVC, las cuales se encargan de delegar responsabilidades, en respuesta a alguna acción del usuario, a las clases modelo e interfaces.

Active Record: Define una forma de acceder a los datos de una base de datos y convertir las filas de una tabla en objetos.

Yii, implementa el patrón de persistencia ActiveRecord, que facilita el control de la información almacenada en la base de datos. Se trata de una clase que se encarga de implementar todas las operaciones de consulta y modificación de una tabla concreta de la base de datos. De esta forma, la aplicación delega el trabajo con SQL, a la capa de componentes ActiveRecord que maneja Yii.

Luego de modelada y diseñada, es necesario conocer si la arquitectura cumple el propósito por la cual fue concebida, de ahí que uno de los pasos principales en la creación de una arquitectura es su evaluación y validación.

1.9. Evaluación de arquitectura de *software*

La arquitectura es el resultado de las decisiones tempranas de diseño, que son necesarias antes de empezar a construir el sistema. La AS posee gran impacto sobre la calidad de un sistema, por lo que se hace necesario evaluarla para determinar su potencial para alcanzar los atributos de calidad requeridos. Es impor-

tante destacar que la evaluación no define si una arquitectura es buena o no, simplemente expresa donde se encuentran los riesgos y fortalezas de la misma. La evaluación arquitectónica demuestra si las decisiones, estilos y patrones impactan positivamente los atributos de calidad del sistema. Evaluar una arquitectura tiene como objetivo verificar que el software cumpla con los atributos de calidad y restricciones planteadas.[28].

El primer paso para la evaluación es conocer qué es lo que se quiere evaluar, de esta forma es posible establecer la base para la evaluación, otra decisión importante es determinar cuándo se realizará la evaluación. Para esto, aunque es posible evaluar la arquitectura en cualquier fase del desarrollo, existen dos variantes definidas de cuando realizar la evaluación: evaluación temprana y evaluación tardía [29].

Para realizar la evaluación temprana no es necesario esperar que la arquitectura esté totalmente especificada, esta puede realizarse desde las fases tempranas del diseño y a lo largo del proceso de desarrollo, lo que permite tomar decisiones arquitectónicas producto a una evaluación en función de los atributos de calidad esperados. La evaluación tardía se realiza cuando la arquitectura ya está establecida y la implementación se ha culminado, realizar la evaluación en este momento se considera muy útil, pues se puede observar el cumplimiento de los atributos de calidad asociados al sistema y su comportamiento de forma general [30].

En este trabajo se tendrá en cuenta la evaluación temprana para comprobar que las decisiones arquitectónicas tomadas durante el proceso de diseño influyan positivamente en los atributos de calidad.

1.10. Atributos de calidad

La ISO, por sus siglas en inglés, *International Organization for Standardization* y la IEC, por sus siglas en inglés, *International Electrotechnical Commission*, definen la ISO/IEC 25000, la cual es una familia de normas que tiene por objetivo la creación de un marco de trabajo común para evaluar la calidad del producto *software*. Es el resultado de la evolución de otras normas anteriores, especialmente de las normas ISO/IEC 9126, e ISO/IEC 14598, que abordaba el proceso de evaluación de productos *software*, convirtiéndose así en el referente para la evaluación de la calidad del producto *software*. Los atributos de calidad que se quieren alcanzar en el diseño arquitectónico son los propuestos por uno de los modelos de esta familia de normas, el modelo ISO/IEC 25010:2011. En total son 8 las características de calidad identificadas en este modelo [31]:

- **Funcionalidad:**Representa la capacidad del producto *software* para proporcionar funciones que satisfacen las necesidades declaradas.
- **Seguridad:** Capacidad de protección de la información y los datos de manera que personas o sistemas no autorizados no puedan leerlos o modificarlos.
- **Interoperabilidad:** Habilidad de que un grupo de partes del sistema trabajen con otro sistema.
- **Confiabilidad:** Capacidad de un sistema o componente para desempeñar las funciones especificadas, cuando se usa bajo unas condiciones y periodo de tiempo determinados.
- **Usabilidad:** Capacidad del producto *software* para ser entendido, aprendido, usado y resultar atractivo para el usuario.

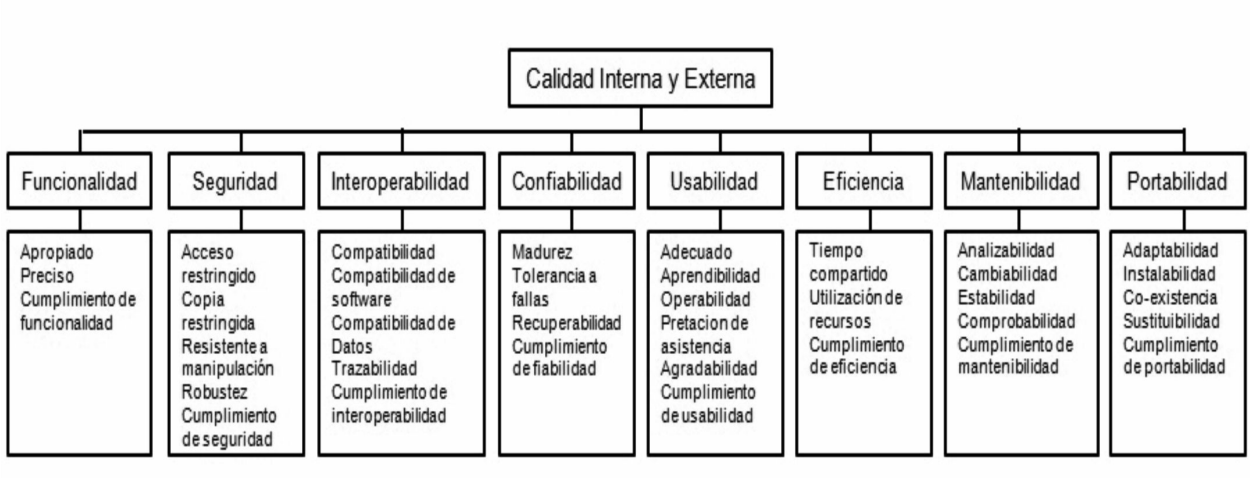

Figura 1.5. Modelo de calidad del producto *software* ISO/IEC 25010:2011.

- **Eficiencia:** Esta característica representa el desempeño relativo a la cantidad de recursos utilizados bajo determinadas condiciones.
- **Mantenibilidad:** Esta característica representa la capacidad del producto *software* para ser modificado efectiva y eficientemente, debido a necesidades evolutivas, correctivas o perfectivas.
- **Portabilidad:** Capacidad del producto o componente de ser transferido de forma efectiva y eficiente de un entorno *hardware*, *software*, operacional o de utilización a otro.

La calidad del producto *software* se puede interpretar como el grado en que dicho producto satisface los requisitos de sus usuarios. Son precisamente estos requisitos los que categorizan la calidad del producto en características y subcaracterísticas y la base de los métodos y técnicas de evaluación de arquitecturas.

1.11. Métodos de evaluación de la Arquitectura de Software

En la actualidad existen varios métodos que permiten realizar la evaluación de la AS. Existen métodos de evaluación que solo analizan específicamente un atributo de calidad. Debido a que en este trabajo se desea realizar una evaluación completa de la arquitectura propuesta, a continuación, se analizan los métodos que evalúan la AS teniendo en cuenta todos los atributos de calidad definidos por el arquitecto:

Método de Análisis para Arquitecturas de Software (SAAM): Es un método para la evaluación temprana de arquitecturas, con respecto a atributos de calidad expresados en el contexto de circunstancias específicas (escenarios). Proporciona un marco conceptual suficiente para saber qué es necesario, pero no suficiente para realizar una evaluación completa de la arquitectura. La utilización de escenarios ha demostrado constituir una herramienta importante para la comunicación entre el equipo de desarrollo, el equipo de diseñadores y los gerentes y superiores. En la práctica SAAM ha demostrado ser útil para evaluar muchos atributos de

calidad rápidamente, como portabilidad, modificabilidad, extensibilidad, integrabilidad, así como el cubrimiento funcional que tiene la arquitectura sobre los requerimientos del sistema. El método también puede ser utilizado para evaluar aspectos más ligados con la arquitectura. Sin embargo, existe el métodos ATAM que explora estos aspectos con más profundidad. [28].

ATAM: Está inspirado en tres áreas distintas: los estilos arquitectónicos, el análisis de atributos de calidad y el método de evaluación SAAM. Evalúa las consecuencias de las decisiones arquitectónicas a la luz de los requisitos de atributos de calidad. Este método dice cuan bien una arquitectura particular satisface las metas de calidad y provee ideas de cómo esas metas de calidad interactúan entre ellas, como realizan concesiones mutuas entre ellas. El método se concentra en la identificación de los estilos arquitectónicos o enfoques arquitectónicos utilizados. ATAM permite comprender mejor los atributos de calidad del sistema [32].

El método de evaluación ATAM comprende nueve pasos que se enumeran a continuación:

1. Presentación de ATAM.
2. Presentación de las metas del negocio.
3. Presentación de la arquitectura.
4. Identificación de los estilos arquitectónicos.
5. Generación del árbol de utilidad.
6. Análisis de los estilos arquitectónicos.
7. Establecimiento de la prioridad a los escenarios.
8. Análisis de los estilos arquitectónicos usando como base los escenarios.
9. Presentación de resultados.

