

Universidad de las Ciencias Informáticas

FACULTAD 6



Título: Herramienta para la evaluación de Arquitecturas de Software
en el Centro de Tecnologías de Gestión de Datos (DATEC)

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

Autores: Gleibis Ivonnet Hervella
José Lázaro Izquierdo Rodríguez

Tutores: Ing. Dairys Febles Pérez
Ing. Annia Susel Jiménez Peña

La Habana, junio del 2012.
“Año 54 de la Revolución”



“La revolución es algo que se lleva en el alma, no en la boca para vivir de ella.”

Ernesto Guevara de la Serna

DECLARACIÓN DE AUTORÍA

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

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

Gleibis Ivonnet Hervella

Firma del Autor

José Lázaro Izquierdo Rodríguez

Firma del Autor

Ing. Dairys Febles Pérez

Firma del Tutor

Ing. Annia Susel Jiménez Peña

Firma del Tutor

DATOS DE CONTACTO

TUTORES

Nombre: Dairys Febles Pérez

E-mail: dfebles@uci.cu

Nombre: Annia Susel Jiménez Peña

E-mail: asjimenez@uci.cu

AGRADECIMIENTOS

Hoy celebramos el fin de una etapa importante en nuestra vida, nos despedimos de grandes amigos y profesores, a quienes agradecemos su comprensión y ayuda incondicional durante todos estos años, pero en especial:

A mi compañero de Tesis José Lázaro, por hacer realidad mi sueño, por aguantar mis malcriadeces aunque yo se las aguanté más a él; SABES QUE SIN TI ESTO NO HUBIERA SIDO POSIBLE.

A mis padres, las personas más importantes de mi vida, que han sacrificado gran parte de su vida para formarme y educarme, que me enseñaron a ser fuerte en los momentos débiles, a tener fe y me guiaron siempre por buen camino; GRACIAS POR ESTAR EN MI VIDA y SER PARTE DE LO QUE MÁS AMO.

A la Virgen María y a Jesucristo por escucharme y ser mi apoyo en mis momentos de inseguridad, por mantener a mis seres queridos y a mí con vida, salud, paz, prosperidad, inteligencia y guiarnos hasta hoy.

A mi hermano que aunque se encuentra lejos y no puede estar conmigo en este momento está presente en mi corazón.

A mi novio Abelito por estar a mi lado en todo momento, darme fuerzas cuando creía que todo estaba perdido, por darme ese infinito amor que alegra cada día de mi vida y por ser la persona más especial que he conocido.

A mis suegros Abel y Maricelis por acogerme como una hija y darme su apoyo y cariño incondicional.

A mis tutoras Annia y Dairys por apoyarnos y guiarnos en el trabajo hasta en los momentos más difíciles.

Al profesor Lobo por habernos dado la luz cuando todo era oscuridad.

A mis amigos, por los buenos momentos que pasamos juntos y los recuerdos que quedaron grabados para siempre, en especial a mis amigas Yoanna y Marisbelis que más que unas amigas son unas hermanas para mí, por tantas experiencias vividas juntas en todos estos años que nunca olvidaré.

A mis compañeros de baile que me hicieron vivir muchos momentos de felicidad a lo largo de mi carrera.

En general, agradezco a todos aquellos que están presentes y que pusieron su granito de arena para que un día como hoy, pudiera dedicarle estas palabras. A todos ustedes, Gracias, Muchas Gracias.

Gleibis Ivonnet Hervella

A mi compañera de Tesis Gleibis por aguantarme durante todo un año y no rendirse nunca, al final no eres tan malcriada como crees.

A mis padres por mantenerse incondicionalmente a mi lado durante todo el camino que condujo a este momento.

A mis hermanos y el resto de mi familia por mantenerse siempre a mi lado y brindarme su apoyo.

A mis amigos de estos 5 años los que están hoy aquí y los que no por los momentos de alegría que compartimos, todos aportaron un poco aunque no lo crean.

A mis tutoras por su preocupación durante este tiempo y por los regaños cuando hicieron falta.

A mis profes de todos estos años, de todos aprendí algo.

José Lázaro Izquierdo Rodríguez

DEDICATORIA

A la mujer más linda del mundo, mi mamá, por todo lo que has hecho en la vida por mí, solo de darme la posibilidad de venir al mundo fue lo más hermoso que cualquier ser humano puede pedir, por su amor, confianza, dedicación, apoyo, por ser guía ejemplar y constante, por ser una madre excepcional y no te comparo con nada en este mundo, sin ser perfecta me has guiado por el buen camino. Te quiero mucho.

