Universidad de las Ciencias Informáticas

Facultad 6

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

Título: Arquitectura del tablero digital para la línea Integración de Soluciones.

Autor: Noel Valle Castro.

Tutores: MSc. Reynaldo Álvarez Luna.

Ing. Jorge Bedoya Rusenko

La Habana, Cuba, junio de 2012

"Año 54 de la Revolución"

"Programar sin una arquitectura o diseño en mente es como explorar una gruta sólo con una linterna: no sabes dónde estás, dónde has estado ni hacia dónde vas"

Danny Thorpe.

Declaración de Autoría

Declaro ser autor del presente trabajo de diploma y reconozco a la Universidad de las Ciencias Informáticas los derechos patrimoniales del mismo, con carácter exclusivo.

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

Noel Valle Castro

_____

[Firma Autor]

MSc. Reynaldo Álvarez Luna.

_____

[Firma Tutor]

Ing. Jorge Bedoya Rusenko

_____

[Firma Tutor]

Datos de Contacto

Nombre: Reynaldo Álvarez Luna

Ingeniero en Ciencias Informáticas, UCI 2008

Máster en Informática Aplicada, UCI 2011

Correo: rluna@uci.cu

Nombre: Jorge Bedoya Rusenko

Ingeniero en Ciencias Informáticas, UCI 2010

Correo: jbedoya@uci.cu

AGRADECIMIENTOS

En especial a mis padres, por su esfuerzo, confianza, apoyo, sacrificio y amor de toda la vida.

A mis hermanos por ser tan especiales.

A toda mi familia, por su cariño y apoyo incondicional.

A todos mis amigos, no los nombro para que no se quede ninguno. Aunque no nos veamos nunca más, siempre estarán presentes.

A todos los profesores que de una forma u otra ayudaron a realizar mi sueño.

A mis tutores, gracias por el tiempo dedicado.

A Fidel por crear esta magnífica escuela y darme la oportunidad de conocer gente tan maravillosa.

DEDICATORIA

A mis padres por ser un ejemplo a seguir en todo momento y sobre todo por esperar siempre lo mejor de mí.

RESUMEN

La investigación surge en el marco de trabajo del proyecto Plataforma de Apoyo a la Toma de Decisiones y Soluciones Integrales (PATDSI), desarrollado en el Centro de Tecnologías de Gestión de Datos (DATEC) de la Universidad de las Ciencias Informáticas (UCI). Dada la necesidad de integrar los módulos de un tablero digital PATDSI con el fin agilizar el proceso de toma de decisiones, se decidió diseñar una arquitectura que permita esta integración. La arquitectura diseñada proporciona los elementos necesarios para el desarrollo de una aplicación que brinde a los especialistas en estadísticas un entorno de trabajo provisto de las funcionalidades necesarias para la evolución de sus investigaciones. Su éxito viene dado por las decisiones arquitectónicas que fueron tomadas durante su desarrollo, por la definición de herramientas, tecnologías y aspectos esenciales del diseño.

ÍNDICE.

INTRODUCCIÓN

Con el surgimiento de las primeras empresas, surgió también la competencia de producir con mayor calidad, prestar los mejores servicios o alcanzar un mayor control en el mercado. Con este fin la mayoría de las organizaciones en la actualidad mantienen una constante lucha, no sólo por permanecer, sino también por continuar desarrollándose siguiendo algunos paradigmas, pero a la vez siendo creativos. Para el crecimiento empresarial se hace necesario que los directivos de las empresas mantengan un estricto control de los datos de las mismas y a su vez estén provistos de herramientas para la toma de decisiones.

A finales del siglo XVIII William Playfair (1759-1823) expone y defiende su idea de que los gráficos brindan una comunicación más eficiente que las tablas y los contenidos escritos. Es considerado como el inventor de los gráficos lineales, de barras y de sectores. Playfair publicó el libro titulado “The Commercial and Political Atlas” (1786) el cual contiene 43 gráficos de series de tiempo y por primera vez, es usado un gráfico de barras. En 1801 utiliza el primer gráfico de sectores en su obra “Playfair’s Statistical Breviary”. O sea que desde hace mucho tiempo se conoce que mediante la modelación de los procesos y los datos, convirtiéndolos en gráficas se logra un mejor entendimiento de la situación existente. Por todo esto surge como herramienta principal para la toma de decisiones la modelación del contenido. [1]

Con el desarrollo de las nuevas tecnologías, y potenciada por la competencia entre organizaciones, poco a poco se ha ido generalizando la informatización de las empresas. Por lo que las acciones y procedimientos que antes eran de forma manual, física o especulativa, se fueron automatizando. Las herramientas de apoyo para la toma de decisiones no están ajenas al movimiento tecnológico mundial, por lo que de forma colateral con las empresas, se han ido transformando en herramientas informáticas para la toma de decisiones.

En la Universidad de las Ciencias Informáticas (UCI) también se realizan implementaciones de sistemas para la toma de decisiones. La UCI forma parte de un programa de la revolución cubana que surge como alternativa económica con un sinnúmero de perspectivas futuras. En el Centro de Tecnologías de Gestión de Datos (DATEC) de la UCI se encuentra en desarrollo la Plataforma de Apoyo a la Toma de Decisiones y Soluciones Integrales (PATDSI) perteneciente a la línea de Integración de Soluciones. Proyecto que tiene el objetivo de desarrollar soluciones informáticas que agilicen la captura y el análisis de la información que requiere el cliente para la toma de decisiones. Perteneciente a esas soluciones se encuentra desarrollado el Sistema Integrado de Gestión Estadística (SIGE) y en desarrollo la Solución Integral de Gestión de Datos (SIGDAT) que

garantizarán todo el proceso de captura y gestión de la información en los diferentes niveles de la empresa.

Como parte de este paquete de soluciones se ha identificado la necesidad de una herramienta informática para el análisis de la información gestionada por los sistemas mencionados; la visualización oportuna de los datos y un alto nivel de actualización contribuyen a elevar la capacidad y certeza de los usuarios. Los tableros digitales o Dashboard, como se les conoce también por su nombre en inglés constituyen la herramienta más generalizada en el proceso de toma de decisiones en la empresa mediante gráficos e indicadores que se obtienen a través de documentos, plantillas o bases de datos. Un tablero digital es una herramienta informática, usada principalmente en el campo de la administración de las empresas, la cual puede ser aplicada en cualquier organización y a distintos niveles de la misma, su objetivo principal es precisar de forma casi segura un contexto determinado dentro de la compañía.

Sin embargo debido a que esta solución requiere la comunicación con los sistemas que gestionan la información y la interacción con diferentes modelos de datos, así como que deberá estar orientada al desarrollo como un activo de software para ser integrado a otras soluciones del centro que lo requieran; no puede ser desarrollada según la totalidad de los principios establecidos en la arquitectura base del proyecto.

A partir de las dificultades identificadas se plantea como **problema de la investigación**:

¿Cómo diseñar la estructura de un tablero digital para su integración a las soluciones de DATEC que lo requieran?

La investigación tiene como **objeto de estudio** la arquitectura de software de sistemas para la toma de decisiones, y el **campo de acción** está enmarcado en la arquitectura de software para el tablero digital de la línea Integración de Soluciones.

Para solucionar el problema de la investigación se define como **objetivo general** establecer la arquitectura del tablero digital para la línea Integración de Soluciones. Este objetivo para su mayor comprensión estará dividido en los siguientes **objetivos específicos**:

1. Describir la línea base de la arquitectura.
2. Representar el modelo de la arquitectura.
3. Implementar componentes necesarios para la integración del tablero digital a sistemas externos.
4. Evaluar el diseño de la arquitectura propuesto.

Para dar cumplimiento a cada uno de estos objetivos es necesario expresarlos en **tareas de la investigación**, las cuales son expresadas a continuación:

- Identificación de los elementos reutilizables de la arquitectura base de la línea Integración de Soluciones.
- Definición de las herramientas y tecnologías de software para el desarrollo.
- Selección de un estilo arquitectónico para el diseño de la arquitectura.
- Selección de los patrones arquitectónicos y fundamentación de la elección.
- Selección de los patrones de diseño y fundamentación de la elección.
- Diseño de las vistas arquitectónicas.
- Representación del modelo propuesto.
- Implementación de un componente de gestión de usuarios para el tablero digital diseñado.
- Selección del método de validación y fundamentación de la elección.
- Aplicación del método de validación seleccionado.

Al finalizar la investigación arrojará algunos **resultados esperados**:

- Diseño de la arquitectura de un tablero digital totalmente integrable a sistemas externos.
- Integración de los componentes implementados a PATDSI, aportando un tablero digital completamente funcional.

La estructura del trabajo de diploma está concebida de la siguiente manera: resumen, introducción, tres capítulos, conclusiones, recomendaciones y referencias bibliográficas.

Capítulo 1: En el transcurso de este capítulo se realiza una investigación sobre la arquitectura de software de sistemas para la toma de decisiones, se expone su definición desde distintos puntos de vista, y se realiza un análisis de algunos de los conceptos más importantes relacionados con este tema. En otro momento se efectuará la definición de las herramientas, técnicas, lenguajes y metodologías que se utilizarán para dar solución a la problemática planteada.

Capítulo 2: En este capítulo se hace una propuesta de solución al problema planteado. Se realiza el diseño de las vistas arquitectónicas apoyado en las 4+1 vistas definidas por Philippe Kruchten y se relacionan los requisitos funcionales y no funcionales que tendrá la aplicación.

Capítulo 3: En este capítulo se realiza la evaluación del diseño propuesto a través del método seleccionado. Se identifican los riesgos y se toman las decisiones pertinentes para mitigarlos.

CAPÍTULO 1. Fundamentación teórica de la investigación.

Introducción.

En el transcurso de este capítulo se realiza una investigación sobre la arquitectura de software de sistemas para la toma de decisiones, exponiendo su definición desde distintos puntos de vista y sus tendencias actuales. Se realiza un análisis de algunos de los conceptos más importantes relacionados con este tema. En otro momento se efectuará la definición de las herramientas, técnicas, lenguajes y metodologías que se utilizarán para dar solución a la problemática planteada.

1.1 Definición de arquitectura de software.

Los primeros indicios de la existencia de un sistema para estructurar el desarrollo de un proyecto informático se remontan a 1968, cuando Edsger Dijkstra llevó a cabo la propuesta de establecer un orden de estructuración para la solución de cualquier problema informático, ya que las ciencias de la computación eran una rama aplicada de la matemática, para la cual, también había que seguir pasos lógicos. Dijkstra nunca usó el término arquitectura de software para definir su propuesta, aunque sí marcó un paradigma para el diseño del software. [2]

El término “arquitectura de software” fue usado por primera vez en 1992 por Perry y Wolf. Estos proponían que el software se tratara al igual que las construcciones en lo relacionado con la arquitectura. [3]

A través de los años, muchos han sido los puntos de vista sobre la verdadera definición de arquitectura de software, algunos de estos se citan a continuación:

La definición planteada por Bass, Clements y Kazman es:

“La arquitectura de software de un sistema de programa o computación es la estructura de las estructuras del sistema, la cual comprende los componentes del software, las propiedades de esos componentes visibles externamente, y las relaciones entre ellos.”[4]

Otra definición importante es la de Philippe Kruchten:

“La arquitectura de software, tiene que ver con el diseño y la implementación de estructuras de software de alto nivel. Es el resultado de ensamblar un cierto número de elementos arquitectónicos de forma adecuada para satisfacer la mayor funcionalidad y requerimientos de desempeño de un sistema, así como requerimientos no funcionales, como la confiabilidad, escalabilidad, portabilidad, y disponibilidad”.[5]

La definición que actualmente rige el término es la provista por la IEEE en su documento STD 1417-2000:

"La arquitectura del software es la organización fundamental de un sistema formada por sus componentes, las relaciones entre ellos y el contexto en el que se implantarán, y los principios que orientan su diseño y evolución". [6]

Aunque los conceptos mencionados no son aprobados por la totalidad de arquitectos sí se puede obtener de ellos una definición más relacionada con el problema de la investigación.

La arquitectura del software es el diseño más completo de la estructura de un sistema informático, se conforma de un grupo de elementos que se unen de manera específica para dar solución a una problemática determinada y obtener el mejor resultado posible. Representarla de forma eficiente facilita el trabajo y constituye un gran eslabón dentro del ciclo de vida de un proyecto informático.

1.2 Arquitectura de software de sistemas para la toma de decisiones.

La arquitectura de software para la toma de decisiones no tiene una amplia variación en cuanto a los elementos que la componen, la mayoría de los tableros digitales asemejan la forma en que está dispuesta su arquitectura.

Uno de los ejemplos de la estructura de la arquitectura la presenta el tablero digital desarrollado por Juan Ramírez Monreal para la plataforma Paula y se descompone en 4 niveles: [7]

- En el primer nivel, se encuentra todo lo relacionado con la interfaz gráfica del usuario, la vista de la aplicación. Son las páginas que le aparecen al cliente en su navegador.
- En un segundo nivel, se encuentran los servlets que realizan la misión de redirigir los datos y eventos producidos en la capa web, a las clases que deben tratar esos datos o eventos. Los servlets hacen la misión de controlador del sistema.
- En el tercer nivel, se tendrá el modelo de clases que conforma el sistema. Que no son más que las clases persistentes de la aplicación.
- En el cuarto nivel se tendrá el modelo de datos, este nivel representa las tablas contenidas en las bases de datos con las relaciones entre ellas.