Revisiones Activas para Diseño Intermedio (ARID): por sus siglas en inglés *Active Reviews for Intermediate Design*. Este método es conveniente para realizar la evaluación de diseños parciales en las etapas tempranas del desarrollo. Es un híbrido entre Revisión Activa del Diseño (ADR), por sus siglas en inglés, *Active Design Review*, y ATAM. En ADR, los involucrados en la evaluación reciben documentación detallada y llenan sus cuestionarios de forma independiente. En ATAM se evalúa toda la arquitectura. El método ARID recoge de ADR, la fidelidad de las respuestas que se obtiene de los involucrados en el desarrollo y del ATAM la idea de que sean los involucrados con el sistema los encargados de generar los escenarios. Permite reunir las partes interesadas y los diseñadores en las primeras fases de desarrollo del *software* [33].

Luego de analizados estos métodos de evaluación se puede llegar a la conclusión de que el método de evaluación SAAM se enfoca en un conjunto de escenarios en los que se representan los cambios probables a los que estará sometido el sistema en el futuro. Con este método se obtienen los lugares en que la AS puede fallar, en términos de modificabilidad. Por otro lado, el ARID evalúa mejor la factibilidad de la arquitectura,

haciendo revisiones a la misma en etapas intermedias. Mientras que el método ATAM evalúa con más profundidad, en relación a los otros métodos, aspectos como los atributos de calidad y centra su evaluación en las consecuencias de las decisiones arquitectónicas sobre estos atributos. También permite mediante los escenarios describir la forma en la que el sistema puede crecer y responder a cambios. Después de comparados estos métodos se decide usar el ATAM, pues se pretende hacer una evaluación profunda de la arquitectura teniendo en cuenta los atributos de calidad propuestos por la norma ISO/IEC 25010.

1.12. Técnicas de evaluación de arquitecturas

Las técnicas de evaluación de arquitecturas tienen como principal objetivo, crear especificaciones y predicciones respecto a una propuesta arquitectónica, para saber si satisface las cualidades que el *software* debe cumplir. Pueden clasificarse en dos vertientes fundamentales las cuantitativas y las cualitativas. Generalmente las técnicas de evaluación cualitativas son utilizadas cuando la arquitectura está en construcción, mientras que las técnicas de evaluación cuantitativas, se usan cuando la arquitectura ya ha sido implantada. Estos tipos de técnicas permiten establecer comparaciones entre arquitecturas candidatas para determinar cuál satisface mejor un atributo de calidad específico [34].

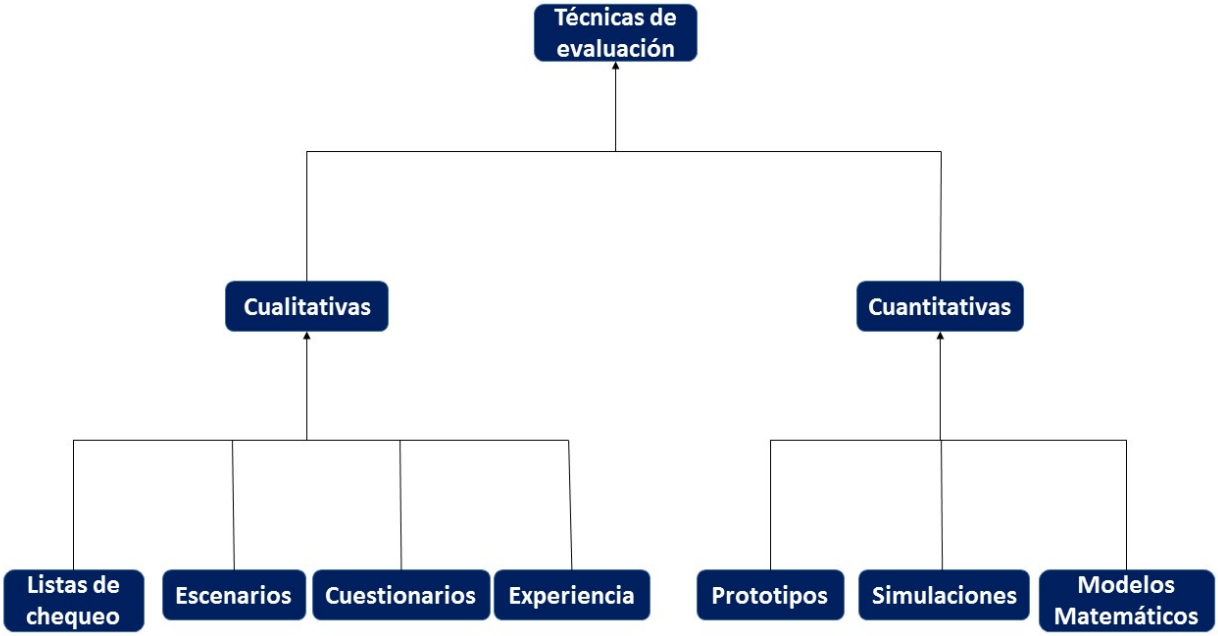

Figura 1.6. Técnicas de evaluación.

Dado a que se usará ATAM como método de evaluación y en uno de sus pasos se especifican escenarios, es necesario analizar la evaluación basada en escenarios. Se analizará también la evaluación por prototipo, pues se desarrollará un prototipo funcional usando la arquitectura propuesta.

1.12.1. Evaluación basada en escenarios

La evaluación basada en escenarios, toma como entrada escenarios. Un escenario es una breve descripción de la interacción de alguno de los involucrados en el desarrollo del sistema con éste. Un escenario consta de tres partes: el estímulo, el contexto y la respuesta. El estímulo es la parte del escenario que explica o describe lo que el involucrado en el desarrollo hace para iniciar la interacción con el sistema. El contexto es el que describe qué sucede en el sistema al momento del estímulo. La respuesta describe, a través de la arquitectura, cómo debería responder el sistema ante el estímulo. Este último elemento es el que permite establecer cuál es el atributo de calidad asociado. Dentro de las herramientas utilizadas por este método se encuentra el árbol de utilidad, el cual es un esquema en forma de árbol que presenta los atributos de calidad de un sistema de *software*, refinados hasta el establecimiento de escenarios que especifican el nivel de prioridad de un atributo de calidad. Su intención es identificar los atributos de calidad más importantes para el proyecto [35].

1.12.2. Evaluación basada en prototipo

Consiste en implementar una parte de la AS y ejecutarla en el contexto del sistema. Mediante esta técnica se obtiene un resultado de evaluación con mayor exactitud. Entre los aspectos favorables de su uso se destaca la confiabilidad, pues se puede ver de manera directa que tanto se afecta o no, un atributo de calidad en el sistema que se desarrolla, los usuarios finales pueden observar el resultado que se está obteniendo y evaluar junto al equipo de desarrollo si satisface o no sus expectativas [34].

1.13. Conclusiones del capítulo

A partir del estudio de los estilos y patrones arquitectónicos y las arquitecturas más reconocidas en la ingeniería de *software* se pudo obtener un grupo de conocimientos necesarios para definir una AS para plataformas *web*. Además se adoptan las herramientas y tecnologías definidas en la línea de desarrollo del proyecto inicial, para desarrollar las funcionalidades del prototipo a implementar.

Para la conformación de la base tecnológica de la arquitectura se selecciona como patrón arquitectónico: MVC. Además, se utilizará el estilo arquitectónico basado en componentes. Por último, queda seleccionado el método de evaluación de la arquitectura a utilizar siendo este ATAM, y los atributos de calidad ISO/IEC 25010 que serán empleados para una correcta evaluación de la arquitectura.

CAPÍTULO 2

Propuesta de solución

2.1. Introducción al capítulo

El diseño de la arquitectura de un sistema define qué componentes forman el sistema, cómo se relacionan entre ellos, y cómo mediante su interacción llevan a cabo la funcionalidad especificada, cumpliendo con los criterios de calidad indicados.

En este capítulo se define la propuesta de solución, la cual se representará a partir del modelo de las "4+1" Vistas de la AS de Kruchten y se especifican los requisitos funcionales y no funcionales.

2.2. Requisitos funcionales

Los requisitos funcionales son las circunstancias y condiciones que el sistema debe cumplir. Describen los servicios que debe brindar el *software*, ayudando en la comunicación entre clientes y desarrolladores. En resumen, los requisitos funcionales describen lo que la aplicación debe hacer. A continuación, se presentan los requisitos principales de la plataforma Medicando.

Tabla 2.1. Requisitos funcionales.

No.	Nombre	No.	Nombre
RF. 1	Adicionar usuario	RF. 12	Eliminar módulo
RF. 2	Autenticar usuario	RF. 13	Activar módulo
RF. 3	Actualizar usuario	RF. 14	Desactivar módulo
RF. 4	Eliminar usuario	RF. 15	Salvar resultados de juego
RF. 5	Añadir videojuego	RF. 16	Generar estadísticas
RF. 6	Actualizar videojuego	RF. 17	Consultar estadísticas
RF. 7	Eliminar videojuego	RF. 18	Adicionar tratamiento
RF. 8	Incorporar módulo	RF. 19	Actualizar tratamiento
RF. 9	Actualizar módulo	RF. 20	Eliminar tratamiento
RF. 10	Instalar módulo	RF. 21	Asignar tratamiento
RF. 11	Desinstalar módulo		

2.3. Requisitos no funcionales

Los requisitos no funcionales definen propiedades o cualidades que el producto debe poseer. Estas propiedades hacen que un producto sea atractivo, usable, rápido o confiable. Son precisamente estos requisitos los que se encuentran representados en el modelo de calidad ISO/IEC 25010 especificados en el epígrafe 1.10, por el cual se guiará la evaluación de la arquitectura propuesta.