A mi papi, por su amor, sus enseñanzas, sus años de sacrificio, su confianza, sus consejos, y apoyo en cada momento, por confiar tanto en mí y estar seguro que no te defraudaría, te quiero mucho.

Padres de mi vida, he llegado al final de este camino y en mi han quedado marcadas huellas profundas de éste recorrido, quiero que sientan que el objetivo logrado también es tuyo y que la fuerza que me ayudó a conseguirlo fue su apoyo. No olviden que mi trofeo es también suyo.

Gleibis Ivonnet Hervella

A mis padres, ustedes son lo más importante que tengo, gracias por la educación que he recibido de ustedes y por estar siempre presentes.

A mis hermanos, los quiero mucho y espero ser un buen ejemplo como hermano mayor.

José Lázaro Izquierdo Rodríguez

RESUMEN

La investigación está asociada al trabajo del Grupo de Arquitectura del centro DATEC encargado de la definición y evaluación de arquitecturas de los productos desarrollados. Actualmente el proceso de evaluación se realiza de forma empírica a través de talleres de discusión.

El objetivo del presente trabajo de diploma es realizar el análisis, diseño e implementación de una herramienta para asistir al proceso de evaluación de la Arquitectura de Software a partir del método de evaluación ATAM, que cumpla con los requisitos funcionales y no funcionales exigidos por el cliente. Como resultado se obtuvo una herramienta web que mejora sustancialmente el proceso de evaluación, lo que favorece a los arquitectos en la toma de decisiones para apoyar la implementación de la arquitectura propuesta.

En el trabajo se presenta un estudio de las herramientas y tecnologías utilizadas para la construcción del mismo. Se exponen además las funcionalidades implementadas así como la documentación generada en cada una de las fases de la metodología utilizada.

PALABRAS CLAVES

Arquitectura de Software, ATAM, evaluación de arquitectura de software.

ÍNDICE

AGRADECIMIENTOS	I
DEDICATORIA	III
RESUMEN	IV
INTRODUCCIÓN	1
CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA	5
1.1 ARQUITECTURA DE SOFTWARE	5
1.2 EVALUACIÓN DE LA ARQUITECTURA DE SOFTWARE	6
1.2.1 MÉTODOS PARA LA EVALUACIÓN DE LA ARQUITECTURA DE SOFTWARE	7
1.3 METODOLOGÍAS DE DESARROLLO	15
1.4 LENGUAJE DE MODELADO	18
1.5 HERRAMIENTA DE MODELADO	18
1.6 IDE DE DESARROLLO	20
1.7 LENGUAJE DE PROGRAMACIÓN	21
1.8 FRAMEWORK DE DESARROLLO	22
1.9 GESTOR DE BASE DE DATOS	23
1.10 HERRAMIENTA PARA REALIZAR PRUEBAS DE SOFTWARE	25
1.11 CONCLUSIONES DEL CAPÍTULO	26
CAPÍTULO 2: ANÁLISIS Y DISEÑO DEL SISTEMA	27
2.1 ESPECIFICACIÓN DE LOS REQUISITOS DEL SISTEMA	27
2.2 MODELO DE CASOS DE USO DEL SISTEMA	30
2.3 DIAGRAMA CONCEPTUAL DE LA HERRAMIENTA	35
2.4 DISEÑO DEL SISTEMA	37
2.5 CONCLUSIONES DEL CAPÍTULO	46
CAPÍTULO 3: IMPLEMENTACIÓN Y PRUEBA	47
3.1 MODELO DE DESPLIEGUE	47
3.2 MODELO DE IMPLEMENTACIÓN	48
3.3 CÓDIGO FUENTE	49
3.4 PRUEBAS	51
3.5 CONCLUSIONES DEL CAPÍTULO	57
CONCLUSIONES	58
RECOMENDACIONES	59
REFERENCIAS BIBLIOGRÁFICAS	60
BIBLIOGRAFÍA	62
ANEXOS	64
ANEXO 1: INTERFACES PRINCIPALES DE LA APLICACIÓN	64
ANEXO 2: CARTA DE ACEPTACIÓN	65
GLOSARIO	66