1.3 Tendencias actuales en el diseño de sistemas para la toma de decisiones.

El tablero digital surge a causa de la carencia de una técnica para instruir a los dirigentes a establecer y conformar la información. Por ello cada diseño para desarrollar un nuevo tablero debe seguir algunos paradigmas que responden a las tendencias actuales en la implementación de los mismos.

Es importante mencionar como elemento principal que a todos los tableros se le seleccionan o se le da la posibilidad al usuario de especificar los indicadores que se van a analizar. Hoy en día, después

del esclarecimiento de los temas e indicadores claves a seguir, hay una tendencia a implementar la mayoría o todos estos elementos: [8]

- **Reporte o Pantalla:** Accesorio que muestra la información clave para el diagnóstico.
- **Período del Indicador:** Rango de tiempo en el que se hará el ejercicio.
- **Apertura:** Forma en la cual se podrá abrir y clasificar la información.
- **Frecuencia de actualización:** Tiempo que transcurre entre distintas actualizaciones de los datos.
- **Referencia:** Base sobre la cual se desean calcular las desviaciones. Puede ser un estándar.
- **Parámetro de alarma:** Niveles por encima o por debajo de los cuales el indicador es preocupante.
- **Gráfico:** La mejor forma de representar la realidad que nos muestra la información.
- **Responsable de monitoreo:** Es quien debe informar al nivel superior cuando haya en el indicador algo fuera de lo común.
- **Avisos automáticos:** Advertencias que emite el sistema para detectar problemas de acuerdo a parámetros incluidos por el usuario.

Los tableros digitales tienen una amplia gama de aplicaciones en la actualidad; herramientas que se ven a diario en cualquier lugar constituyen un tablero digital y por desconocimiento de la definición de estos no se tienen presente. Ejemplo se puede señalar a la medicina, la cual está apoyada de controles para el diagnóstico de la salud de los pacientes; otro ejemplo es la aeronáutica, cuyos indicadores sintetizan la información de la máquina y del ambiente en que se mueve para evitar contratiempos y permite a los conductores dirigirse a lugares seguros; el tablero de un sistema eléctrico, la pizarra de un automóvil o la de una represa constituyen otros ejemplos. En todos estos casos el tablero permite a través del color de las luces y alarmas ser el disparador para la toma de decisiones. En todos estos ejemplos es fundamental definir los indicadores a monitorear.

1.4 Metodología para el diseño de tableros digitales.

Comúnmente, los tableros digitales utilizan una metodología centrada en el usuario que integra datos de acuerdo con los problemas, funciones principales o procesos comerciales críticos de la empresa. Están diseñados frecuentemente para tratar un único problema de forma aislada y desarrollar desde simples informes en línea hasta una compleja representación visual de mediciones clave. Los tableros digitales pueden mostrar una gran cantidad de resúmenes y mediciones detallados. Estos datos se actualizan automáticamente en forma diaria, semanal, mensual o en tiempo real. [9]

La metodología para el diseño del tablero digital será una variación de la usada por Juan Ramírez Monreal en su tesis Desarrollo de un Dashboard en la Plataforma Paula. Expone que la fase de

construcción de un tablero digital debe estar marcada por 4 etapas fundamentales, con actividades específicas en cada etapa. Se brindará una descripción a continuación: [7]

- **Estudio Previo.**
  - Análisis de la información disponible sobre los cuadros de mando y todo lo relacionado con ellos.
  - Desarrollo de una estrategia para ir cumpliendo plazos dentro de lo posible, con un pequeño margen, tenido en cuenta también, a la hora de ir marcando los hitos.
  - Estudio y comprensión de todas las herramientas necesarias para la realización del proyecto.
  - Establecimiento de puntos críticos.
  - Investigación de dominios disponibles.
- **Diseño de las interfaces y prototipos.**
  - Desarrollo del contenido visual de la aplicación.
  - Desarrollo del prototipo de un indicador.
  - Desarrollo de un prototipo de páginas con varios indicadores.
  - Aprobación del tutor en la empresa del trabajo realizado.
- **Creatividad aplicada al sitio y elaboración del anteproyecto.**
  - Creación de gráficos con la librería seleccionada.
  - Desarrollo de la navegabilidad del sitio
  - Creación del modelo de datos de la plataforma.
  - Definición de los formularios necesarios para interactividad.
  - Pruebas Unitarias.
  - Aprobación del tutor para pasar al proyecto final.
- **Desarrollo del proyecto final.**
  - Introducción de las mejoras solicitadas por el tutor.
  - Creación de documento que recogerá toda la información sobre las herramientas y trabajo realizado.
  - Verificación de completitud y exactitud.
  - Verificación en diferentes tipos de navegadores para asegurar compatibilidad.
  - Aprobación del tutor del trabajo final realizado.

1.5 Patrones

Los patrones constituyen una especie de paradigma que describe la forma de estructurar e implementar un sistema informático para dar la solución más óptima a una problemática determinada. Cada patrón es usado para suplir determinados requerimientos y a su vez cada requerimiento puede necesitar de la utilización de varios patrones para ser cumplido.

Los patrones para el desarrollo de software constituyen un avance en la Tecnología Orientada a Objetos. Los mismos participan en la intención de capturar, recopilar, y transmitir la experiencia del diseñador. [10]

Una forma eficiente de evaluar un patrón es a través de algunas de las características que estos deben tener [10]:

- Debe solucionar un problema, no solo plantear estrategias abstractas.
- Debe ser una solución demostrada, no teoría o especulación.
- Describe participantes y relaciones entre ellos, no solo módulos.

Los elementos fundamentales por los cuales están compuestos los patrones, son [11]:

- **Nombre:** Deben presentar un nombre para hacer referencia a estos y no a su descripción. Es recomendable poner los nombres sugerentes a la función que realiza el patrón.
- **Problema:** Describe el problema que debe presentarse para su utilización.
- **Solución:** Describe los elementos por los cuales está compuesto: clases, objetos, relaciones, responsabilidades y colaboraciones.
- **Consecuencias:** Describe los costos y beneficios que reportara su utilización. Incluye el impacto sobre la flexibilidad, extensibilidad y portabilidad del sistema.

Los patrones se dividen y clasifican en cinco grupos principales [10]:

- **Patrones de arquitectura:** Relacionados a la interacción de objetos entre niveles arquitectónicos, problemas arquitectónicos o adaptabilidad a requerimientos cambiantes.
- **Patrones de diseño:** Fueron construidos dado los problemas con la claridad de diseño, multiplicación de clases y adaptabilidad a requerimientos cambiantes.
- **Patrones de análisis:** Usualmente específicos de aplicación.
- **Patrones de proceso o de organización:** Tratan todo lo concerniente con el desarrollo o procesos de administración de proyectos, técnicas, o estructuras de organización.
- **Patrones de idioma:** Regulan la nomenclatura en la cual se escriben, se diseñan y desarrollan los sistemas.

1.5.1 Estilos y patrones arquitectónicos.

Muy a menudo se solapan los temas relacionados con estilos y patrones arquitectónicos, ya que ambos hacen referencia a formas de estructurar un sistema y relacionar sus componentes. Pero su diferencia radica en el nivel de abstracción con la que ejercen. Los estilos están en un primer plano,

definen la arquitectura de forma más abstracta que los patrones, son los que dictan las pautas principales en la organización y estructura de un sistema. [12]

Por el nivel de abstracción con el trabajan y la generalidad de los conceptos, los estilos por sí solos no definen la estructura arquitectónica de un proyecto, es por ello que se ven definidos un conjunto de “subestilos” dentro de cada uno. Estos “subestilos” se encargan de definir a un nivel más específico la forma abstracta con que los estilos representan la estructura organizativa de los sistemas, son los conocidos patrones arquitectónicos. Cada uno de estos patrones va enfocado hacia un modelo arquitectónico determinado, manteniendo las pautas del estilo al cual pertenece.

Los estilos se ordenan de acuerdo a su concepción, a esta forma organizativa de dividirlos se le denomina clases de estilo. A continuación se presenta la descripción de la clase de estilo propuesta a usar en la investigación:

- **Estilos de llamada y retorno:** Son los estilos más generalizados en proyectos de gran escala. Consiste en la implementación de un programa principal que controla el sistema y varios subprogramas que se comunican con éste mediante el uso de llamadas [13]. Es el propuesto a usar en la investigación debido a que le permite al diseñador obtener una estructura de programa fácil de modificar y se basan en la bien conocida abstracción de procedimientos/ funciones/ métodos.

  - **Modelo Vista Controlador (MVC):** Separa el modelado del dominio, la presentación y las acciones basadas en datos ingresados por el usuario en tres clases diferentes: Modelo, Vista y Controlador. [13]

    Ventajas: Soporte de vistas múltiples. Dado que la vista se halla separada del modelo y no hay dependencia directa del modelo con respecto a vista, la interfaz de usuario puede mostrar múltiples vistas de los mismos datos simultáneamente. Adaptación al cambio.

    Desventajas: Los cambios en las capas de bajo nivel tienden a filtrarse hacia las de alto nivel.
  - **Arquitecturas basadas en componentes:** Los componentes son las unidades de modelado, diseño e implementación, estos se implementan con la intención de poder ser accesibles hasta cierto punto, de modo que su funcionalidad y propiedades puedan ser descubiertas y utilizadas en tiempo de ejecución.

    Ventajas: Permite alcanzar un mayor nivel de reutilización de software. Permite que las pruebas sean ejecutadas probando cada uno de los componentes antes de probar el conjunto completo de componentes ensamblados. Simplifica el mantenimiento del sistema, cuando existe un débil acoplamiento entre componentes, el desarrollador es libre de actualizar y/o agregar componentes según sea necesario, sin afectar otras partes del sistema. Dado que un componente puede ser construido y luego mejorado continuamente por un experto u

organización, la calidad de una aplicación basada en componentes mejorará con el paso del tiempo.

Desventajas: Las actualizaciones de los componentes adquiridos no están en manos de los desarrolladores del sistema. Si no existen los componentes necesarios, hay que desarrollarlos y se puede perder mucho tiempo. Los componentes que se adquieren pueden tener conflictos si las versiones de estos no están estandarizadas con la aplicación donde serán acoplados.

1.5.2 Patrones GoF.

Estos son los Patrones de Diseño en el campo del Diseño Orientado a Objetos más conocidos y usados en la actualidad. Fue por los años 1994, que apareció el libro “Design Patterns: Elements of Reusable Object Oriented Sofware” escrito por los ahora famosos Gang of Four (GoF, que en español es la pandilla de los cuatro) formada por Erich Gamma, Richard Helm, Ralph Johnson y John Vlissides. Ellos recopilaron y documentaron 23 patrones de diseño aplicados usualmente por expertos diseñadores de software orientado a objetos. Desde luego que ellos no son los inventores ni los únicos involucrados, pero no fue hasta luego de la publicación de ese libro que empezó a difundirse con más fuerza la idea de patrones de diseño. La totalidad de los patrones definidos por “la pandilla de los cuatro” se concentran en tres grupos: de creación, de estructura y de comportamiento. [11]

A continuación se listan los patrones GoF que serán usados en la implementación de la aplicación con una breve descripción del propósito de cada uno de ellos [13]:

- **Patrones de creación.**
  - ***Fábrica Abstracta (Abstract Factory)*:** Proporciona una interfaz para crear familias de objetos o que dependen entre sí, sin especificar sus clases concretas.
  - ***Instancia Única (Singleton)*:** Garantiza que una clase sólo tenga una instancia, y proporciona un punto de acceso global a ella.

- **Patrones estructurales**
  - **Objeto Compuesto (Composite):** Combina objetos en estructuras de árbol para representar jerarquías. Permite que los clientes traten de manera uniforme a los objetos individuales y a los compuestos.
  - **Envoltorio (Decorator):** Añade dinámicamente nuevas responsabilidades a un objeto, proporcionando una alternativa flexible a la herencia para extender la funcionalidad.
  - **Fachada (Facade):** Provee de una interfaz unificada simple para acceder a una interfaz o grupo de interfaces de un subsistema.

- **Patrones de comportamiento.**
  - ***Orden (Command):*** Encapsula una petición en un objeto, permitiendo así parametrizar a los clientes con distintas peticiones, encolar o llevar un registro de las peticiones y poder deshacer las operaciones.

1.5.3 Patrones GRASP.

Los patrones GRASP describen los principios fundamentales de diseño de objetos para la asignación de responsabilidades. En cuanto a las responsabilidades UML define una responsabilidad como "un contrato u obligación de un clasificador". Las responsabilidades están relacionadas con las obligaciones de un objeto en cuanto a su comportamiento. [14]

Básicamente, estas responsabilidades son de los siguientes dos tipos: [14]

Conocer.

- Conocer los datos privados encapsulados.
- Conocer los objetos relacionados.
- Conocer las cosas que puede derivar o calcular.

Hacer.

- Hacer algo él mismo, como crear un objeto o hacer un cálculo.
- Iniciar una acción en otros objetos.
- Controlar y coordinar actividades en otros objetos.