2.4. Restricciones arquitectónicas

La definición de una AS debe poseer un conjunto de reglas para el desarrollo de un *software*, en este contexto, las reglas se denominan restricciones arquitectónicas. A continuación, se relacionan las restricciones arquitectónicas que debe cumplir la propuesta de solución:

- La arquitectura debe garantizar que los productos que se desarrollen sean multiplataforma (Linux y Windows).
- La arquitectura debe permitir la actualización o incorporación de componentes sin la necesidad de modificar el código base del sistema.

2.5. Descripción de la arquitectura

Una arquitectura basada en componentes describe una aproximación de ingeniería de *software* al diseño y desarrollo de un sistema. Esta arquitectura se enfoca en la descomposición del diseño en elementos funcionales o lógicos que expongan interfaces de comunicación bien definidas. Un componente es un objeto

de *software* específicamente diseñado para cumplir con cierto propósito y para tener una dependencia mínima de otros elementos. Por lo tanto, pueden ser instalados en el ambiente adecuado sin afectar a otros o al sistema en general [36].

A partir del estudio realizado se propone una arquitectura basada en componentes. Donde los componentes serán los módulos que se le añadan al sistema, siendo el paquete de elementos funcionales principales de la aplicación el núcleo del sistema. Los componentes se servirán del núcleo o entre ellos, buscando siempre la mayor independencia entre estos y siendo el núcleo totalmente independiente (figura 2.1) .

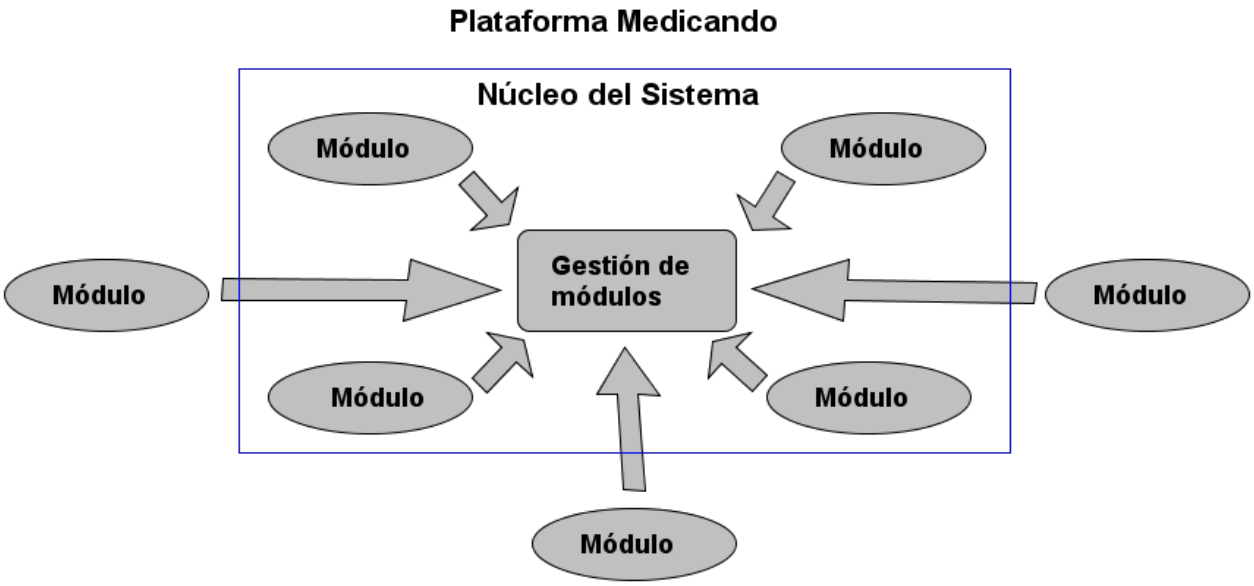

Figura 2.1. Representación abstracta de los componentes del sistema.

Para brindarle modularidad y extensibilidad al sistema se propone implementar un paquete funcional que gestione los módulos y la adición de sus configuraciones en la plataforma, de esta manera cuando se desee agregar una nueva funcionalidad no será necesario ir al código del núcleo base del sistema. Logrando con este diseño poder deshabilitar funcionalidades sin afectar el funcionamiento de otras ni de la plataforma en general. La gestión de módulos tendrá un formulario para la adición de los módulos el cual contará con los siguientes datos a introducir:

- **Nombre del módulo**: Debe ser un nombre sugerente a las funcionalidades del mismo, pues será este nombre el que aparezca en el menú de módulos.
- **Descripción**: Descripción sobre las funcionalidades del módulo, es un campo opcional.
- **Url Index**: Se debe introducir la dirección de acceso al *index* del módulo. Ejemplo de la dirección del módulo tratamientos: ***/treatment/treatment/index*.**
- **Core**: Se debe seleccionar si el módulo forma parte del núcleo del sistema.
- **Habilitado**: Se debe seleccionar el estado del módulo: habilitado o no.

- **Archivo**: Se debe agregar un fichero en formato ZIP, el cual contendrá el módulo a incorporar con una estructura específica. A continuación, se muestra la estructura que debe tener el fichero ZIP, se toma como ejemplo el módulo tratamientos.

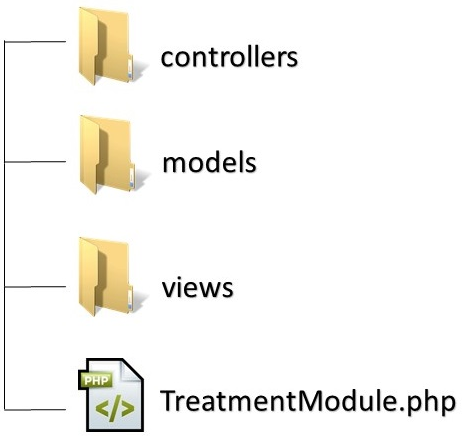

Figura 2.2. Estructura del fichero ZIP.

- **Json Config**: En este campo se debe introducir la configuración del módulo, la cual permitirá el acceso al mismo. Se debe tener en cuenta que para el correcto funcionamiento del módulo es necesario que dicha configuración se introduzca en formato JSON. A continuación se muestra un ejemplo general de los datos en formato JSON.

```
{
        "id de la clase módulo":{
                "class": "dirección del directorio de la clase módulo"

        }
}
```

Figura 2.3. Ejemplo general de los datos de la configuración en JSON.

Seguidamente se muestra un ejemplo específico, usando los datos de configuración del módulo tratamiento:

```
{
        "treatment":{
                "class": "app\\modules\\treatment\\Treatment"

        }
}
```

Figura 2.4. Ejemplo de los datos de la configuración en JSON del módulo Tratamiento.

Para organizar estructuralmente los elementos de la arquitectura se utilizan los patrones arquitectónicos. A continuación, se describe el patrón usado en la arquitectura propuesta.

Como patrón arquitectónico se propone el MVC. Este patrón es una guía para el diseño de arquitecturas de aplicaciones que ofrezcan una fuerte interactividad con usuarios. A partir del uso de *frameworks* basados en el patrón MVC se puede lograr una mejor organización del trabajo y mayor especialización de los desarrolladores y diseñadores [37].

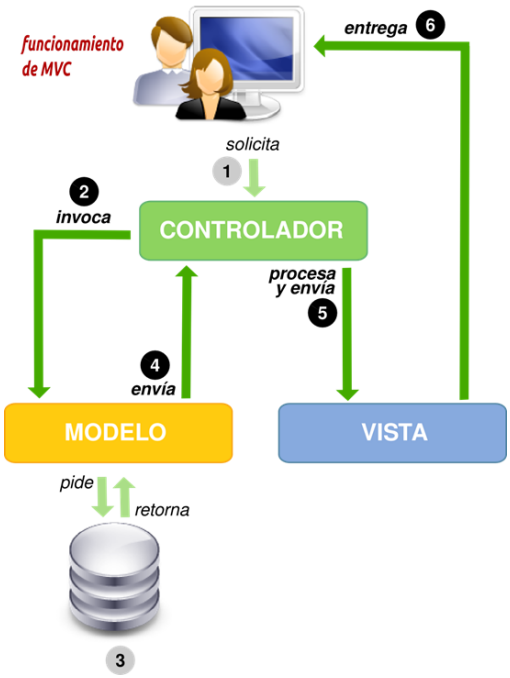

Figura 2.5. Patrón Modelo-Vista-Controlador.

En cuanto a metodologías de desarrollo, el modelado de la arquitectura en este trabajo es independiente a metodologías usadas en desarrollos anteriores del sistema, pues esta es una nueva iteración en la cual se retoma el proceso de desarrollo de la plataforma Medicando, redefiniendo su diseño arquitectónico.

Solo se utilizarán los diagramas necesarios de RUP, dado a que esta metodología propone una guía para establecer la línea base de la arquitectura de un proyecto de desarrollo de *software* basada en la modelación de las "4+1" vistas [38].