ÍNDICE DE FIGURAS

Figura 1: Diagrama de caso de uso del sistema.....	31
Figura 2: Diagrama conceptual de la herramienta.....	35
Figura 3: Diagrama de clases del diseño CU Gestionar_escenario.....	41
Figura 4: Diagrama de secuencia CU Adicionar escenario.	43
Figura 5: Diagrama de secuencia CU Eliminar escenario.	44
Figura 9: Diagrama de clases persistentes.	45
Figura 10: Modelo de datos.	45
Figura 11: Modelo de despliegue.	47
Figura 12: Diagrama de componentes CU Gestionar_escenario.....	48
Figura 13: Ejemplo de Código Fuente: Método Adicionar escenario.	50
Figura 14: Resultado de la prueba de Carga.....	56
Figura 15: Resultado de la prueba de Estrés.	56
Figura 16: Interfaz de autenticación.	64
Figura 17: Interfaz de Adicionar un escenario.	64
Figura 18: Interfaz de Visualizar reporte de evaluación.....	65
Figura 19: Carta de Aceptación.....	65

ÍNDICE DE TABLAS

Tabla 1: Actores del sistema.	31
Tabla 2: Descripción del caso de uso Gestionar_escenario.	32
Tabla 3: No Conformidades por casos de uso en cada iteración.	51
Tabla 4: Caso de Prueba realizado al caso de uso Gestionar_escenario.	53

INTRODUCCIÓN

En los primeros años de la Informática, el desarrollo de software era centrado en la programación, sin embargo, existían problemas inherentes a esta forma de desarrollo: presiones en los tiempos de ejecución de un proyecto, ausencia de métodos controlados para cumplir con los objetivos planteados, continuos cambios en las funcionalidades requeridas y documentación pobre o inexistente. Con el transcurso del tiempo se fueron descubriendo y desarrollando formas de diseño de software que se fueron manifestando en los procesos de creación de los sistemas informáticos. Estas formas de diseño dieron origen a lo que más tarde sería conocido como Arquitectura del Software (AS): disciplina que intenta contrarrestar estos efectos negativos, ocupando cada vez más un rol significativo en la estrategia de negocio de una organización.

La AS está relacionada con el diseño y la implementación de estructuras de software de alto nivel, consiste en un conjunto de patrones y abstracciones coherentes que proporcionan el marco arquitectónico. Es el resultado de ensamblar un cierto número de elementos arquitectónicos de forma adecuada para satisfacer la mayor funcionalidad y eficiencia de un sistema, así como requerimientos no funcionales: la confiabilidad, escalabilidad, portabilidad y disponibilidad.

Durante el proceso de desarrollo de software se insertan riesgos que limitan y dificultan el proceso en sí, entre los que se perciben: la escasa reutilización, la mala selección de componentes de software, estilos arquitectónicos, patrones de arquitectura, protocolos de comunicación, composición de elementos de diseño, asignación de funcionalidad a elementos del diseño, herramientas, problemas de sincronización y acceso a datos, programadores escribiendo código de cualquier manera siempre y cuando cumpla con lo que quieren que haga el programa, influyendo todo esto en cuestiones de escalabilidad, rendimiento y costo, donde la solución a todos estos problemas está en la correcta definición y descripción de una línea base arquitectónica de software.

La AS surge de la necesidad de hacer los sistemas más modulares, que permitan la reutilización de componentes, por lo que su implementación reduce considerablemente el costo de desarrollo. Permite además mantener un bajo acoplamiento entre los elementos del sistema, pero con muy alta cohesión, ya que se establece bien claro la estructura de los componentes, sus formas de comunicarse y las relaciones que existen entre ellos.

Evaluar una AS sirve para prevenir todos los posibles errores de un diseño que no cumple con los requerimientos de calidad y para saber qué tan adecuada es la arquitectura aplicada para la

implementación del sistema. La evaluación no brinda un resultado específico de la arquitectura, indica dónde está el riesgo, es decir, fortalezas y debilidades identificadas. Después de terminar una evaluación, se pueden tomar algunas decisiones: si puede seguir el proyecto con las áreas de debilidad encontradas en la evaluación, si hay que redefinir la AS o comenzar de nuevo el proceso.

En Cuba a raíz de las dificultades económicas el estado comenzó a invertir en varios sectores estratégicos de la economía y empezó a destinar gran cantidad de recursos a elevar el nivel de informatización de la sociedad, el ejemplo más fehaciente lo constituye la creación de la Universidad de las Ciencias Informáticas (UCI) para desarrollar la industria del software y contribuir al desarrollo económico del mismo.