GRASP define en total nueve patrones, aunque de estos, los que se usarán son: [14]

Experto.

Problema: ¿Cómo asignar responsabilidades, de la forma más eficiente?

Solución: Asignar una responsabilidad al experto en información: la clase que cuenta con la información necesaria para cumplir la responsabilidad.

Explicación: Experto es un patrón que se usa más que cualquier otro al asignar responsabilidades; es un principio básico que suele utilizarse en el diseño orientado a objetos. Con él no se pretende designar una idea oscura ni extraña; expresa simplemente la "intuición" de que los objetos hacen cosas relacionadas con la información que poseen.

Ventajas:

- Se conserva el encapsulamiento, ya que los objetos se valen de su propia información para hacer lo que se les pide. Esto soporta un bajo acoplamiento, lo que favorece el hecho de tener sistemas más robustos y de fácil mantenimiento.
- El comportamiento se distribuye entre las clases que cuentan con la información requerida, alentando con ello definiciones de clases “sencillas” y más cohesivas que son fáciles de comprender y mantener. Así se brinda una alta cohesión.

Creador

Problema: ¿Quién debería ser el responsable de crear una nueva instancia de alguna clase?

Solución: La responsabilidades de crear una instancia de la clase A se le dará a aquella clase B, en los siguientes casos:

- B agrega los objetos A.
- B contiene los objetos A.
- B registra las instancias de los objetos A.
- B utiliza específicamente los objetos A.
- B tiene los datos de inicialización que serán transmitidos hacia A cuando este objeto sea creado (B es experto en la creación de A).

Explicación: El patrón Creador guía la asignación de responsabilidades relacionadas con la creación de objetos, tarea muy frecuente en los sistemas orientados a objetos. El propósito fundamental de este patrón es encontrar un creador que debemos conectar con el objeto producido en cualquier evento.

Ventajas:

- Se brinda un soporte al bajo acoplamiento, lo cual supone menos dependencias respecto al mantenimiento y mejores oportunidades de reutilización. Es probable que el acoplamiento no aumente, pues la clase creada tiende a ser visible a la clase creador, debido a las asociaciones actuales que llevaron a elegirla como el parámetro adecuado.

Bajo Acoplamiento

Problema: ¿Cómo dar soporte a una dependencia escasa y a un aumento de la reutilización?

Solución: Asignar una responsabilidad para mantener bajo acoplamiento.

Explicación: El bajo acoplamiento estimula la asignación de responsabilidades de forma tal que la inclusión de éstas no incremente el acoplamiento, creando clases más independientes y con mayor resistencia al impacto de los cambios, que aumentan la productividad y la posibilidad de reutilización.

Ventajas:

- No se afectan por cambios en otros componentes.

- Fáciles de entender por separado.
- Fáciles de reutilizar.

Alta Cohesión

Problema: ¿Cómo mantener la complejidad dentro de los límites manejables?

Solución: Asignar una responsabilidad de modo que la cohesión siga siendo alta. La cohesión es una medida de cuan relacionadas y enfocadas están las responsabilidades de una clase, además de que una alta cohesión garantiza que clases con responsabilidades estrechamente relacionadas no realicen un trabajo enorme.

Explicación: Como el patrón Bajo Acoplamiento, también Alta Cohesión es un principio que debemos tener presente en todas las decisiones de diseño: es la meta principal que ha de buscarse en todo momento. Es un patrón evaluativo que el desarrollador aplica al valorar sus decisiones de diseño.

Ventajas:

- Mejoran la claridad y la facilidad con que se entiende el diseño.
- Se simplifican el mantenimiento y las mejoras en funcionalidad.
- A menudo se genera un bajo acoplamiento.
- Las clases soportan una mayor capacidad de reutilización, porque una clase muy cohesiva puede destinarse a un propósito muy específico.

Controlador

Problema*:* ¿Quién debería encargarse de atender un evento del sistema?

Solución: Asignar la responsabilidad del manejo de un mensaje de los eventos del sistema a una clase que represente una de las siguientes opciones: Controlador de fachada, controlador de tareas o controlador de casos de uso.

Explicación: La mayor parte de los sistemas reciben eventos de entrada externa, los cuales generalmente incluyen una interfaz gráfica para el usuario operado por una persona. En todos los casos, si se recurre a un diseño orientado a objetos, hay que elegir los controladores que manejen esos eventos de entrada. Este patrón ofrece una guía para tomar decisiones apropiadas que generalmente se aceptan.

Ventajas:

- Mayor potencial de los componentes reutilizables. Garantiza que la empresa o los procesos de dominio sean manejados por la capa de los objetos de dominio y no por la de la interfaz.
- Reflexionar sobre el estado del caso de uso.

1.6 Herramientas y Lenguajes.

El tablero digital que se implementará deberá ser un activo de software que sea capaz de integrarse a otras soluciones informáticas, en un inicio deberá ser totalmente compatible con PATDSI. Para facilitar el proceso de integración se deben lograr algunas pautas regidas por la arquitectura base del proyecto. Debido a esto se realizó un análisis sobre las herramientas y lenguajes que se usarán para la implementación de la aplicación y se arribó a la conclusión de continuar usando las existentes.

1.6.1 Lenguaje Unificado de Modelado. (UML)

UML (Unified Modeling Language) es un lenguaje que permite modelar, construir y documentar los elementos que forman un software construido orientado a objetos. Se ha convertido en el estándar de facto de la industria. Fue concebido por Grady Booch, Ivar Jacobson y Jim Rumbaugh. Estos autores fueron contratados por la empresa Rational Software Company para crear una notación unificada en la que basar la construcción de sus herramientas CASE. En el proceso de creación de UML han participado, no obstante, otras empresas de gran peso en la industria como Microsoft, Oracle o IBM, así como grupos de analistas y desarrolladores. [15]

1.6.2 Metodología OpenUP.

OpenUP es un marco del proceso del desarrollo del software, es un proceso iterativo para el desarrollo de software que tiene las características de ser: [16]

- **Mínimo**: Solo incluye el contenido del proceso fundamental
- **Completo**: Puede ser manifestado como proceso entero para construir un sistema.
- **Extensible**: Puede ser utilizado como base para agregar o para adaptar más procesos.

El uso de OpenUP proyecta beneficios como: [16]

-Ya que es apropiado para proyectos pequeños y de bajos recursos permite disminuir las probabilidades de fracaso en los proyectos pequeños e incrementar las probabilidades de éxito.

- Permite detectar errores tempranos a través de un ciclo iterativo.
- Evita la elaboración de documentación, diagramas e iteraciones innecesarios requeridos en la metodología RUP.
- Por ser una metodología ágil tiene un enfoque centrado al cliente y con iteraciones cortas.

OpenUp se divide en 4 fases: [16]

Concepción: Primera de las fases del ciclo de vida del proyecto, trata acerca del entendimiento del propósito y objetivos del sistema. Se obtiene suficiente información para confirmar qué el proyecto debe hacer. El objetivo de ésta fase es capturar las necesidades de los stakeholder.

Elaboración: Es la segunda de las 4 fases del ciclo de vida del OpenUP, donde se trata los riesgos significativos para la arquitectura. El propósito de esta fase es establecer la base la elaboración de la arquitectura del sistema.

Construcción: Esta fase está enfocada al diseño, implementación y prueba de las funcionalidades para desarrollar un sistema completo. El propósito de esta fase es completar el desarrollo del sistema basado en la arquitectura definida.

Transición: Es la última fase, cuyo propósito es asegurar que el sistema es entregado a los usuarios, y evalúa la funcionalidad y rendimiento del último entregable de la fase de construcción.

1.6.3 Visual Paradigm 3.4 sp1.

Visual Paradigm es una herramienta UML fácil de usar que soporta la ingeniería inversa, generación de código, importación desde Rational Rose(Otra herramienta CASE muy poderosa), exportación/importación XML, generador de informes, editor de figuras, integración con Microsoft Visio, plug-in, integración IDE con Visual Studio, Eclipse, NetBeans, generación de script SQL para múltiples gestores y otros. Entre sus características se incluyen el modelado colaborativo con CVS y Subversion e interoperabilidad con modelos UML2 a través de XMLI.

Usando Visual Paradigm se tendrá:

- Un diseño centrado en casos de uso y enfocado al negocio que generan un software de mayor calidad.
- Uso de un lenguaje estándar común a todo el equipo de desarrollo que facilita la comunicación.
- Un modelo y código que permanece sincronizado en todo el ciclo de desarrollo.
- Disponibilidad de múltiples versiones, para cada necesidad.
- Disponibilidad de integrarse en los principales IDEs de desarrollo.
- Posibilidad de ejecución en múltiples plataformas.

1.6.4 PHP 5.3.

PHP es un lenguaje de programación interpretado que significa Hypertext Pre-processor, diseñado específicamente para la creación de páginas web dinámicas, embebido en páginas HTML y ejecutado en el servidor. La mayor parte de su sintaxis ha sido tomada de C, Java y Perl con algunas características específicas de sí mismo. La meta del lenguaje es permitir rápidamente a los desarrolladores la generación dinámica de páginas. No es un lenguaje de marcas como podría ser HTML, XML o WML. Está más cercano a JavaScript o a C. Las principales características de PHP son: su rapidez; su facilidad de aprendizaje; su soporte multiplataforma tanto de diversos Sistemas Operativos, como servidores HTTP y de bases de datos; y el hecho de que se distribuye de forma gratuita bajo una licencia abierta. [17]

1.6.5 Symfony 1.4.6.

Symfony es uno de los frameworks PHP más populares y usados por los usuarios y las empresas dentro del mundo del software. Es muy estable y posee muy buena documentación. Separa la lógica

de negocio, la lógica de servidor y la presentación de la aplicación web. Proporciona varias herramientas y clases encaminadas a reducir el tiempo de desarrollo de una aplicación web compleja. Además, automatiza las tareas más comunes, permitiendo al desarrollador dedicarse por completo a los aspectos específicos de cada aplicación. El resultado de todas estas ventajas es que no se debe reinventar la rueda cada vez que se crea una nueva aplicación web. [18]

1.6.6 ExtJS 3.3.

ExtJS es un framework de presentación de JavaScript completamente orientado a objetos, es multiplataforma y hace uso de la tecnología AJAX. Este framework brinda múltiples posibilidades para el trabajo con las validaciones y manejo de errores en el cliente. Además permite la personalización de temas de estilos y provee el trabajo con una amplia configuración e intenso trabajo con las hojas de estilo CSS. Basa toda su funcionalidad en JavaScript a través de librerías YUI, jQuery, o haciendo uso de la librería nativa, así en tiempo de ejecución carga y crea todos los objetos HTML a través del uso intenso de Modelo de Objetos del Documento (DOM). Cuenta con dos licencias, una comercial y otra Open Source. [19]

1.6.7 Propel.

Propel es un ORM (object/relational mapping) para PHP que facilita la labor de desarrollo de aplicaciones web, gracias a la capa que transforma el tratamiento de la BD mediante objetos, con la que se puede recuperar, insertar y modificar datos. No es necesario preocuparse por las conexiones de la BD y escribir SQL. Tampoco es necesario escapar datos o realizar casting. Tan solo es necesario definir la base de datos en formato XML u obtener la definición desde una base de datos ya existente. [20]

1.6.8 NetBeans 6.9.

Este IDE es una herramienta para escribir, compilar, depurar y ejecutar programas. Está escrito en Java pero puede servir para cualquier otro lenguaje de programación. Existen además un número importante de módulos para extenderlo, es un producto libre y gratuito sin restricciones de uso. NetBeans permite crear aplicaciones Web con PHP 5, posee un potente debugger integrado y además viene con soporte para Symfony.

1.6.9 PostgreSQL 8.4.

PostgreSQL es un Gestor de Base de Datos muy competente y eficaz, que a diferencia de MySQL, soporta tanto bases de datos relacionales como orientadas a objetos. Es de código abierto, por lo que sus potencialidades están en constante perfeccionamiento, característica que lo sitúa por delante de otros productos competitivos como Oracle o Microsoft SQL Server 2000, ambos; software propietarios. Funciona en todos los sistemas operativos importantes. Es totalmente compatible con la

introducción de llaves foráneas, consultas usando joins, vistas, disparadores y procedimientos almacenados en distintos idiomas. Incluye la mayoría de los tipos de datos.

1.6.10 Apache 2.

Apache comenzó en 1995 como un proyecto para la creación de un servidor Web de código abierto. A través de los años se han ido añadiendo otras funcionalidades importantes como la habilidad de que un solo servidor pueda mantener múltiples sitios Web virtuales y esquemas de autenticación. Actualmente es uno de los servidores más usados para la creación de sitios Web comerciales. Está disponible para una gran cantidad de plataformas.

1.7 Calidad en la arquitectura de software.

La evaluación de la arquitectura es el análisis que se le realiza a cada uno de los requerimientos de calidad con el fin de minimizar los riesgos de construir un sistema que falle y consecuentemente disminuirá el costo de desarrollo del sistema. De esta manera, el interés se centra en soportar la arquitectura en este proceso. Hasta hace poco no existían métodos de utilidad general para evaluar arquitecturas de software. Si alguno existía, sus enfoques eran incompletos y no repetibles, por lo que resultaban poco confiables [21].