2.6. Vistas arquitectónicas

Es muy complejo capturar la AS en un sólo modelo (o diagrama). Para manejar esta complejidad se representan diferentes aspectos y características de la arquitectura en múltiples vistas. Una vista es una presentación de un modelo, la cual es una descripción completa de un sistema desde una particular perspectiva. El modelo más aceptado a la hora de establecer las vistas necesarias para describir una AS es el modelo

"4+1" de Kruchten [17].

Este modelo define 4 vistas principales:

Figura 2.6. El modelo de "4+1" vistas de la AS.

- **La vista de casos de uso o escenarios**: es una representación de los casos de uso que tienen mayor importancia para la arquitectura.
- **La vista lógica**: que comprende las abstracciones fundamentales del sistema a partir del dominio del problema.
- **La vista de proceso**: describe los aspectos de concurrencia y sincronización del diseño.
- **La vista de despliegue**: un mapeado del *software* sobre el *hardware*.
- **La vista de implementación**: la organización estática de módulos en el entorno de desarrollo.

2.7. Vista de Casos de Uso

En la vista de casos de uso se describen las funcionalidades del sistema y los actores que interactúan en la aplicación. Los casos de uso son una técnica para especificar el comportamiento de un sistema, es una secuencia de interacciones entre un sistema y alguien o algo que usa alguno de sus servicios. Un caso de uso representa una interacción típica entre un usuario y un sistema informático [39].

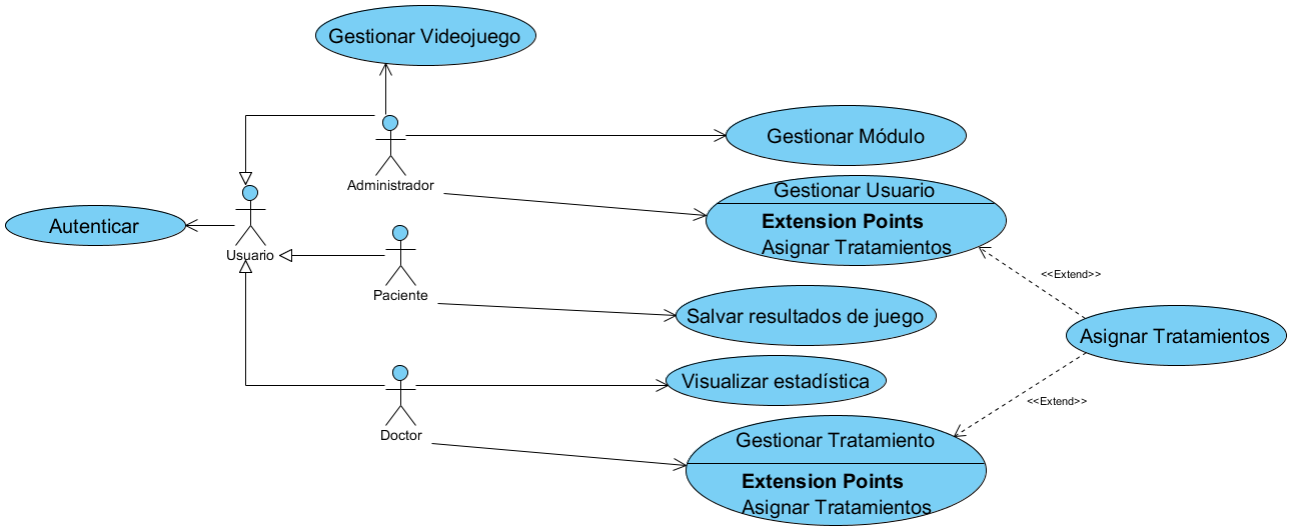

Figura 2.7. Vista de Caso de uso.

2.7.1. Descripción de los casos de uso

A continuación, se describen brevemente los casos de uso:

- **Caso de uso Autenticar**: Solicita al usuario su nombre de usuario y contraseña para guardar registro de uso y dar permisos según su rol.
- **Caso de uso Gestionar videojuego**: Permite eliminar, modificar la información o añadir videojuegos a la plataforma.
- **Caso de uso Gestionar módulo**: Permite incorporar, eliminar y modificar módulos con nuevas funcionalidades y darle un estado de habilitado o no, así como instalarlo o desinstalarlo.
- **Caso de uso Gestionar usuario**: Permite eliminar, modificar los datos y añadir usuarios al sistema.
- **Caso de uso Salvar resultados de juego**: Permite salvar la información de juego.
- **Caso de uso Gestionar tratamiento**: Permite eliminar, modificar los datos y añadir tratamientos.
- **Asignar tratamientos**: Permite asignarles tratamientos creados a pacientes que se encuentren en el sistema.
- **Caso de uso Visualizar estadística**: Permite visualizar las estadísticas generadas de la salva de juego.

2.7.2. Actores del sistema

Los actores son roles o perfiles externos al sistema que ejercen los grupos que interactúan con el sistema [40]. En este caso se tiene los usuarios, divididos en varios roles: Administrador, Paciente y Doctor. Cada uno de estos roles tienen un nivel de acceso distinto en el sistema.

Actor	Descripción
Usuario	Son todos los que interactúan con el sistema, al registrarse, dependiendo de su rol se le asigna un nivel de acceso.
Administrador	Tiene acceso para realizar cambios en las funcionalidades del sistema al poder incluir, eliminar, activar y desactivar módulos. Se encarga además de la gestión de usuarios y videojuegos.
Doctor	Los doctores pueden gestionar los tratamientos a los pacientes, así como generar y visualizar las estadísticas de las salvas de los juegos de los pacientes.
Paciente	Los usuarios con este rol pueden salvar la información del videojuego que le fue asignado en el tratamiento para el posterior análisis de la misma por parte de los doctores.

Tabla 2.2. Actores del sistema.

2.8. Vista lógica

La vista lógica comprende los elementos del diseño significativos para la arquitectura. Permite observar cómo está diseñada la funcionalidad en el interior del sistema.

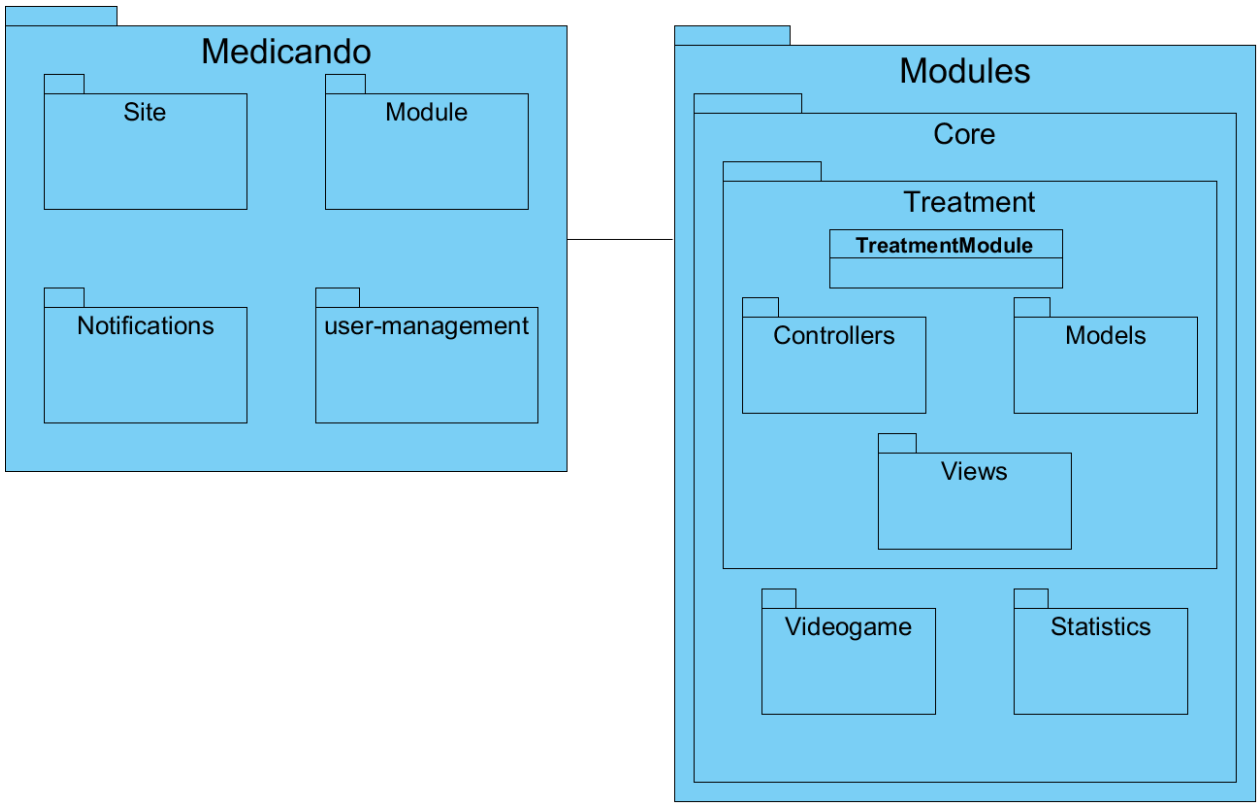

Figura 2.8. Vista lógica del sistema.

A continuación, se describen algunas de los paquetes más importantes y las funcionalidades que recogen.