La UCI se presenta como una institución joven dentro del desarrollo de productos y servicios de software, promoviendo el avance de la informática tanto para el ámbito nacional como internacional, mediante la realización de software en los proyectos productivos, vinculando a estudiantes y profesores con un alto nivel científico y productivo.

Recientemente se realizó una reestructuración de los proyectos, agrupando estos en Centros de Desarrollo con temáticas afines, dentro de ellos se encuentra el Centro de Tecnologías de Gestión de Datos (DATEC) perteneciente a la Facultad 6 que apoya la producción de software. Este centro tiene como misión: proveer soluciones integrales, productos y servicios relacionados con las tecnologías de gestión de datos y análisis de la información, desarrollar el procesamiento y representación de la información a partir del perfeccionamiento de proyectos de Investigación y Desarrollo (I+D), con el objetivo de contribuir al cumplimiento de las misiones fundamentales propuestas a alcanzar por la Universidad.

El Centro está estructurado en cuatro departamentos uno de ellos: Soluciones Integrales que posee un Grupo de Arquitectura encargado de la definición y evaluación de arquitecturas de los productos desarrollados. Actualmente el proceso de evaluación se realiza de forma empírica a través de talleres de discusión entre los miembros de mayor experiencia del Centro, el proceso es realizado mediante un método de evaluación y los resultados son plasmados en un acta donde se redactan las pautas y decisiones tomadas, esto afecta el tiempo que demoran los arquitectos en tomar decisiones que apoyan la implementación de una arquitectura propuesta, además de no saber con certeza si la evaluación realizada es correcta, provocando atrasos en el desarrollo de los proyectos.

Por todo lo anteriormente expuesto se define como **problema de la investigación**: ¿Cómo contribuir a la toma de decisiones arquitectónicas para la evaluación de la AS en DATEC?

Donde el **objeto de estudio** es: La evaluación de la AS. Enmarcado en el **campo de acción**: La toma de decisiones arquitectónicas para la evaluación de la AS.

Se plantea como **objetivo general**: Desarrollar una herramienta web que asista la evaluación de la AS en DATEC.

Desglosado en los siguientes **objetivos específicos**:

1. Definir funcionalidades de la herramienta.
2. Diseñar la herramienta para asistir el proceso de evaluación de la AS en DATEC.
3. Implementar la herramienta a partir del estudio de métodos de evaluación de la AS.
4. Realizar pruebas de calidad de software a la herramienta desarrollada.

Para guiar la investigación se tienen las siguientes **tareas**:

1. Análisis de la bibliografía existente sobre el tema y confección del marco teórico de la investigación.
2. Estudio del estado actual de las herramientas y métodos de evaluación de arquitecturas.
3. Análisis y selección de la metodología de desarrollo, lenguajes de modelado, plataforma de desarrollo y framework a utilizar.
4. Modelado del negocio.
5. Especificación de los requerimientos funcionales y no funcionales del software.
6. Elaboración del análisis y diseño del sistema informático.
7. Implementación del software.
8. Diseño y aplicación de pruebas de calidad de software al sistema informático.

El presente trabajo está estructurado en tres capítulos donde se exponen el desarrollo y los resultados del proceso de selección, análisis, diseño e integración de soluciones para la gestión de proyecto:

➤ **Capítulo 1: Fundamentación Teórica**

En este capítulo se explica el estado del arte, se abordan brevemente los principales conceptos de la AS, los métodos de evaluación de la misma, las metodologías y las herramientas que se utilizarán para llevar a cabo el desarrollo del sistema, así como lenguajes y framework en el que se desarrollará la aplicación.

➤ **Capítulo 2:** Análisis y Diseño del sistema

En este capítulo se realiza el proceso de captura de los requerimientos funcionales y no funcionales que debe cumplir la aplicación, además se expone el diagrama de casos de uso del sistema y la descripción de los mismos. Se describe la representación arquitectónica de la aplicación, se hace énfasis en el patrón de arquitectura utilizado, así como los patrones de diseño que fueron empleados y se lleva a cabo el diseño del sistema.