En virtud de esto, se han propuesto múltiples métodos de evaluación, entre los cuales se destacan: SAAM (Software Analysis Architecture Method), ATAM (Architecture Trade-off Analysis Method), ARID (Active Reviews for Intermediate Designs).

Método de Análisis de Arquitecturas de Software (SAAM).

SAAM está basado en escenarios, su objetivo principal es el atributo modificabilidad. Permite evaluar una arquitectura o evaluar y comparar varias. Dentro de sus ventajas se encuentra que el esfuerzo y el costo de los cambios pueden ser estimados con anticipación. Su principal debilidad es que no provee una métrica clara sobre la calidad de la arquitectura evaluada [21].

Método de Análisis de Acuerdos de Arquitectura (ATAM)

ATAM (Architecture Trade-off Analysis Method): está inspirado en tres áreas distintas: los estilos arquitectónicos, el análisis de atributos de calidad y el método de evaluación SAAM (Software Architecture Analysis Method). El nombre del método ATAM surge del hecho de que revela la forma en que una arquitectura específica satisface ciertos atributos de calidad, y provee una visión de cómo los atributos de calidad interactúan con otros. El método se concentra en la identificación de los estilos arquitectónicos o enfoques arquitectónicos utilizados. Estos elementos representan los medios empleados por la arquitectura para alcanzar los atributos de calidad, así como también permiten

describir la forma en la que el sistema puede crecer, responder a cambios, e integrarse con otros sistemas.[21]

Método de evaluación para Arquitecturas Parciales (ARID)

ARID (Active Reviews for Intermediate Design), es un método de bajo costo y gran beneficio, el mismo es conveniente para realizar la evaluación de diseños parciales en las etapas tempranas del desarrollo. Se basa en ensamblar el diseño de los clientes para articular los escenarios de usos importantes o significativos, y probar el diseño para ver si satisface los escenarios. Como resultado de la aplicación de dicho procedimiento se obtiene un diseño de alta fidelidad acompañado de una alta familiarización con el diseño de los stakeholders. Este método consta de 9 pasos agrupados en dos fases (Actividades Previas y Evaluación) [21].

El método SAAM se sugiere cuando el atributo de calidad modificabilidad es el de mayor interés. Mientras que el método ATAM evalúa con más profundidad, en relación con otros métodos, cuestiones referentes a la arquitectura, como son los atributos de calidad. Por otro lado el ARID evalúa mejor la factibilidad de la arquitectura. Debido a que se pretende una evaluación profunda de la arquitectura propuesta teniendo en cuenta la incidencia de distintos atributos de calidad, así como la determinación del impacto de las decisiones arquitectónicas tomadas, se utilizará el método ATAM para la evaluación de la misma.

Conclusiones parciales.

En el presenta capítulo se ha realizado una investigación sobre el objeto de estudio, enfocado principalmente a la rama de la toma de decisiones. Además se definieron las técnicas, herramientas y metodologías que serán usadas en el desarrollo del tablero digital. Se seleccionó el estilo arquitectónico de llamada y retorno con la arquitectura modelo-vista-controlador. Como lenguaje de modelación se seleccionó UML. La metodología a usar será OpenUP y la herramienta CASE será Visual Paradigm 3.4 sp1. PHP 5 es el lenguaje de programación idóneo para la implementación, con Symfony 1.4.6 como framework de desarrollo y propel como ORM. La parte visual de la aplicación la manejará ExtJS 3.3. El IDE seleccionado fue NetBeans 6.9. El gestor de bases de datos recomendado es PostgreSQL 8.4 y el servidor web Apache 2. Además de definió ATAM como método para la evaluación de la arquitectura.

CAPÍTULO 2: Descripción Arquitectónica.

Introducción.

En el transcurso de este capítulo se realiza una propuesta de solución al problema de la investigación planteado en el diseño teórico, se lleva a cabo un análisis de los requerimientos y las restricciones arquitectónicas de la aplicación y se muestra el principio del funcionamiento del sistema a través de la representación de las vistas arquitectónicas.

2.1 Estructura del sistema

La aplicación en cuestión se encuentra dividida en 4 módulos. El módulo Diseñador de Modelos, que se encarga de la creación de los modelos a partir de las base de datos. El módulo Diseñador de Tablero Digital, que tiene la función de crear tableros digitales para que sean guardados y posteriormente cargados por el módulo Visor de Tablero Digital. Además se cuenta con un módulo para la administración de los tableros digitales.

Figura 1. Estructura del sistema.

2.2 Requisitos funcionales.

Los requisitos funcionales (RF) son capacidades o condiciones que el sistema debe cumplir. Son las funcionalidades que dan sentido de vida al sistema; son el motivo por el que fue creado.

2.2.1 RF del módulo Diseñador de Tablero Digital.

RF1 Cargar modelos.

RF2 Buscar modelo.

RF3 Adicionar categoría.

RF4 Modificar categoría.

RF5 Eliminar categoría.

RF6 Listar categoría.

RF7 Buscar categoría.

RF8 Adicionar tablero digital.

RF9 Modificar tablero digital.

RF10 Eliminar tablero digital.

RF11 Buscar tablero digital.

RF12 Cambiar de categoría un tablero digital.

RF13 Guardar tablero digital.

RF14 Insertar panel.

RF15 Seleccionar fuente de datos.

RF16 Transformar datos de panel.

RF17 Exportar datos de panel.

RF18 Convertir panel en panel de pestañas.

RF19 Adicionar pestaña.

RF20 Renombrar pestaña.

RF21 Eliminar pestaña.

RF22 Convertir panel de pestaña en panel.

RF23 Eliminar tabla con fuente de datos.

RF24 Visualizar tablero digital.

2.2.2 RF del módulo Diseñador de Modelos.

RF1Visualizar orígenes de datos.

RF2 Seleccionar un origen de datos.

RF3 Mostrar objetos y entidades del origen de datos seleccionado.

RF4 Seleccionar objetos y entidades desde el origen de datos seleccionado.

RF5 Modificar orígenes de datos.

RF6 Adicionar origen de datos.

RF7 Eliminar origen de datos.

RF8 Crear modelos semánticos.

RF9 Buscar los modelos semánticos.

RF10 Modificar modelos semánticos.

RF11 Eliminar modelos semánticos.

RF12 Obtener entidades relacionadas.

RF13 Renombrar un modelo semántico.

RF14 Probar conexión a un origen de datos.

RF15 Visualizar modelos semánticos.

RF16 Renombrar las entidades del origen de datos seleccionado y los atributos de esas entidades.

RF17 Mostrar objetos y entidades del modelo semántico seleccionado.

RF18 Seleccionar entidades del modelo semántico seleccionado.

RF19 Mostrar el origen de datos desde el cual fue diseñado el modelo semántico.

2.2.3 RF del módulo Visor de Tablero Digital.

RF1 Buscar tablero digital.

RF2 Visualizar tablero digital.

2.2.4 RF del módulo Administrador de Tablero Digital.

RF1 Adicionar usuario.

RF2 Modificar usuario.

RF3 Eliminar usuario.

RF4 Buscar usuario.

RF5 Eliminar tablero digital.

2.3 Requisitos no funcionales.

Los requisitos no funcionales (RNF) son las propiedades o cualidades que el producto debe tener y que no se encuentran dentro de las funcionalidades imprescindibles para cumplir con los requerimientos del cliente. Son las propiedades que hacen que un software sea confiable, seguro, rápido o agradable.

2.3.1 RNF de Usabilidad.

RNF 1: Tipo de Aplicación Informática.

El software es una herramienta WEB, pero con características muy similares a las aplicaciones de escritorio en cuanto al diseño de las interfaces visuales y los tiempos de respuesta de la interacción del usuario con el sistema.

RNF 2 Finalidad.

Diseñar tableros digitales a partir de modelos estructurados de manera estándar que posibiliten la toma de decisiones mediante indicadores claves de desempeño.

RNF 3 Búsqueda rápida y sencilla de un tablero digital.

El usuario debe poder localizar un tablero digital de manera rápida, si es posible tenerlo ubicado en alguna categoría que permita dicha ubicación y tener la posibilidad de introducir criterios de búsqueda.

RNF 4 Permitir personalizar los tableros digitales

El usuario podrá al diseñar un tablero digital ajustarlo lo más posible a sus necesidades, estructurar los paneles y permitir el trabajo con gráficos.

RNF 5 Portabilidad.

El sistema debe ser visible desde cualquiera de los 2 sistemas operativos más usados (Linux y Windows). Debe tener el 100% de compatibilidad con el navegador Mozilla Firefox 4.0 o superior, además debe ser funcional para otros navegadores como Chrome 13 e Internet Explorer 8 o versiones superiores a estas. El servidor web y servidor de base de datos podrán funcionar sobre Windows o Linux, aunque este último es el recomendado.

RNF 6 Facilidad de Uso.

La aplicación tiene que ser capaz de ofrecer facilidades de uso para un buen entendimiento y aceptación del producto por los usuarios finales.

2.3.2 RNF de Confiabilidad.

RNF 7 Disponibilidad.

El sistema debe poder permanecer constantemente funcionando excepto cuando sea necesario reiniciarlo o detenerlo por tareas de mantenimiento o cambio de configuración.

RNF 8 No deben ocurrir errores críticos.

En la aplicación debe asegurarse el trabajo con las excepciones y las validaciones para que no ocurran errores críticos. Entre los errores críticos están: pérdida total de la información, o inhabilitación para el uso de ciertas partes del funcionamiento del sistema.

RNF 9 Disponibilidad de Servicios.

Debido a que el sistema mantendrá en todo momento una cola de solicitudes, es necesario que este permanezca en línea constantemente para atenderlas. El sistema permanecerá fuera de servicio sólo en caso que se estén realizando tareas de mantenimiento o de configuración.

El sistema deberá estar disponible las 24 horas del día, pudiendo estar fuera de servicio por un período de 72 horas máximo. La precisión y exactitud de las salidas del sistema se corresponden con la calidad y exactitud de la información contenida en las base de datos desde donde se extraigan los reportes. El sistema no es responsable por la falta de veracidad de dicha información. Además existen algunas situaciones externas que dificultan la disponibilidad de servicios y de las cuales el sistema no es responsable:

- Ausencia de conectividad con servidores de bases de datos: Que salgan de funcionamiento las bases de datos desde donde se extraen los reportes o que no exista conectividad hacia ellas.
- Fallo en el entorno de despliegue: Que falle el servidor donde se despliegue la solución.

2.3.3 RNF de Eficiencia.

La eficiencia del sistema depende en gran medida de la velocidad de conexión a las base de datos donde se encuentre, así como del volumen de información contenido en las mismas.

RNF 10 Tiempo mínimo de respuesta para la obtención de un tablero digital.

El tiempo mínimo para la obtención de un tablero digital no debe sobrepasar los 2 segundos en una red sin sobrecarga en el intercambio de datos.

RNF 11 Cantidad de usuarios conectados simultáneamente.

El sistema deberá permitir que existan 100 usuarios conectados de forma simultánea, ya que esta es la cifra de conexiones múltiples que permite el gestor de bases de datos.

2.3.4 RNF de Soporte.

RNF 12 Aceptabilidad de mejoras.

El sistema debe tener la posibilidad de ser mejorado en un futuro y aceptar nuevas funcionalidades sin necesidad de implementar una nueva aplicación.

2.3.5 RNF de Seguridad.

RNF 13 Seguridad.

El sistema debe restringir el acceso a los datos para las personas que no estén autorizadas a acceder a los distintos módulos de la aplicación.

2.3.6 RNF de Software.

RNF 14 Navegador Web.

Para tener acceso al sistema solo se deberá tener instalado un navegador web. Se recomienda Mozilla Firefox 4.0 o superior y no se necesita tener algún plugin o extensión extra instalados.

2.3.7 RNF de Hardware.

RNF 15 Prestaciones de PC Cliente.

Las PC que sean clientes de la aplicación deben tener las siguientes prestaciones mínimas:

Microprocesador: 500 MHz

Memoria RAM: 128 MB

Tarjeta de Red.

RNF 16 Prestaciones de Servidor Web.

El servidor web de la aplicación deben tener las siguientes prestaciones mínimas:

Microprocesador: 3.00 GHz

Memoria RAM: 1 GB

Capacidad de Disco Duro: 10 GB de espacio libre en el disco duro donde se encuentra almacenada la aplicación.

Tarjeta de Red.

RNF 17 Prestaciones de Servidor de Bases de Datos.

El servidor de bases de datos deben tener las siguientes prestaciones mínimas:

Microprocesador: 3.00 GHz

Memoria RAM: 1 GB

Capacidad de Disco Duro: 30 GB de espacio libre en el disco duro donde se encuentra almacenada la base de datos.

Tarjeta de Red.