- **Site**: Se encarga del control de acceso al sistema a partir de la vista para la autenticación en la plataforma. Además, contiene las vistas de navegación para invitados.
- **user-management**: Contiene los archivos de la gestión de usuarios y asignación de roles.
- **Notifications**: Gestiona las notificaciones en tiempo real en el sistema.
- **Module**: Gestiona el control y la creación de los módulos en la aplicación.
- **Modules**: Contiene los módulos con las funcionalidades del sistema, estos módulos seguirán el patrón MVC y podrán ser habilitados o deshabilitados, exceptuando las funcionalidades del núcleo. Todos tendrán una misma estructura.
    - **Core**: Contiene las funcionalidades principales e indispensables del sistema, estas funcionalidades no podrán ser deshabilitadas.

2.9. Vista de procesos

La vista de procesos explica los procesos del sistema. Representa flujos de trabajo del negocio y operaciones que conforman el sistema. Con el objetivo de que los desarrolladores comprendan mejor el funciona-

miento del proceso de la funcionalidad propuesta a implementar en la arquitectura, la cual dará modularidad y extensibilidad al sistema, se muestra el diagrama de procesos de dicha funcionalidad.

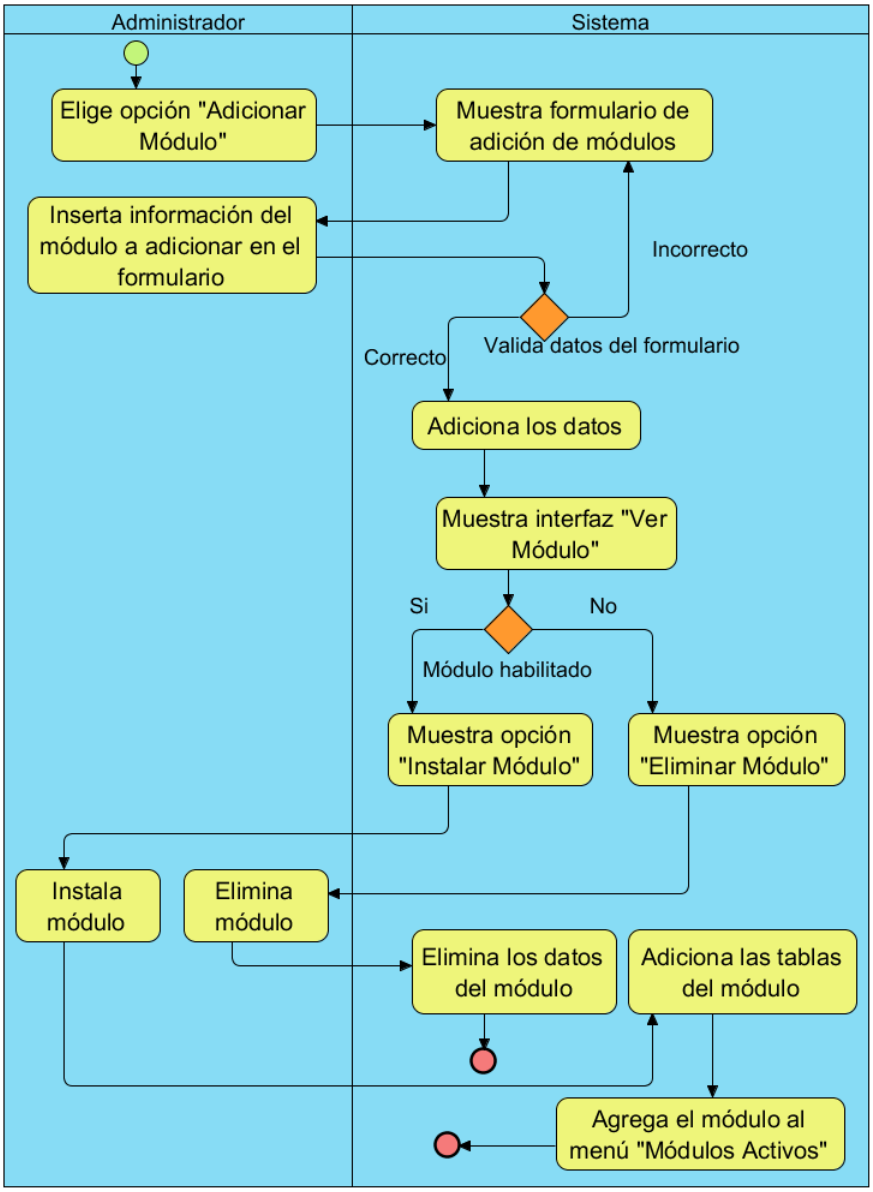

Figura 2.9. Proceso “Gestionar Módulo”.

2.10. Vista de implementación

En esta vista se muestran todos los componentes físicos del sistema: componentes, ficheros y librerías, así como las conexiones físicas entre esos componentes que conforman la solución. Se enfoca en la organización de los módulos en el entorno de desarrollo y provee una vista de la trazabilidad de los elementos de

diseño de la vista lógica ahora para la implementación. En resumen, esta vista muestra los componentes más significativos del sistema.

En la propuesta de solución realizada, todos los componentes presentan la misma estructura: cada módulo presenta su propia organización de acuerdo al patrón MVC. En este caso se toma como ejemplo el módulo de Tratamientos, este gestiona los tratamientos y la asignación de los mismos a los pacientes en la plataforma.

Descripción de los componentes del sistema

clase módulo (TreatmentModule.php): cada módulo debe tener una única clase módulo que hereda de [[yii/base/Module]][1]. Cuando se esté accediendo a un módulo, se creará una única instancia de la clase módulo correspondiente. La instancia se utiliza para compartir datos y componentes de código dentro de los módulos. Se accederá a la instancia a través de la clase [[yii/base/Applications]][2]. En esta clase deben estar implementadas las funciones para instalar y desinstalar los módulos. Para instalar se hará una consulta a la base de dato que adicione las tablas nuevas que trabaje el módulo y para desinstalar una consulta que elimine dichas tablas.

controlador (TreatmentController.php): clase que hereda de [[yii/base/Controller]][3] y se encarga de procesar las peticiones de los usuarios, generando respuestas. Los controladores analizan los datos que entran en la petición, los pasan a los modelos, posteriormente envían los datos resultantes de los modelos a las vistas y finalmente generan las respuestas de salida.

modelo (Treatment.php): clase que representa los datos de negocio, reglas y lógica. Hereda de la clase base [[yii/base/Model]][4] , la cual soporta muchas características útiles, por ejemplo:

- Atributos: representan los datos de negocio y se puede acceder a ellos como propiedades de un objeto.
- Validación: asegura la validez de los datos de entrada basándose en reglas declaradas.
- Exportación de datos: permite que los datos del modelo sean exportados en términos de arreglos.
- Asignación masiva: soporta la asignación múltiple de atributos en un único paso.

interfaces (index.php, create.php, update.php, view.php): responsables de presentar los datos al usuario final. Las vistas son manejadas por el componente de la aplicación [[yii/web/View]][5], el cual provee métodos comúnmente utilizados para facilitar el renderizado de las mismas.

El siguiente ejemplo muestra el contenido de un módulo:

[1] Es la clase base para clases de módulos.
[2] Es la clase base para todas las clases de aplicación.
[3] Es la clase base para las clases que contienen la lógica del controlador.
[4] Es la clase base para los modelos de datos.
[5] Representa un objeto de vista en el patrón MVC.

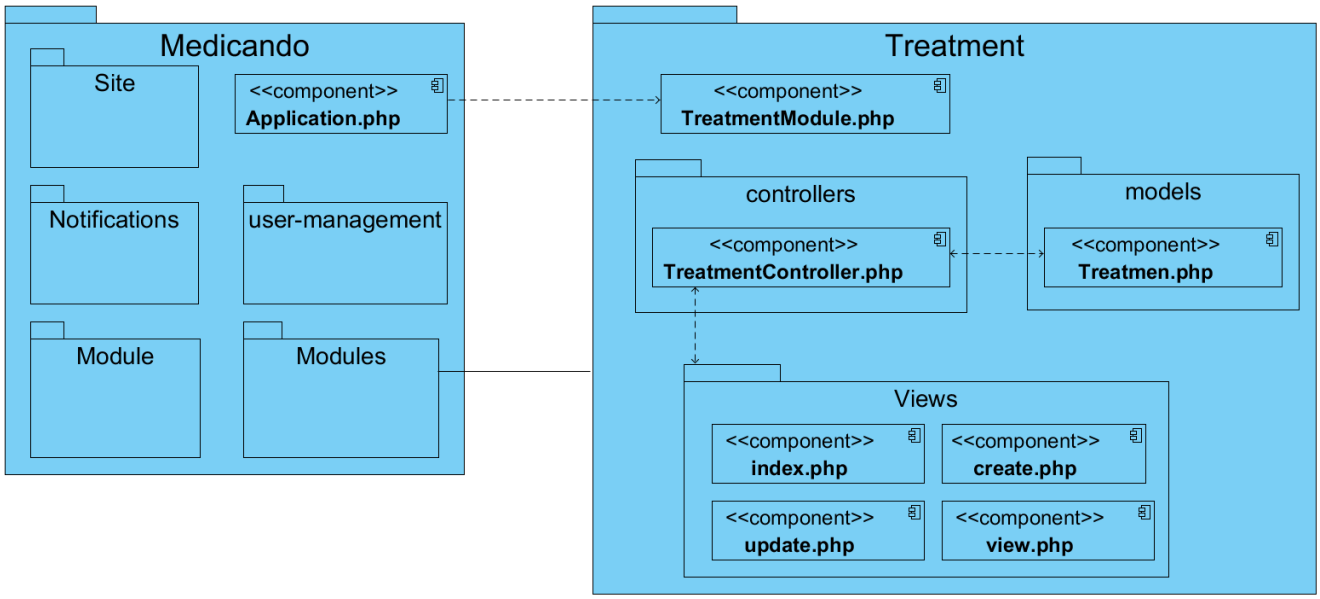

Figura 2.10. Vista de implementación de un módulo.

2.11. Vista de despliegue

En esta vista se muestra la configuración física de un sistema para su ejecución.

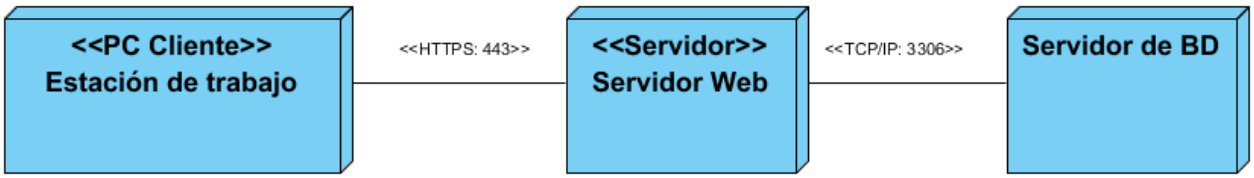

Figura 2.11. Vista de Despliegue.

Estación de trabajo: Ordenador cliente desde donde el usuario podrá visualizar e interactuar con el sistema a través de un navegador web que debe estar previamente instalado en la máquina.

Servidor Web: Contiene la aplicación web a la que se conectan los clientes por medio de sus estaciones de trabajo. Este servidor debe tener las medidas de seguridad y protección pertinentes.

Servidor de Base de Datos: En este servidor se encuentran almacenados los datos asociados al sistema.

Descripción de los protocolos de comunicación

HTTPS (Protocolo Seguro de Transferencia Hipertexto): Protocolo que se describe como HTTP (protocolo estándar básico de la arquitectura *web*) pero que utiliza un cifrado basado en SSL/TLS, Capa de puertos Seguros (SSL, por sus siglas en inglés *Secure Sockets Layer*) y Seguridad de la Capa de Transporte (TLS, por sus siglas en inglés *Transport Layer Security*), los cuales son protocolos para enviar paquetes cifrados

a través de *Internet*, lo que permite que la información del sistema que viaja por la red lo haga de forma segura.

TCP/IP: Protocolo de Control de Transmisión (TCP, por sus siglas en inglés *Transmission Control Protocol*) y Protocolo de Internet (IP, por sus siglas en inglés *Internet Protocol*), es un protocolo que representa la manera en la que se realizan las comunicaciones entre el servidor *web* y el servidor de Bases de Datos. Para garantizar la seguridad del envío de información se usa el protocolo SSL/TLS, el cual proporciona autenticación de servidores, integridad de mensajes y autenticación de cliente para conexiones TCP/IP.

2.12. Conclusiones del capítulo

En este capítulo se logró definir, como propuesta de solución, la arquitectura para la nueva versión de la plataforma Medicando. Se definen los requisitos funcionales del sistema y la línea base de la arquitectura propuesta, donde se tuvo en cuenta el estilo y patrón arquitectónico definidos, con el objetivo de que los desarrolladores tengan una guía a la hora de implementar las funcionalidades del sistema y se utilizan las vistas propuestas por Kruchten para representar la arquitectura.

CAPÍTULO 3

Evaluación de la arquitectura

3.1. Introducción al capítulo

En este capítulo se realizará el proceso de evaluación de la arquitectura propuesta mediante el método ATAM y a través de las técnicas de evaluación basada en prototipo y en escenarios. Se analizan e identifican riesgos potenciales en el diseño, que puedan afectar al sistema. Se verifica en que grado se satisfacen los atributos de calidad en la arquitectura.

3.2. Evaluación de la arquitectura propuesta

En conjunto con el método ATAM, se seleccionan las técnicas basada en escenarios, dado a que en uno de los pasos que se ejecutarán del método se especificarán los escenarios que validarán los atributos de calidad definidos y basada en prototipo, pues para evaluar de forma práctica la arquitectura se desarrolló un prototipo funcional. El ATAM, como se explicó anteriormente, está compuesto por nueve pasos divididos en cuatro fases, los cuales se adaptarán de acuerdo a los requerimientos del proyecto.

3.2.1. Árbol de Utilidad

En la tabla 3.1 se muestra el árbol de utilidad correspondiente al paso 5 de la fase 2 de ATAM. La prioridad se establece teniendo en cuenta los siguientes aspectos: la necesidad de alcanzar el requisito y la dificultad de alcanzar dicho requisito. La prioridad de estos aspectos se mostrará en este mismo orden y para mayor facilidad y entendimiento se utilizará la siguiente escala: Alta(A), Media(M) y Baja(B).

Tabla 3.1. Árbol de utilidad.

Atributo	Subcaracterística	Escenario	Prioridad
Funcionalidad	• Corrección funcional	Gestión de módulos.	(A,M)
Seguridad	• Integridad • Acceso Restringido • Confidencialidad	Autenticación de usuarios. Las funcionalidades del sistema se muestran en dependencia de los permisos que tenga el usuario que esté autenticado.	(A,M)
Confiabilidad	• Recuperabilidad • Madurez	Estabilidad del sistema ante la ocurrencia de errores.	(M,B)
Usabilidad	• Operabilidad • Agradabilidad	Interfaces amigables y sencillas de usar.	(M,B)
Eficiencia	• Comportamiento temporal • Utilización de recursos	Los tiempos de respuestas a las peticiones efectuadas son mínimos. Recursos mínimos.	(M,M)
Mantenibilidad	• Modularidad • Capacidad para ser modificado • Capacidad para ser probado.	Sistema capaz de ser modificado eficientemente y permitir adicionar funcionalidades sin afectar al resto. Los requisitos funcionales y no funcionales descritos al inicio del diseño de la arquitectura brindan las pautas para establecer escenarios de prueba.	(A,M)

Portabilidad	• Adaptabilidad	Capaz de funcionar correctamente en varios sistemas operativos.	(A,M)
Interoperabilidad	• Coexistencia	Capaz de coexistir con otro *software* independiente, en un entorno común, compartiendo recursos comunes sin detrimento.	(A,M)

3.2.2. Especificación de los escenarios

Las tablas siguientes muestran el tratamiento de los escenarios a través del método ATAM como parte del proceso de evaluación de la arquitectura.

Tabla 3.2. Escenario: Gestión de módulos.

Atributo de Calidad	Adecuación Funcional.
Subatributos/Subcaracterísticas	Corrección.
Objetivo	El sistema debe cumplir las tareas para las que fue creado.
Origen	Administrador del sistema.
Artefacto	Gestión de módulos.
Entorno	El sistema funciona correctamente.
1. A. Gestión de módulos funcionales en el sistema. (Adicionar módulo)	**Respuesta: Flujo de eventos (Escenarios)**
Se selecciona la opción *Adicionar módulo* en la interfaz *Módulos*, se llena el formulario y se selecciona la opción *Adicionar*.	El sistema guarda la información del nuevo módulo y añade su configuración a la del sistema.
1. B. Gestión de módulos funcionales en el sistema. (Eliminar módulo)	
Se selecciona la opción *Eliminar módulo* en la interfaz *Módulos*.	Se elimina el módulo y su configuración.
1. C. Gestión de módulos funcionales en el sistema. (Habilitar módulo)	
Se selecciona la opción *Habilitar módulo* en la interfaz de *Actualizar* del módulo seleccionado.	Si el módulo está deshabilitado se habilitan las funcionalidades del módulo a los usuarios.

1. D. Gestión de módulos funcionales en el sistema. (Deshabilitar módulo)	
Se selecciona la opción *Deshabilitar módulo* en la interfaz de *Actualizar* del módulo seleccionado.	Si el módulo está habilitado se deshabilitan las funcionalidades del módulo a los usuarios.
Medida de respuesta	
Navegar en el sistema.	

Tabla 3.3. Escenario: Autenticación de usuarios.

Atributo de Calidad	Seguridad.
Subatributos/Subcaracterísticas	Integridad.
Objetivo	Asegurar que los datos del sistema sean accesibles por las personas autorizadas.
Origen	Usuarios del sistema.
Artefacto	El sistema.
Entorno	El sistema funciona correctamente.
1. A. Permitir el acceso a personas autorizadas	**Respuesta: Flujo de eventos (Escenarios)**
Autenticarse en el sistema.	1. El sistema muestra la pantalla de autenticación al usuario. 2. El usuario escribe los datos de su cuenta(usuario y contraseña) y presiona el botón “Aceptar”. 3. El sistema verifica los datos y en caso correcto permite al usuario acceder a las funcionalidades de la plataforma. 4. En caso contrario el sistema impide el acceso al usuario.
Medida de respuesta	
Navegar en el sistema.	

Tabla 3.4. Escenario: Las funcionalidades del sistema se muestran en dependencia de los permisos que tenga el usuario que esté autenticado.