2.4 Vistas Arquitectónicas.

La arquitectura de software se utiliza para tener una idea organizada y centralizada acerca de la estructura y funcionalidad del sistema. Los involucrados con la aplicación no necesariamente deben presentar roles comunes, es por ello que dentro de la arquitectura no se puede representar toda la funcionalidad del sistema con solo diseñar una única representación. Para la representación de la arquitectura propuesta se usarán las 4+1 vistas descritas por Philippe Kruchten. [5]

2.4.1 Vista de Casos de Uso.

La confección del diagrama de casos de usos del sistema se realiza con el fin de facilitar la comprensión de los requisitos funcionales y tener una idea clara de las acciones que el sistema debe llevar a cabo y el encargado de realizarlas. Está representado por los actores que son los encargados de interactuar con el sistema y los casos de uso que son las acciones que deben realizar estos actores. Para el caso de la representación de los diagramas no se tuvieron en cuenta todos los casos de uso, se realizó una selección de los arquitectónicamente significativos.

En la figura 2 se muestra el diagrama de casos de uso para el módulo Administrador de Tablero Digital. El usuario encargado de realizar las funcionalidades de este módulo es el administrador del sistema. A continuación se muestra la descripción de los casos de uso:

CU **Gestionar usuarios**: Permite al administrador tener control de los usuarios que acceden al sistema y los permisos que requiere cada uno de estos.

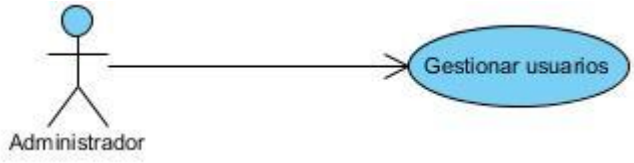

Figura 2. Diagrama de Casos de Uso del Sistema de Administrador de Tablero Digital.

En la Figura 3 se muestra el Diagrama de Casos de Uso del módulo Diseñador de Tablero Digital. El sistema cuenta con un solo actor, que es el que se encarga de llevar a cabo todas las acciones relacionadas con el diseño del tablero digital. El diagrama cuenta con dos paquetes, el Diseñar tablero digital que contiene los casos de uso relacionados con el diseño del tablero digital y el paquete complementarios que contiene casos de uso que constituyen herramientas necesarias para la creación de los tableros digitales, pero que no intervienen directamente en el diseño de estos.

A continuación el proceso de descripción de los casos de uso (CU):

CU **Guardar tablero digital**: Permite al especialista guardar un tablero digital diseñado.

CU **Transformar tabla en gráfico**: Permite que el especialista obtenga un gráfico a partir de la tabla que forma la relación de los datos y los indicadores.

CU **Exportar datos de panel**: Permite al especialista exportar el contenido que contiene el panel. Si el panel contiene un gráfico, el especialista tendrá la oportunidad de exportarlo como imagen.

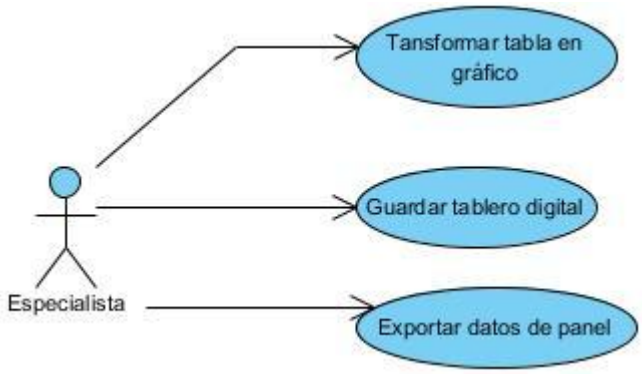

Figura 3. Diagrama de Casos de Uso del Sistema de Diseñador de Tablero Digital.

En la figura 4 se muestra el Diagrama de Casos de Uso para el módulo Diseñador de Modelos. Está compuesto por un actor, que es el encargado de diseñar un modelo determinado. A continuación se muestra la descripción de los casos de uso:

CU **Diseñar modelo**: Permite al diseñador crear un nuevo modelo de datos que será usado en la creación de los tableros digitales.

CU **Administrar modelo**: Permite al diseñador ver, modificar o eliminar un modelo de datos creado anteriormente.

CU **Gestionar origen de datos**: Permite al diseñador crear, modificar, ver o eliminar los orígenes de datos con que se cuenta para la creación del modelo de datos.

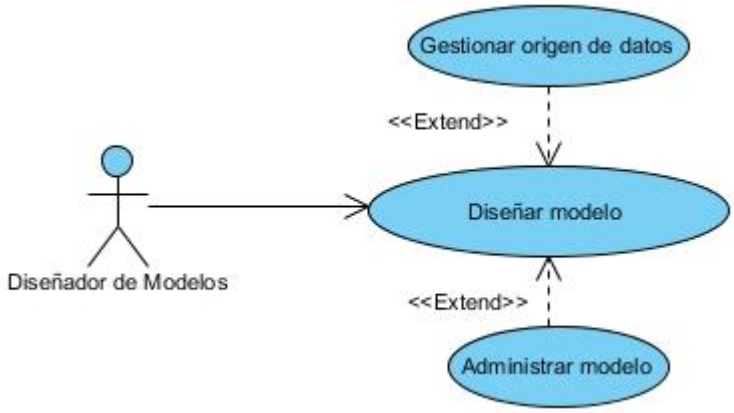

Figura 4. Diagrama de casos de uso del Sistema de Diseñador de Modelos.

La figura 5 muestra el Diagrama de Casos de Uso para el módulo Visor de Tablero Digital. Tiene un único actor encargado de iniciar el caso de uso. A continuación se muestra la descripción de este caso de uso:

CU **Cargar tablero digital**: Permite al usuario visualizar un tablero digital anteriormente guardado.

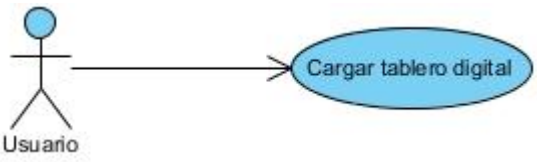

Figura 5. Diagrama de casos de uso del Sistema de Visor de Tablero Digital.

2.4.2 Vista Lógica.

La vista lógica representa los elementos de diseño más importantes para la arquitectura del sistema. Muestra de manera general una propuesta de subsistemas en los que estará dividida la aplicación. El sistema estará implementado usando Symfony, que es un framework que usa el patrón Modelo-Vista-Controlador; por lo que la estructura organizacional del tablero digital se representa a través de tres elementos fundamentales: el modelo, la vista y el controlador.

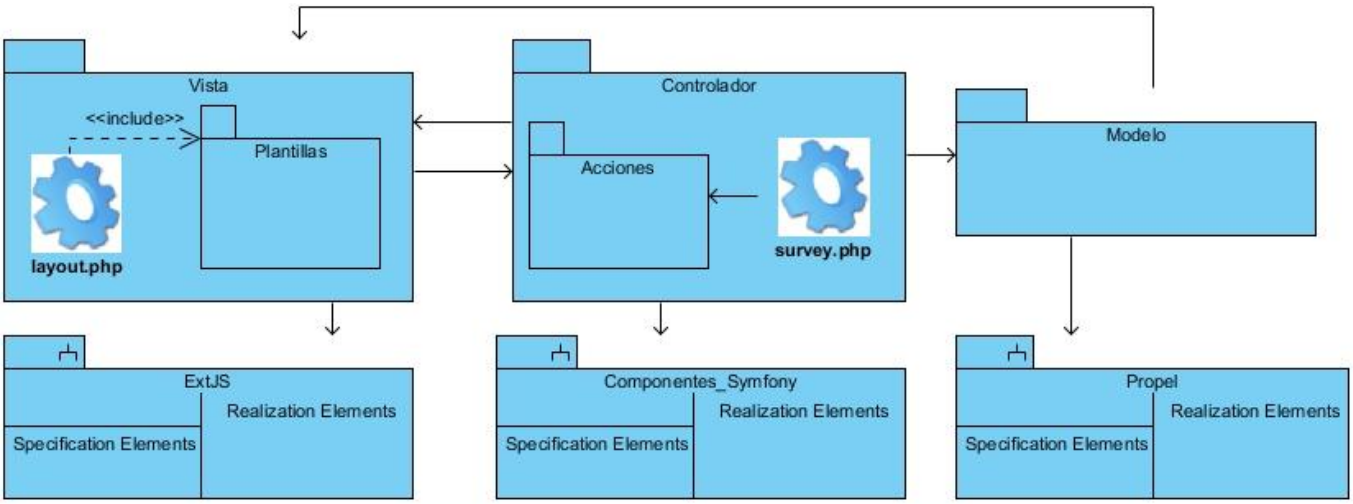

Figura 6. Vista Lógica General.

A continuación se describen los paquetes y subsistemas del diseño que conforma la vista lógica.

Modelo.

La capa del modelo es la que representa la abstracción de la base de datos. A grandes rasgos contiene la información con la que trabaja la aplicación, es decir, su lógica de negocio. ***Paquete del modelo.***

Dentro de este paquete están contenidas las clases que llevan implícita la lógica del dominio, son las encargadas del acceso a los datos almacenados en el gestor de base de datos y poseen el código para la manipulación de estos. La creación del modelo en Symfony se hace de manera automática atendiendo al esquema relacional de la base de datos. Por cada tabla del esquema relacional Symfony crea cuatro clases, las cuales son mostradas en la figura 6, poniendo de ejemplo la tabla "*indicador*". Las clases BaseIndicador.php y BaseIndicadorPeer.php son abstractas y contienen implementados los métodos básicos de acceso a los datos, no deben modificarse porque cada vez que se genera el modelo estas se renuevan. Las otras dos clases heredan de las nombradas "*base*" y es en estas donde se implementarán los métodos que los usuarios necesiten, ya que al generarse un nuevo modelo estas clases persisten. Dentro del Paquete Modelo se encuentran las clases necesarias para el acceso a los datos del sistema.

Subsistema Propel.

Las bases de datos son relacionales, mientras que PHP 5 y Symfony están orientados a objetos. Para acceder de forma efectiva a una base de datos desde un lenguaje de programación orientado a objetos, se necesita una herramienta informática capaz de traducir la lógica de objetos a la lógica relacional. Esta herramienta recibe el nombre de ORM. El ORM Propel tiene tareas para generar automáticamente las sentencias SQL necesarias para crear las tablas de la bases de datos. La librería Propel se encarga de esta generación automática, ya que crea la estructura básica de las

clases y genera automáticamente el código necesario. De modo que si en algún momento fuera necesario cambiar de gestor, no se tenga que realizar la traducción de las sentencias, con solo modificar el archivo de configuración se tendrán las consultas realizadas en el nuevo lenguaje.

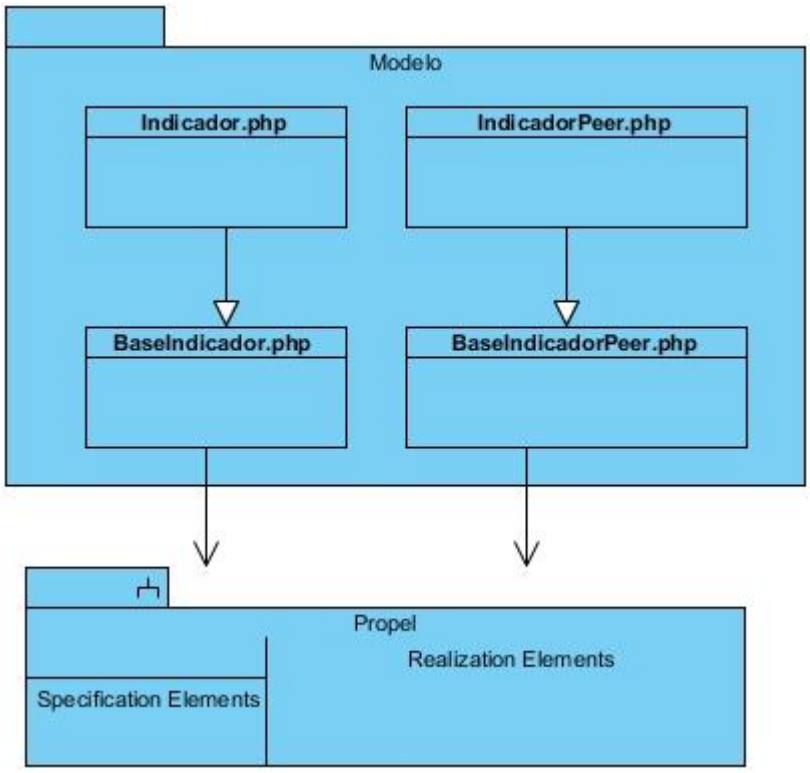

Figura 7. Vista Lógica. Paquete Modelo.

Vista.

La capa de vista es la encargada de mostrar los datos y resultados obtenidos por las acciones que se realizan. Constituye el punto de intercambio entre la aplicación y los usuarios.

Paquete vista.

La capa de vista de Symfony funciona como lo hacen la mayoría de las páginas web en la actualidad. Dichas páginas suelen tener elementos que se muestran ininterrumpidamente mientras la aplicación esté en curso: los pies de página, las barras laterales, los menús verticales, el banner o el fondo. Normalmente solo varía el interior de la página, que es el destinado a mostrar el contenido seleccionado. La estructura que se mantiene fija dentro de las páginas, en Symfony está guardada en un archivo que se denomina layout.php para mostrar una página web es preciso unir este archivo con una plantilla. De modo que una página web en Symfony está compuesta por el archivo layout.php y este a su vez incluye la plantilla que muestra el contenido deseado.

Paquete Plantillas.

Este paquete agrupa el contenido de cada módulo. Constituye la presentación de los datos de la acción que se está ejecutando en ese momento.

Subsistema ExtJS.

Esta potente librería basada en JavaScript posee un conjunto de componentes como cuadros de diálogo, gráficos, menús, tablas, pestañas y paneles, que usados de manera correcta le dan a la aplicación un diseño muy vistoso.

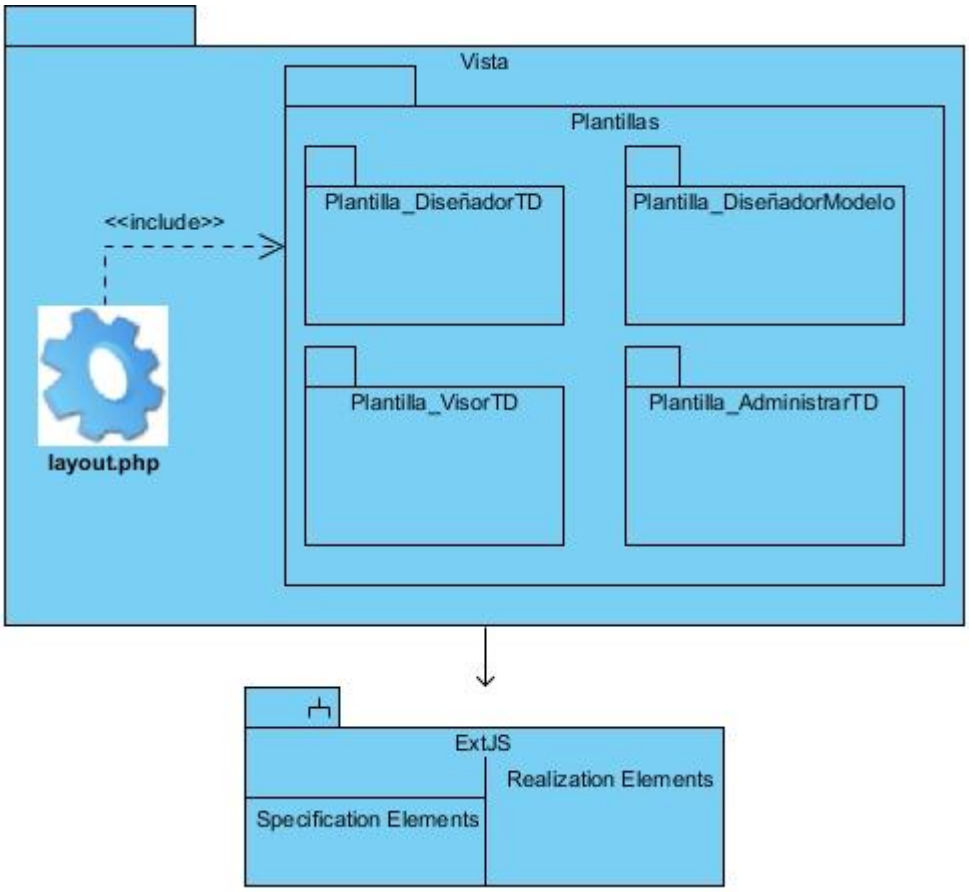

Figura 8. Vista Lógica. Paquete Vista.

Controlador.

La capa controladora es la que sirve de intercambio entre el modelo de datos y la interfaz de la aplicación. Es la encargada de gestionar las peticiones realizadas por los usuarios a través de la capa de vista.

Paquete Controlador.

En las aplicaciones Web desarrolladas con Symfony existe un controlador que es común para todos los módulos de la aplicación, y que constituye el único punto de entrada a la misma. En el caso del tablero digital el controlador frontal es la clase survey.php, todas las peticiones que se realicen entran al controlador frontal y este las distribuye en dependencia del módulo que esté realizando la petición.

Paquete Acciones:

Contiene la clase actions.php para cada módulo de la aplicación. Cuando alguna acción es requerida el controlador frontal ejecuta esta clase del módulo correspondiente. Esta clase define las acciones que se realizarán en el módulo donde esté contenida.

Subsistema Componentes_Symfony.

Representa las clases de Symfony que la aplicación usará y que ya se encuentran implementadas, tales como la validación de los formularios, componentes de seguridad, entre otros.

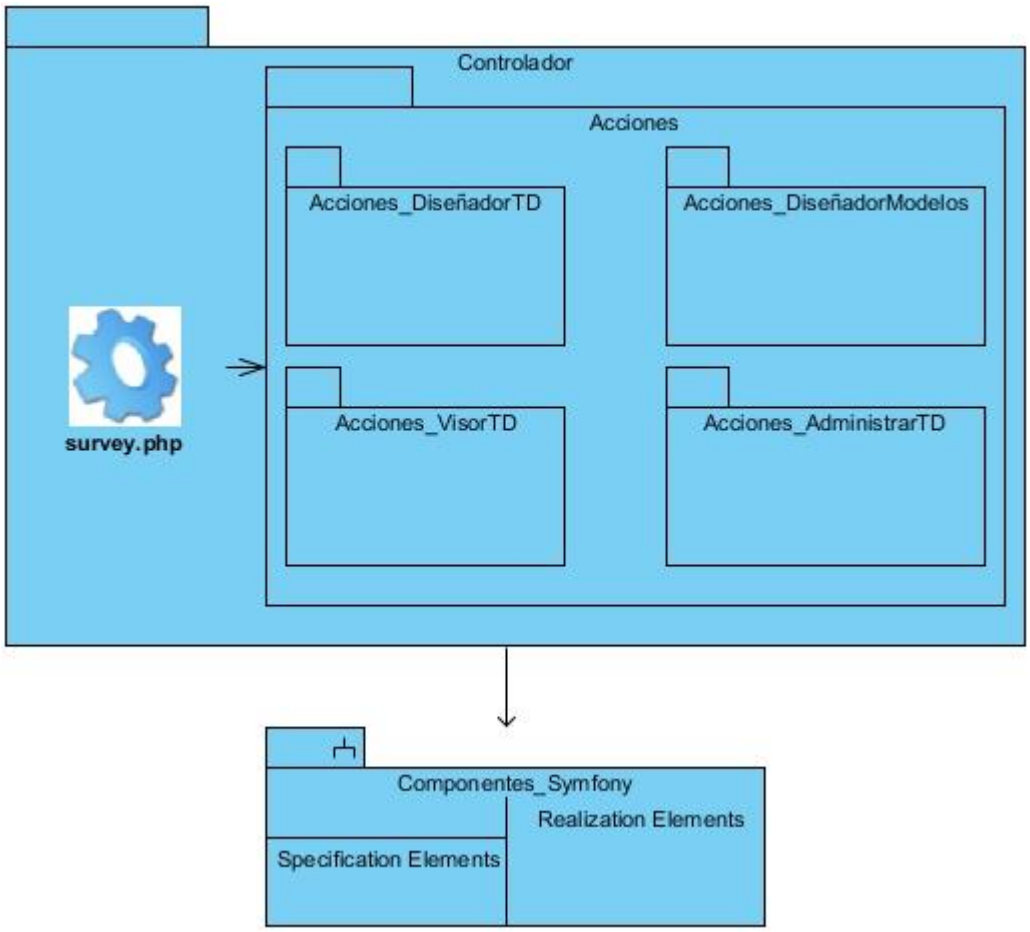

Figura 9. Vista Lógica. Paquete Controlador.

2.4.3 Vista de Implementación.

La vista de implementación está representada por un diagrama de componentes que refleja la organización de los módulos de software dentro del entorno de desarrollo. Muestra el software con los elementos físicos que lo integran. La vista de implementación del tablero digital está conformada por componentes de tipo fichero y subsistemas de implementación. Cada uno de los componentes puede ser implementado por separado y posteriormente integrarlo a la aplicación.

A continuación se muestra una breve descripción de cada uno de los componentes que se encuentran presentes en la vista de implementación diseñada:

Componente layout.php: Este componente va implementado en HTML o JS, y llevará los rasgos visuales de la página que sean comunes en todos los módulos.

Paquete Plantillas: Dentro de este paquete se encuentran las plantillas visuales de los módulos de la aplicación. También va implementado en HTML o JS y llevarán los elementos visuales específicos para cada módulo.

Paquete Acciones: Este paquete contiene las acciones de los módulos, va implementado en php y cada uno de los componentes que se encuentran dentro responde a la acción que realizará el módulo correspondiente.

Componente survey.php: Este componente hace la función de controlador frontal, el código que posee está escrito en php y es generado automáticamente por symfony. Esta clase se encarga de despachar las peticiones de la vista hacia el controlador que debe atender la solicitud.

Componente Origen de Datos: Este componente representa la base de datos contenida en el gestor, está escrito en código SQL.

Componentes tabla.php, BaseTabla.php, TablaPeer.php y BaseTablaPeer.php: Contienen los métodos básicos y los que se implementarán para asegurar la ejecución de las consultas y el acceso al origen de datos.

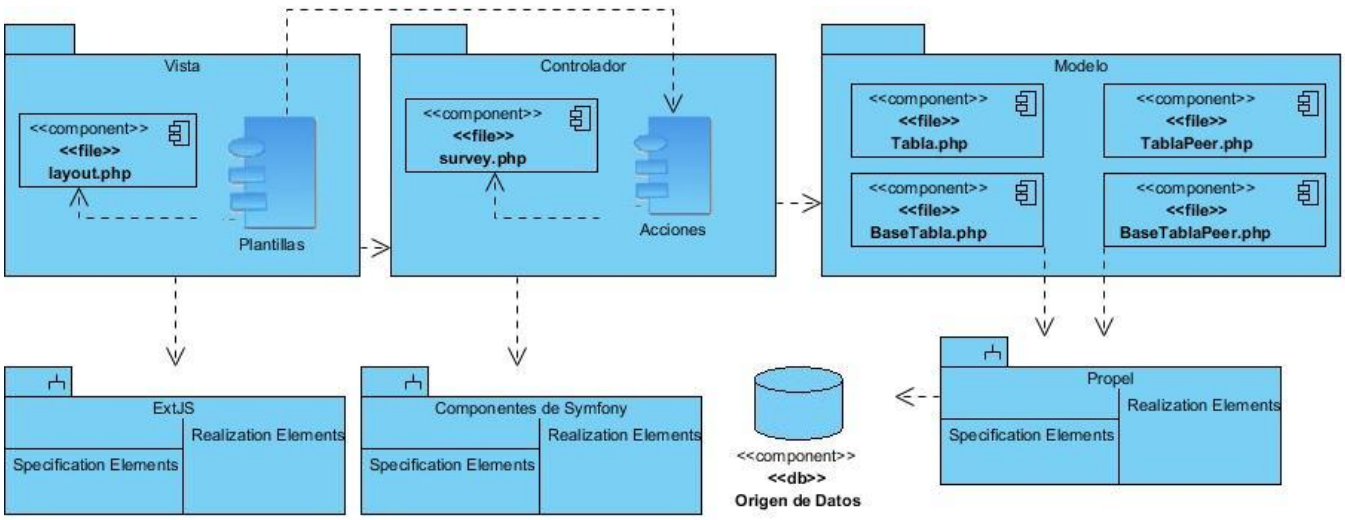

Figura 10. Vista de Implementación.

2.4.4 Vista de Despliegue.

La vista de despliegue describe los principales nodos físicos, ordenadores, así como los dispositivos que se necesitan para configurar la plataforma que pueda soportar la implementación del sistema. Describe la situación de los componentes en una posible implantación del sistema de acuerdo a los

requisitos iniciales. Es representada por un Diagrama de Despliegue y es un subconjunto del Modelo de Despliegue. Para representarla se tienen en cuenta los siguientes requerimientos: disponibilidad del sistema, rendimiento y escalabilidad.

A continuación se muestra una descripción de los elementos que conforman la vista de despliegue:

PC Cliente: Son las estaciones de trabajo que el especialista utilizará para acceder a la aplicación.

Servidor Web: Es el servidor utilizado para la publicación de la aplicación. Para lograr la conexión con la PC Cliente usa HTTP (Hypertext Transfer Protocol) como protocolo de comunicación.

Servidor de Bases de Datos: Es el servidor utilizado para contener los datos con los que trabajará la aplicación, de estos datos se obtendrán los modelos, y se crearán los tableros digitales. Para lograr la conexión con el servidor web se usa el protocolo TCP/IP.

Figura 11. Vista de Despliegue.

Conclusiones Parciales.

En este capítulo se realizó un análisis de la estructura central del sistema. Se mostraron los objetivos principales de la aplicación, así como las características que debe tener para hacerla más fiable, segura y vistosa. Se realizó la propuesta de la arquitectura del sistema a través de las 4+1 vistas definidas por Philippe Kruchten. El diseño de las vistas arquitectónicas permitió la representación del sistema desde distintas perspectivas, lo que le proporcionará a los implicados una visión común del mismo.

CAPÍTULO 3: Evaluación de la arquitectura.

Introducción.