Atributo de Calidad	Seguridad.
Subatributos/Subcaracterísticas	Acceso restringido-Confidencialidad.
Objetivo	Proteger el sistema contra el acceso a funcionalidades, datos e información no autorizados.
Origen	Usuarios del sistema.
Artefacto	El sistema.
Entorno	El sistema funciona correctamente.
1. A. Permitir el acceso a los datos a personas autorizadas	**Respuesta: Flujo de eventos (Escenarios)**
Acceder a las funcionalidades e información del sistema.	1. Una vez autenticado accede a las funcionalidades la aplicación. 2. El sistema da acceso al usuario sobre las distintas funcionalidades e informaciones según los permisos del rol al que pertenece.
Medida de respuesta	
Navegar en el sistema.	

Tabla 3.5. Escenario: Interfaces amigables y sencillas de usar.

Atributo de Calidad	Usabilidad.
Subatributos/Subcaracterísticas	Operabilidad-Agradabilidad.
Objetivo	Permitir al usuario operar y controlar el sistema con facilidad.
Origen	Usuarios del sistema.
Artefacto	Las interfaces pertenecientes a todos los requisitos del sistema.
Entorno	El sistema funciona correctamente.
1. A. Pasos a ejecutar para llegar a los requerimientos del sistema.	**Respuesta: Flujo de eventos (Escenarios)**

Interactuar con los requerimientos del sistema.	El usuario autenticado puede acceder con facilidad a las funcionalidades, a las cuales tenga acceso según su rol, mediante interfaces intuitivas y que no requieren una especialización en el tema para su entendimiento.
Medida de respuesta	
Navegar en el sistema.	

Tabla 3.6. Escenario: Recursos mínimos.

Atributo de Calidad	Eficiencia.
Subatributos/Subcaracterísticas	Utilización de recursos.
Objetivo	Mantener al mínimo el uso de recursos.
Origen	Usuarios del sistema.
Artefacto	El sistema.
Entorno	El sistema funciona correctamente.
1. A. Uso de recursos ante alguna acción realizada por el usuario.	**Respuesta: Flujo de eventos (Escenarios)**

Los usuarios navegan y hacen uso de las funcionalidades del sistema.	El sistema necesita como requerimientos mínimos: **De *software*** • Para el cliente: Sistema Operativo Windows o Linux, navegador *web* estándar. • Para el servidor, se recomienda: Sistema Operativo Windows o Linux, servidor *web* Apache, gestor de base de datos MySQL. Se recomienda la utilización del paquete de programas XAMPP, ya que es multiplataforma e incluye Apache, PHP y MySQL por defecto. **De *hardware*** • Para el cliente: Procesador Pentium IV, 256 MB de RAM, 100 MB de disco duro. • Para el servidor: Procesador Pentium IV a 2.8 GHz, 1 GB de RAM, 80 GB de disco duro.
Medida de respuesta	
Debe ser posibles la navegación en el sistema con Ordenadores clientes y servidores con estos requisitos.	

Tabla 3.7. Escenario: Sistema capaz de ser modificado eficientemente y permitir adicionar funcionalidades sin afectar al resto.

Atributo de Calidad	Mantenibilidad.
Subatributos/Subcaracterísticas	Modularidad-Capacidad para ser modificado.
Objetivo	Permitir que sea modificado de forma efectiva y eficiente sin introducir defectos o degradar el desempeño y que un cambio en un componente tenga un impacto mínimo en los demás.

Origen	Arquitecto de *software*.
Artefacto	El código fuente.
Entorno	Desarrollo del sistema.
1. A. Capacidad de modificación del sistema.	**Respuesta: Flujo de eventos (Escenarios)**
La arquitectura del sistema está diseñada para ser capaz de permitir que una adición de un componente o un cambio en alguno de estos tenga un impacto mínimo en los demás.	
Medida de respuesta	
Introducir un módulo al sistema.	

Tabla 3.8. Escenario: Los requisitos funcionales y no funcionales descritos al inicio del diseño de la arquitectura brindan las pautas para establecer escenarios de prueba.

Atributo de Calidad	Mantenibilidad.
Subatributos/Subcaracterísticas	Capacidad para ser probado.
Objetivo	Establecer criterios de prueba para un sistema o componente y con la que se pueden llevar a cabo las pruebas para determinar si se cumplen dichos criterios.
Origen	Arquitecto de *software*.
Artefacto	El sistema.
Entorno	El sistema funciona correctamente.
1. A. Facilidad con la que se pueden establecer criterios de prueba para el sistema.	**Respuesta: Flujo de eventos (Escenarios)**
Desde la concepción inicial del sistema se definen y delimitan las funciones según los requisitos funcionales y no funcionales a implementar, elementos que favorecen el proceso de identificación y elaboración de los distintos escenarios de prueba.	
Medida de respuesta	
Escenario de prueba del sistema.	

Tabla 3.9. Escenario: Capaz de funcionar correctamente en varios sistemas operativos.

Atributo de Calidad	Portabilidad.
Subatributos/Subcaracterísticas	Adaptabilidad.
Objetivo	Capacidad del producto que le permite ser adaptado de forma efectiva y eficiente a diferentes entornos determinados de *hardware*, *software*, operacionales o de uso.
Origen	Arquitecto de *software*.
Artefacto	El sistema.
Entorno	El sistema funciona correctamente.
1. A. Capacidad del sistema de adaptarse de forma efectiva a diferentes entornos.	**Respuesta: Flujo de eventos (Escenarios)**
El sistema está diseñado con tecnologías que permiten que este se adapte a varios sistemas operativos: Windows, Linux.	
Medida de respuesta	
El ambiente de despliegue del sistema.	

Tabla 3.10. Escenario: Capaz de coexistir con otro *software* independiente, en un entorno común, compartiendo recursos comunes sin detrimento.

Atributo de Calidad	Interoperabilidad.
Subatributos/Subcaracterísticas	Coexistencia.
Objetivo	El sistema debe llevar a cabo sus funciones requeridas aún cuando comparta el mismo entorno *hardware* o *software* con otras aplicaciones.
Origen	Arquitecto de *software*.
Artefacto	El sistema.
Entorno	El sistema funciona correctamente.
1. A. Coexistencia del sistema con otras aplicaciones.	**Respuesta: Flujo de eventos (Escenarios)**
El sistema se encuentra en un servidor donde coexisten otros sistemas usando los mismos recursos.	El sistema funciona correctamente.
Medida de respuesta	
El ambiente de despliegue del sistema.	

Tabla 3.11. Escenario: Los tiempos de respuestas a las peticiones efectuadas son mínimos.

Atributo de Calidad	Eficiencia.
Subatributos/Subcaracterísticas	Comportamiento temporal.
Objetivo	Mantener al mínimo los tiempos de respuesta.
Origen	Usuarios del sistema.
Artefacto	El sistema.
Entorno	El sistema funciona correctamente.
1. A. Tiempo de respuesta ante alguna acción de los usuarios.	**Respuesta: Flujo de eventos (Escenarios)**
Los usuarios navegan y hacen uso de las funcionalidades del sistema.	El tiempo de respuesta máximo del sistema ante las acciones de los usuarios debe estar en el rango de 5-8 ms.
Medida de respuesta	
Los tiempos de respuesta máximo del sistema deben ser de 5-8 ms.	

3.2.3. Resultados de la evaluación con ATAM

Basada en la información recolectada haciendo uso del método ATAM se pudo identificar 1 riesgo en el escenario 1 (ver tabla 3.2). El riesgo encontrado, demuestra que el sistema puede presentar problemas en caso de que la información y datos que se administran en este escenario sean mal manejados por los usuarios que los gestionan. Luego del análisis del riesgo detectado se determinó que dicha información será manejada por administradores del sistema, expertos y conocedores del mismo.

3.3. Técnica de evaluación: Prototipo

Como cierre del ciclo de desarrollo de la arquitectura, se evalúa la misma mediante la técnica de evaluación basada en prototipo. Usando esta técnica se implementó un prototipo funcional, siguiendo el estilo y patrón arquitectónico indicados, del paquete funcional *Module* (Gestión de módulos) propuesto en la arquitectura con el fin de brindar modularidad al sistema. De esta manera se evalúa de forma práctica si el mismo cumple con este fin, integrando el módulo Tratamiento sin la necesidad de modificar código en el sistema.

La creación de este prototipo permitió evaluar satisfactoriamente los atributos de calidad de interés para los usuarios, tal es el caso de la modularidad, lo cual le brinda extensibilidad al sistema. De esta forma se pudo comprobar que la arquitectura propuesta satisface los principales objetivos definidos para la investigación.

3.4. Discusión general sobre la arquitectura propuesta

Luego de la aplicación del ATAM y la técnica de evaluación basada en prototipo, se comprobó que el empleo de la arquitectura propuesta influye positivamente en los atributos de calidad que se ven afectados con el uso de la arquitectura actual. A continuación, se especifica el efecto de la aplicación de la arquitectura propuesta sobre los principales atributos de calidad:

Modificabilidad - Mantenibilidad: para la actualización o adición de nuevas funcionalidades, no es necesario repetir el proceso completo de despliegue, solo se necesita copiar al directorio ***modules***, ubicado en la carpeta del sistema, el módulo que contiene las nuevas funcionalidades y adicionarlo desde la interfaz visual "Gestionar módulos".

Reusabilidad - Escalabilidad: las funciones definidas en el paquete funcional *Module*, pueden emplearse para construir dinámicamente la aplicación, esto significa que la adición de un módulo registra automáticamente su configuración en el sistema.