No sirve de nada un sistema que no cumple con los atributos de calidad que se especificaron en los requerimientos no funcionales de los clientes. Por lo que diseñar una correcta arquitectura va a determinar el éxito o fracaso de un sistema de software, en la medida que esta cumpla o no con sus objetivos. [22]

Para reducir los riesgos que pueden conllevar al fracaso de la arquitectura y como una buena práctica de ingeniería es recomendable realizar evaluaciones a la arquitectura. Después de la evaluación de una arquitectura de software, se pueden tomar algunas decisiones como: si se puede seguir el proyecto con las áreas de debilidad dadas en la evaluación, si hay que reforzar la arquitectura del sistema o si hay que comenzar de nuevo el proceso.

3.1 Atributos de calidad.

La calidad del software se define como el grado en el cual el este posee una combinación deseada de atributos. Los cuales constituyen requerimientos del sistema que responden a las características que el sistema debe satisfacer y es lo que se conoce como atributo de calidad. El modelo de calidad expuesto en la norma ISO/IEC 9126 agrupa los atributos de calidad en seis características (Funcionalidad, Fiabilidad, Usabilidad, Eficiencia, Mantenibilidad y Portabilidad), cada una de estas se dividen a su vez en subcaracterísticas que podrán ser medidas a través de métricas internas y externas.

Funcionalidad: Se refiere a la capacidad del software de proveer un conjunto de funciones que permitan cumplir con unos requerimientos o satisfacer unas necesidades implícitas. Estas funciones se deben ejecutar bajo ciertas condiciones.

Fiabilidad: Capacidad de que el software mantenga un nivel especifico de desempeño cuando es usado bajo ciertas condiciones.

Usabilidad: La capacidad del software de que sea perceptible y de fácil aprendizaje.

Eficiencia: Es la capacidad del software para proporcionar una ejecución apropiada, relativo a la cantidad de recursos usados bajo ciertas condiciones.

Mantenibilidad: La capacidad que tiene el software a ser modificado. Estas modificaciones pueden ser correcciones, mejoras o adaptaciones del software a cambios en el ambiente y a requerimientos y especificaciones funcionales.

Portabilidad: La capacidad que tiene el software a ser transferido a otros ambientes. El ambiente puede ser la organización, hardware o software.

3.2 Técnicas de evaluación de la arquitectura de software.

Las técnicas usadas para la evaluación de atributos de calidad requieren grandes esfuerzos para crear especificaciones y predicciones. Entre las técnicas de evaluación de arquitecturas de software se destacan: evaluación basada en escenarios, evaluación basada en simulación, evaluación basada en modelos matemáticos y evaluación basada en experiencia. [21]

Técnicas de Evaluación	Instrumentos de Evaluación
Basada en Escenarios	• Profiles • Árbol de utilidad
Basada en Simulación	• ADLs • Modelos de colas
Basada en Modelos Matemáticos	• Cadenas de Markov • Reliabilidad y Diagrama de bloques
Basada en Experiencia	• Intuición y experiencia • Tradición • Proyectos similares

Tabla 1. Instrumentos asociados a las Técnicas de Evaluación.

3.2.1 Evaluación basada en escenarios.

Un escenario es una breve descripción de la interacción de alguno de los involucrados en el desarrollo del sistema. 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. Puede incluir ejecución de tareas, cambios en el sistema, ejecución de pruebas y configuración. El contexto 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. Las técnicas basadas en escenarios cuentan con dos instrumentos de evaluación relevantes: el Utility Tree propuesto por Kazman y los Profiles, propuestos por Bosch. En este caso se explicará el Utility Tree por ser la empleada en la evaluación de la arquitectura.

Utility Tree

Es una representación de los atributos de calidad de un sistema en forma de árbol. Su nodo central o raíz constituye la utilidad del software. En el siguiente nivel se encuentran los atributos de calidad asociados al sistema, los cuales se van refinando a medida que suceden los niveles, hasta llegar al punto más bajo del árbol que serían los escenarios. La intención del uso del Utility Tree es la identificación de los atributos de calidad más importantes para un proyecto particular. Cada atributo

de calidad perteneciente al árbol contiene una serie de escenarios relacionados, y una escala de importancia y dificultad asociada, que será útil para efectos de la evaluación de la arquitectura.

3.3 Métodos de evaluación.

Un método de evaluación sirve de guía a los involucrados en el desarrollo del sistema, en la búsqueda de conflictos que puede presentar una arquitectura, y sus soluciones. [23] A continuación se verá la descripción del método que se utilizará para la evaluación.

3.3.1 Método ATAM

El método de evaluación ATAM comprende nueve pasos, agrupados en cuatro fases (Presentación, Investigación y Análisis, Pruebas y Reporte). A continuación son explicadas cada una de estas fases. [24]

Fase 1: Presentación.

- **Paso 1, Presentación del ATAM:** El equipo de evaluación presenta un panorama general de los pasos de ATAM para establecer las expectativas, las técnicas utilizadas y los resultados del proceso.
- **Paso 2, Presentación de las metas del negocio:** Se realiza la descripción de las metas del negocio que motivan el esfuerzo, y aclara que se persiguen objetivos de tipo arquitectónico. Se presenta brevemente el negocio y el contexto de la arquitectura.
- **Paso 3, Presentación de la arquitectura:** El arquitecto presenta un panorama de la arquitectura, describe la arquitectura, enfocándose en cómo ésta cumple con los objetivos del negocio.

Fase 2: Investigación y análisis.

- **Paso 4, Identificación de los enfoques arquitectónicos:** El equipo de evaluación y el arquitecto deben detallar los planteamientos arquitectónicos descubiertos en el paso anterior. Estos elementos son detectados, pero no analizados.
- **Paso 5, Generación del Utility Tree:** Se identifican, priorizan, definen y refinan los atributos de calidad más importantes en un formato de árbol de utilidad, especificados en forma de escenarios. Se anotan los estímulos y respuestas, así como se establece la prioridad entre ellos.
- **Paso 6, Análisis de los enfoques arquitectónicos:** Con base en los resultados del establecimiento de prioridades del paso anterior, se analizan los elementos del paso 4. Se

identifican riesgos arquitectónicos, puntos de sensibilidad y puntos de balance. Se utilizan las preguntas similares a las presentadas en el paso 5.

Fase 3: Pruebas.

- **Paso 7, Lluvia de ideas y establecimiento de prioridad de escenarios:** Con la colaboración de todos los involucrados, se complementa el conjunto de escenarios y se le da prioridad a los escenarios sobre la base de su importancia relativa.
- **Paso 8, Análisis de los enfoques arquitectónicos:** Este paso repite las actividades del paso 6, haciendo uso de los resultados del paso 7. Los escenarios son considerados como casos de prueba para confirmar el análisis realizado hasta el momento.

Fase 4: Reporte.

- **Paso 9, Presentación de los resultados:** Basado en la información recolectada a lo largo de la evaluación del ATAM, se presentan los hallazgos a los participantes. El equipo de evaluación recapitula la ATAM pasos, los resultados, y recomendaciones.

3.4 Procedimiento de evaluación propuesto.

Para la evaluación de la arquitectura se tomaron en cuenta los atributos de calidad relacionados directamente con la arquitectura del software: funcionalidad, fiabilidad, usabilidad, eficiencia, mantenibilidad y portabilidad, definidos por la ISO/IEC 9126. Además se decidió seguir la línea de la técnica basada en escenarios con el instrumento de evaluación Utility Tree ya que van implícitos en el método ATAM seleccionado para realizar la evaluación, el cual apoya a los involucrados con el proyecto a entender las consecuencias de las decisiones arquitectónicas respecto a los atributos de calidad del sistema.

3.5 Evaluación de la arquitectura.

3.5.1 Identificación de los escenarios.

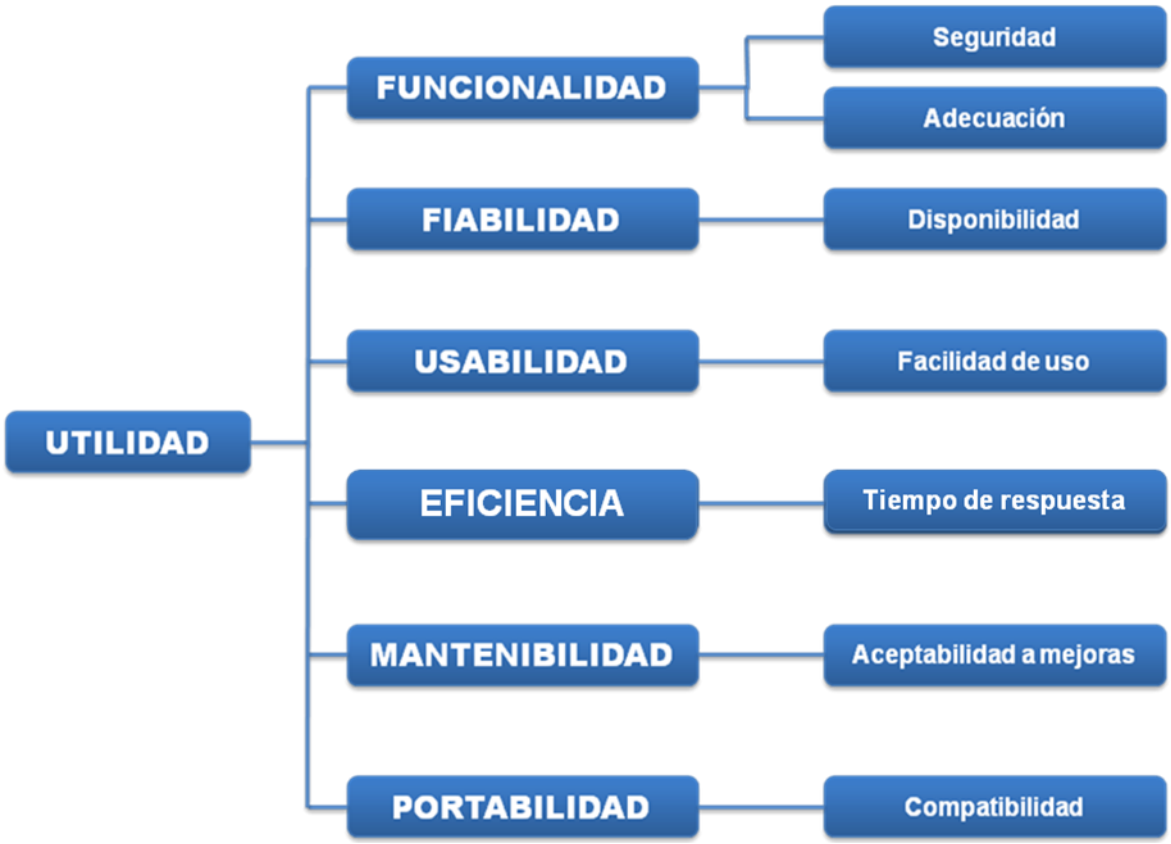

Figura 12. Árbol de Utilidad.

A los atributos de calidad se les asignan escenarios que fueron determinados como resultado de una tormenta de ideas entre los involucrados. Los escenarios deben ser ordenados y priorizados para que los arquitectos puedan contar con más orientaciones para tomar decisiones arquitectónicas. Después se procede a incluir los escenarios más importantes en el árbol de utilidad asignándole un orden de acuerdo a dos criterios.

a) Evaluar el riesgo de no considerar el escenario.
b) Evaluar el esfuerzo necesario para lograr cumplir con el escenario.

La prioridad del escenario define en este método como un par (X, Y) en el cual X define el esfuerzo de satisfacer el escenario, y la Y indica los riesgos que se corren al excluirlos del árbol de utilidad. Los posibles valores para X e Y son A (Alto), B (Bajo) y M (Medio).

Atributo de calidad	Subcaracterísticas	Escenario	Prioridad
Funcionalidad	Seguridad	E1. El sistema debe restringir el acceso a datos.	(M,A)
	Adecuación	E2. El sistema debe cumplir con los requisitos funcionales y los no funcionales.	(A,A)
Fiabilidad	Disponibilidad	E3. El sistema debe permanecer disponible, excepto para tareas de mantenimiento y cambio de configuración.	(A,M)
Usabilidad	Facilidad de uso	E4. El sistema debe ser vistoso y fácil de utilizar.	(B,M)
Eficiencia	Tiempo de respuesta	E5. El sistema debe tener tiempos de respuesta razonables.	(A,M)
Mantenibilidad	Aceptabilidad a mejoras	E6. El sistema debe tener la posibilidad de ser mejorado sin necesidad de volver a implementar la aplicación.	(M,M)
Portabilidad	Compatibilidad	E7. El sistema debe ser funcional con al menos los 3 navegadores más usados.	(B,A)

Tabla 2. Prioridad y categorización de los escenarios.

3.5.2 Análisis de los escenarios.

Una vez definidos los escenarios más importantes por cada uno de los atributos de calidad, es necesario realizar un análisis profundo a cada uno teniendo en cuenta cómo reacciona la arquitectura del sistema. El éxito de esta fase lo proporcionará la identificación de las amenazas potenciales para el sistema.

Cada tabla constituye el análisis de un escenario. Está compuesta por el atributo al que pertenece el escenario, el estímulo, la respuesta, las decisiones arquitectónicas y la explicación sobre cómo responde la arquitectura ante este evento.