Conclusiones

Con la realización de este trabajo se dio cumplimiento al objetivo propuesto al inicio de la investigación ya que:

- Se definió, haciendo uso del estilo basado en componentes con un diseño modular, una arquitectura de software para la plataforma Medicando, que redujo los problemas de extensibilidad y dependencias que presenta la arquitectura actual.
- Se validó la propuesta de solución mediante el método ATAM y las técnicas basadas en escenarios y prototipo; a pesar de que arrojaron la presencia de riesgos, los cuales fueron registrados, se pudo comprobar que la arquitectura satisface los atributos de calidad propuestos por la norma ISO/IEC 25010.

Recomendaciones

Después de analizados los resultados obtenidos en el presente trabajo, se recomienda:

- Implementar el resto de módulos que conforman la plataforma haciendo uso de la arquitectura propuesta.
- Tener en cuenta el riesgo descrito en los resultados de la evaluación con ATAM.

Acrónimos

ADR Revisión Activa del Diseño. 19

ARID Revisiones Activas para Diseño Intermedio. 19

AS Arquitectura de *Software*. 2, 8, 9, 13, 16, 18, 19, 21–23, 26, 27

ATAM Método de Análisis de Acuerdos de Arquitectura. 7, 19–21, 35, 37, 44, 45, 47

IEC Comisión Internacional Electrónica. 17, 18, 20, 21, 23, 46

ISO Organización Internacional de Normalización. 17, 18, 20, 21, 23, 46

MVC Modelo-Vista-Controlador. 11, 14–16, 21, 26, 30, 32

RUP Proceso Unificado Racional. 26

SAAM Método de Análisis para Arquitecturas de Software. 18, 19

UCI Universidad de las Ciencias Informáticas. 1, 7

Vertex Centro de Entornos Interactivos 3D. 1, 5

Referencias bibliográficas

[1] Rubén Cañedo Andalia. «Ciencia y tecnología en la sociedad: Perspectiva histórico-conceptual». En: *Acimed* 9.1 (2001), págs. 72-76 (vid. pág. 1).

[2] D. Outomuro. «Tecnología y salud, Impacto de la tecnología en la práctica de la medicina». En: *Universidad de Buenos Aires* (2014) (vid. pág. 1).

[3] J. Altés. «Papel de las tecnologías de la información y la comunicación en la medicina actual». En: *Seminarios de la Fundación Española de Reumatología* 14.2 (2013), págs. 31-35. ISSN: 1577-3566. DOI: http://dx.doi.org/10.1016/j.semreu.2013.01.005. URL: http://www.sciencedirect.com/science/article/pii/S1577356613000067 (vid. pág. 1).

[4] rae. Real Academia de la Lengua Española. 2017. URL: http://www.rae.es (vid. pág. 3).

[5] M. Rojas y A. Montiel. «Plataforma para el desarrollo de Sistemas de Informacion Geográfica para dispositivos móviles en la Universidad de las Ciencias Informáticas». En: (2012). URL: http://bibliodoc.uci.cu/RDigitales/2013/febrero/18/TD_05855_12.pdf (vid. pág. 3).

[6] Red de Aprendizaje. *Plataforma Informática*. 2017. URL: http://www.reddeaprendizaje.com/inicio/item/47-plataforma-informatica (vid. pág. 3).

[7] Virtualrehab.info. *Virtualrehab | Virtual Rehabilitation System*. 2017. URL: http://www.virtualrehab.info/es (vid. pág. 4).

[8] Neuroathome.org. *NeuroAtHome | Rehabilitación Virtual | Neurorehabilitación Telerehabilitación Kinect*. 2017. URL: http://www.neuroathome.org/p/home.html (vid. pág. 4).

[9] Gaikai. *Gaikai*. 2017. URL: https://www.gaikai.com/ (vid. pág. 4).

[10] J. G. Valdés Díaz. «Módulos de visualización de la información paciente-especialista para la plataforma de gestión de videojuegos Medicando.» Trabajo de diploma para optar por el título de Ingeniero en Ciencias Informáticas. Universidad de las Ciencias Informáticas, 2016 (vid. págs. 5, 6).

[11] E. Gutiérrez Ramos. «Plataforma web para la gestión de videojuegos serios de navegador con fines terapéuticos (Medicando)». Trabajo de diploma para optar por el título de Ingeniero en Ciencias Informáticas. Universidad de las Ciencias Informáticas, 2015 (vid. pág. 5).

[12] D. López y R. Barrio. «Especificación de la arquitectura de la plataforma software Bizkaisense.» En: *Universidad de Deusto* (2012) (vid. pág. 6).

[13] E Martínez Pérez y S Pérez Cancio. «Propuesta de arquitectura basada en componetes para plataformas de gestión de procesos desarrolladas en Django.» En: *Universidad de las Ciencias Informáticas-Facultad Regional de Granma* (2012) (vid. pág. 8).

[14] Hernán Astudillo. «Arquitectura de Software». En: (2014) (vid. pág. 8).

[15] Carlos Billy Reynoso. «Introducción a la Arquitectura de Software». En: *Universidad de Buenos Aires* 33 (2004) (vid. pág. 8).

[16] P. Clements y col. «Documenting Software Architecture». En: (2002) (vid. págs. 8, 10).

[17] P. Kruchten. «Architectural Blueprints - The "4+1" View Model of Software Architecture». En: (1995) (vid. págs. 9, 27).

[18] N. Kiccillof y C. B. Reynoso. «Estilos y Patrones en la Estrategia de Arquitectura de Microsoft». En: (2004) (vid. págs. 10, 13, 15).

[19] A. Szyperski. «Component software: Beyond Object-Oriented programming.» En: (2010) (vid. pág. 11).

[20] C. Llorente y et al. «Guía de Arquitectura N-Capas orientada al Dominio con .NET 4.0 (Beta).» En: (2010) (vid. pág. 12).

[21] L. Fuentes, J. Troya y A. Vallecillo. «Desarrollo de Software Basado en Componentes». En: *España : Universidad de Malaga* () (vid. pág. 12).

[22] D. Garlan y M. Shaw. «An introduction to software architecture.» En: *Carnegie Mellon University. School of Computer Science* (1994) (vid. pág. 12).

[23] R. S. Pressman. *INGENIERÍA DEL SOFTWARE. UN ENFOQUE PRÁCTICO*. McGRAW-HILL INTERAMERICANA EDITORES, 2010 (vid. pág. 14).

[24] J.S Garrido. «Arquitectura y diseño de sistemas Web modernos». En: *InforMAS, Revista de Ingeniería Informática del CIIRM* 1 (2004) (vid. pág. 14).

[25] A. Trellini. «Arquitectura y Diseño de Sistemas». En: (2015) (vid. pág. 14).

[26] E. Lleonart. «Patrones». En: *Facultad de Informática - Universidad Politécnica de Valencia* (2016) (vid. pág. 16).

[27] E. Marquina. «Guía de Patrones, Prácticas y Arquitectura .NET (Versión 2.0).» En: *Microsoft Service, Organización Contoso* (2008) (vid. pág. 16).

[28] M. Dávila y col. «Evaluación de Arquitecturas de Software». En: (2016) (vid. págs. 17, 19).

[29] R. Kazman, M. Klein y P. Clements. «Evaluating software architectures: methods and case studies.» En: (2002) (vid. pág. 17).

[30] O Salvador Gómez. «Evaluando Arquitecturas de Software. Parte 1». En: *Panorama General* 1 (2007), págs. 1870-0888 (vid. pág. 17).

[31] ISO/IEC 25000. *Portal ISO 25000*. 2017. URL: http://iso250000.com (vid. pág. 17).

[32] J. Russell y R. Cohon. «Architecture Tradeoff Analysis Method». En: (2012) (vid. pág. 19).

[33] F. Perez. «ARID, Active Reviews for Intermediate Designs». En: (2012) (vid. pág. 19).

[34] J. Bosch. «Design and use of software architectures: adopting and evolving a product-line approach.» En: (2000) (vid. págs. 20, 21).

[35] L. Bass, R. Kazman y P. Clements. «Software Architecture in Practice. 2da Edición.» En: *Addison-Wesley Professional* (2003) (vid. pág. 21).

[36] Andrés Vignaga y Daniel Perovich. «Enfoque metodológico para el desarrollo basado en componentes». En: 21 (2003) (vid. pág. 24).

[37] Yanette Díaz González y Yenisleidy Fernández Romero. «Patrón Modelo-Vista-Controlador.» En: *Revista Telem@ tica* 11.1 (2012), págs. 47-57 (vid. pág. 26).

[38] Alina Dolores Rodríguez Peña y Luis Guillermo Silva Rojas. «Arquitectura de software para el sistema de visualización médica Vismedic». En: *Revista Cubana de Informática Médica* 8.1 (2016), págs. 75-86 (vid. pág. 26).

[39] Santiago Ceria. «Casos de Uso Un Método Práctico para Explorar Requerimientos». En: (2016) (vid. pág. 27).

[40] Vanesa E. Berasa. «CASOS DE USO: SISTEMA DE AUTOARCHIVO CLACSO». En: *Universidad de Buenos Aires. Facultad de Filosofía y Letras. Depto. de Bibliotecología y Ciencias de la Información Práctica Profesional* (2011) (vid. pág. 28).