El objetivo de ATAM no es predecir exactamente el comportamiento del atributo de calidad. Esto será imposible en etapas tempranas pues no se tiene suficiente información sobre el sistema como para hacer esta predicción. Es por ello que ATAM centra su labor en la identificación de los riesgos. Al finalizar la aplicación del método se tendrán identificados los riesgos, puntos sensibles y puntos de intercambio.

Los riesgos son decisiones arquitectónicamente importantes que no han sido tomadas, o que ya fueron tomadas pero las consecuencias no han sido entendidas a plenitud.

Los puntos sensibles son una propiedad de la arquitectura que es fundamental para el logro de un atributo de calidad específico (por ejemplo, mediante el cifrado se logra la confidencialidad).

Un punto de equilibrio es un punto de sensibilidad que es sensible o afecta a atributos de calidad múltiples (por ejemplo, el cifrado requiere de tiempo y afecta a latencia).

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

Escenario #1	Un usuario no autorizado intenta acceder al sistema.			
Atributo(s)	Funcionalidad-Seguridad.			
Ambiente	Operación normal.			
Estímulo	El usuario intenta acceder al sistema.			
Respuesta	Mensaje de error. Usuario o contraseña incorrectos.			
Decisiones arquitectónicas	**Punto de sensibilidad**	**Punto de intercambio**	**Riesgo**	**No riesgo**
	S1			
Explicación	La integridad de la información resulta extremadamente importante en los sistemas informáticos. Para mantener la información resguardada la aplicación cuenta con un sistema de gestión de usuarios por roles. Cada uno de los usuarios solo podrá acceder al módulo que esté autorizado.			

Tabla 3. Análisis del atributo funcionalidad-seguridad.

Escenario #2	El sistema cumple con los requisitos funcionales y los no funcionales.			
Atributo(s)	Funcionalidad-Adecuación.			
Ambiente	Operación normal.			
Estímulo	El usuario hace uso de las funcionalidades del sistema.			
Respuesta	Realización de cada una de las funcionalidades ordenadas.			
Decisiones	**Punto de**	**Punto de**	**Riesgo**	**No riesgo**

arquitectónicas	**sensibilidad**	**intercambio**		
		T1		
Explicación	La arquitectura propuesta satisface los requisitos funcionales y no funcionales definidos.			

Tabla 4. Análisis del atributo funcionalidad-adecuación.

Escenario #3	El sistema se encuentra fuera de servicio.			
Atributo(s)	Fiabilidad-Disponibilidad.			
Ambiente	Operación anormal.			
Estímulo	Realización de tareas de mantenimiento o cambios de configuración.			
Respuesta	Los usuarios no tienen acceso a la aplicación.			
Decisiones arquitectónicas	**Punto de sensibilidad**	**Punto de intercambio**	**Riesgo**	**No riesgo**
			R1	
Explicación	La aplicación no traerá implementada ninguna alternativa para que los usuarios trabajen sin conexión al servidor. Todas las funcionalidades que brinda el sistema requieren intercambio con el servidor web y el servidor de bases de datos.			

Tabla 5. Análisis del atributo fiabilidad-disponibilidad.

Escenario #4	El sistema debe ser vistoso y fácil de usar.			
Atributo(s)	Usabilidad-Facilidad de uso.			
Ambiente	Operación normal.			
Estímulo	El usuario hace uso de las funcionalidades del sistema por primera vez.			
Respuesta	Realización de cada una de las funcionalidades ordenadas.			
Decisiones arquitectónicas	**Punto de sensibilidad**	**Punto de intercambio**	**Riesgo**	**No riesgo**
		T2		
Explicación	La utilización de la librería visual ExtJS permitirá obtener una aplicación entendible, vistosa y con interfaces amigables.			

Tabla 6. Análisis del atributo usabilidad-facilidad de uso.

Escenario #5	La aplicación tendrá tiempos de respuesta razonables
Atributo(s)	Eficiencia-Tiempo de respuesta.

Ambiente	Operación normal.			
Estímulo	El usuario interactúa con el sistema solicitando los datos de un tablero digital.			
Respuesta	Realización de la funcionalidad en un tiempo mínimo aproximado de 2 segundos.			
Decisiones arquitectónicas	**Punto de sensibilidad**	**Punto de intercambio**	**Riesgo**	**No riesgo**
				NR1
Explicación	La funcionalidad “Cargar tablero digital” es una de las más complejas de la aplicación y la que más tiempo emplea en terminar. El tiempo mínimo establecido para el cumplimiento de esta función está dado por una prueba con red interna sin tráfico y sin temporales almacenados en el navegador.			

Tabla 7. Análisis del atributo eficiencia-tiempo de respuesta.

Escenario #6	Es necesario incorporar nuevas funcionalidades a la aplicación.			
Atributo(s)	Mantenibilidad-Aceptabilidad a mejoras.			
Ambiente	Operación normal.			
Estímulo	Necesidad de adicionar funcionalidades al sistema.			
Respuesta	Posibilidad de adicionar funcionalidades sin necesidad de reimplementar la aplicación.			
Decisiones arquitectónicas	**Punto de sensibilidad**	**Punto de intercambio**	**Riesgo**	**No riesgo**
	S2			
Explicación	La arquitectura diseñada brinda la posibilidad de integrarle otros módulos a la aplicación. También permite integrarle funcionalidades a los módulos existentes. Es una de las ventajas principales con que cuenta el principio de programación orientada a componentes que usa el sistema.			

Tabla 8. Análisis del atributo mantenibilidad-aceptabilidad a mejoras.

Escenario #7	El sistema debe ser funcional con los 3 navegadores más usados.			
Atributo(s)	Portabilidad-Compatibilidad			
Ambiente	Operación normal.			
Estímulo	Uso de la aplicación en Firefox 4.0, Chrome 15.0 o Internet Explorer 8			
Respuesta	Realización de cada una de las funcionalidades ordenadas.			
Decisiones	**Punto de**	**Punto de**	**Riesgo**	**No riesgo**

arquitectónicas	sensibilidad	intercambio		
	S3			
Explicación	Durante la implementación de la aplicación se realizaron pruebas de compatibilidad con estos navegares. De manera que el sistema es completamente compatible con Firefox 4.0 o superior y completamente funcional con Chrome 15 e Internet Explorer 8 o superiores versiones.			

Tabla 9. Análisis del atributo portabilidad-compatibilidad.

3.5.3 Resultados obtenidos.

Se analizaron un total de 7 escenarios, de los cuales se identificaron 3 puntos de sensibilidad, un no riesgo, 2 puntos de intercambio y un riesgo.

Riesgos

- La aplicación no traerá implementada ninguna alternativa para que los usuarios trabajen sin conexión al servidor.

No riesgos

- La aplicación tendrá tiempos de respuesta razonables.

Puntos de sensibilidad

- Acceso a la aplicación por usuarios no autorizados.
- Posibilidad de incrementar funcionalidades sin necesidad de realizar una nueva versión de la aplicación desde el inicio.
- Compatibilidad con navegadores.

Puntos de intercambio

- El sistema cumple con los requisitos funcionales y los no funcionales.
- El sistema debe ser fácil de usar y con interfaces amigables y vistosas.

3.6 Toma de decisiones.

La toma de decisiones se ejecuta con el objetivo de mitigar los riesgos en la arquitectura desde el punto de vista de los atributos de calidad analizados y dar paso a la construcción del sistema. A raíz del riesgo identificado se toma la decisión de definir una estrategia para facilitar las tareas de mantenimiento y cambios de configuración del sistema con el fin de que este se encuentre deshabilitado el menor tiempo posible.

Conclusiones parciales.

En este capítulo se llevó a cabo la evaluación temprana de la arquitectura del sistema. Se utilizó el método de evaluación ATAM, junto a la técnica de evaluación basada en escenarios. La herramienta utilizada para la evaluación fue el Utility Tree, que es la que viene definida por ATAM. Al evaluar cada uno de los escenarios por atributo de calidad se obtuvieron 3 puntos de sensibilidad, 2 puntos de intercambio, un no riesgo y un riesgo. Analizando los resultados se tomaron las decisiones pertinentes. Se determinó que el sistema es funcional y la arquitectura posee un diseño válido por lo que se procede con la implementación.

CONCLUSIONES GENERALES

Un diseño acertado de la arquitectura del software constituye uno de los aspectos más importantes en el ciclo de vida del proyecto, posibilita una mayor visión y un lenguaje común entre los desarrolladores. Además de estructurar el sistema según los patrones, estilos, metodologías y tecnologías actuales. Durante el transcurso de esta investigación:

- Se realizó un estudio sobre las tendencias actuales en la implementación de los tableros digitales y se definieron las herramientas informáticas que se utilizarán en el proceso de desarrollo del proyecto.
- Fue descrito el diseño de la arquitectura de software mediante la vista de casos de uso, vista lógica, vista de implementación y la vista de despliegue.
- Se implementó un componente para la gestión de los usuarios que tendrán acceso a la aplicación.
- Se validó mediante el método ATAM, el diseño de la arquitectura propuesto para la implementación de la solución.
- La arquitectura propuesta fue aplicada en el desarrollo de los módulos del tablero digital, contribuyendo de manera significativa a que el mismo cumpliera con todos los requerimientos. Los resultados obtenidos en esta investigación pueden ser usados en otros sistemas de características similares.

RECOMENDACIONES

- Investigar la posibilidad de que el tablero digital implementado posea una actualización en tiempo real.
- Investigar la posibilidad de incluir servicios web para enriquecer la usabilidad del sistema.

REFERENCIAS BIBLIOGRÁFICAS

1. **Playfair, William.** *The commercial and political atlas.* 1786.

2. **Reynoso, Carlos.** *Introducción a la arquitectura de software.* Buenos Aires : s.n., 2004.

3. **Len Bass, Rick Kazman.** *Software Architecture in Practice.* 2003.

4. **Baragry, Jason.** *Why we need a different view of software architecture.* 2001.

5. **Kruchten, Philippe.** *The "4+1" View Model of Software Architecture.* 1995.

6. **Pressman, Roger.** *Ingeniería de Software.* s.l. : McGraw-Hill, 2005.

7. **Monreal, Juan Ramírez.** *Desarrollo de un Dashboard en la Plataforma Paula.* 2008.

8. **Kaplan, Robert S.,David P. Norton.** *The Balanced Scorecard: Translating Strategy Into Action.* Boston : s.n., 1996.

9. **Sixtina Consulting Group.** *El diseño del Dashboard.* 2008.

10. **Yorio, Dario.** *Identificación y Clasificación de Patrones en el Diseño de Aplicaciones Móviles.* 2006.

11. **Rosanigo, Zulema Beatriz.** *Modelos y Patrones.* 2000.

12. **IEEE.** *Recommended practice for architectural description of software-intensive systems.* 2000.

13. **Reynoso, Carlos.** *Estilos y Patrones en la Estrategia de Arquitectura de Microsoft.* 2004.

14. **Erich Gamma, Ralph Johnson, John Vlissides.** *Design Patterns: Elements of reusable object-oriented software.* 1995.

15. **Reynoso, Carlos.** *Lenguajes de Descripción de Arquitectura.* 2006.

16. **Ivar Jacobson, James Rumbaugh.** *El Proceso Unificado de Desarrollo.* Madrid : A. Wesley, 2000.

17. **Stig Saether Bakken,Alexander Aulbach, Egon Schmid, Jim Winstead, Lars Torben Wilson, Rasmus Lerdorf, Andrei Zmievski, Jouni Ahto.** Manual de PHP. 2002.

18. **Francois Zaninotto, Fabien Potencier.** *The Definitive Guide to symfony.* 2007.

19. *Ext - A foundation you can build on.* [En línea] http://www.extjs.com/5064.

20. **Gracia, Joaquin.** *Diseño de Software Orientado a Objetos.* 2005.

21. **Gómez, Omar Salvador.** *Evaluando Arquitecturas de Software.* México : s.n., 2007.

22. **Gustavo Andrés Brey, Nicolas Passerini, Juan Arias.** *Arquitectura de Proyectos de IT. Evaluación de Arquitecturas, in Departamento de Sistemas.* Buenos Aires : s.n., 2005.

23. **Alexsander Gonzalez, Marizé Mijares, Luis E. Mendoza, Anna Grimán, María Pérez**. Método *de Evaluación de Arquitecturas de Software Basadas en Componentes (MECABIC)*. 2005. Vol 1.

24. **Rick Kazman, Paul Clements, Mark Klein.** *ATAM: Method for Architecture Evaluation.* 2000.

BIBLIOGRAFÍA.

- **Jacobson, Ivar; Booch, Grady; Rumbaugh, James**. “El Proceso Unificado de Desarrollo de Software”. 2004.
- **Reynoso, Carlos.** Introducción a la arquitectura de software. Buenos Aires: s.n., 2004.
- **Pressman, Roger**.” Ingeniería del Software. Un enfoque práctico”. 2005.
- **Rosanigo, Zulema Beatriz.** Modelos y Patrones. 2000.
- **Reynoso, Carlos.** *Estilos y Patrones en la Estrategia de Arquitectura de Microsoft.* 2004.
- **Kruchten, Philippe.** *The “4+1” View Model of Software Architecture.* 1995.
- **Rick Kazman, Paul Clements, Mark Klein.** *ATAM: Method for Architecture Evaluation.* 2000.