➤ **Capítulo 3:** Implementación y Prueba

Está enfocado a la implementación del sistema con el objetivo de darle solución a los requerimientos funcionales, obteniéndose artefactos del flujo de implementación como: el modelo de implementación, los diagramas de componentes y el diagrama de despliegue. Además se realizan pruebas al software para comprobar la correcta implementación de cada una de las funcionalidades definidas.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

En este capítulo se realiza un estudio de los conceptos básicos de la AS, así como el estado actual de las herramientas y métodos de evaluación de la misma. Se hace énfasis en las metodologías a utilizar, así como lenguajes de programación, entorno de desarrollo, gestor de base de datos y framework para seleccionar los más adecuados para el desarrollo de la herramienta.

La herramienta a desarrollar será de tipo web, por las ventajas que presentan sobre las de escritorio:

- Facilitan el trabajo a distancia y de manera colaborativa.
- No se requieren complicadas combinaciones de Hardware/Software para utilizarlas, solamente contar con buen navegador web.
- Son fáciles de usar ya que no requieren conocimientos avanzados de computación.
- Poseen alta disponibilidad, ya que se pueden realizar consultas en cualquier lugar donde tenga acceso al servidor y a cualquier hora.

1.1 Arquitectura de Software

Una definición reconocida es la de Paul Clements: La AS es, a grandes rasgos, una vista del sistema que incluye los componentes principales del mismo, la conducta de esos componentes según se percibe desde el resto del sistema y las formas en que los componentes interactúan y se coordinan para alcanzar la misión del sistema. La vista arquitectónica es una vista abstracta, aportando el más alto nivel de comprensión y la supresión o diferimiento del detalle inherente a la mayor parte de las abstracciones. (1)

En una definición tal vez demasiado amplia, David Garlan establece que la AS constituye un puente entre el requerimiento y el código, ocupando el lugar que en los gráficos antiguos se reservaba para el diseño. (2)

La definición de AS que está acorde con los objetivos de la herramienta es la brindada por el documento IEEE Std 1471-2000: “La AS es la organización fundamental de un sistema encarnada en sus componentes, las relaciones entre ellos y el ambiente y los principios que orientan su diseño y evolución” adoptada también por Microsoft. (3)

De manera general la AS es la visión arquitectónica que utiliza el diseño como herramienta para simular lo que se ha de implementar en el software, conocer las interacciones entre los distintos componentes presentes en el sistema y constituye una abstracción de uno o más sistemas.

1.2 Evaluación de la Arquitectura de Software

Una evaluación es un estudio de factibilidad que pretende detectar posibles riesgos, así como buscar recomendaciones para contenerlos. La diferencia entre evaluar y verificar es que la evaluación se realiza antes de la implementación de la solución. La verificación es con el producto ya construido. (4)

El objetivo de evaluar una arquitectura es saber si puede habilitar los requerimientos, atributos de calidad y restricciones para asegurar que el sistema a ser construido cumple con las necesidades de los clientes. Durante el desarrollo del proyecto de software la labor más complicada es transformar las especificaciones de los requerimientos de calidad en una arquitectura de software.

Una AS se evalúa cuando ya está definida y no ha comenzado la implementación, pero una evaluación puede ser realizada en cualquier momento. Existen dos tipos de evaluación: temprana y tardía y estas se realizan de manera cualitativa o cuantitativa.

Para realizar una evaluación temprana de una AS no es necesario que esté definida completamente, sin embargo esta es válida para futuros componentes que se adhieran a la arquitectura siempre que cumplan los mismos patrones que hayan sido evaluados. Puede ser aplicada a solo una arquitectura o a un grupo de ellas, donde la misma revela las debilidades y fortalezas de cada una, por lo general es utilizada para examinar las decisiones arquitectónicas ya tomadas y decidir entre las opciones que están pendientes.

La evaluación tardía se realiza al terminar la arquitectura y cuando la implementación está completa. Se considera muy útil la evaluación del sistema en este punto, porque puede observarse el cumplimiento de los atributos de calidad asociados al sistema y cómo será su comportamiento general.

Para la realización de la herramienta se utilizará la evaluación temprana porque permite identificar las debilidades en una arquitectura definida, haciendo posible llevar a cabo cambios con el objetivo de minimizar las mismas. Esta evaluación ayuda además a determinar si la arquitectura escogida es conveniente y en caso de no serlo, es desestimada antes de comenzar la implementación, lo que

contribuye a la disminución de los costos y mayor aprovechamiento del tiempo destinado al desarrollo del proyecto.

1.2.1 Métodos para la evaluación de la Arquitectura de Software

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. A continuación se realiza una descripción de los elementos fundamentales de algunos métodos de evaluación temprana como: el Método de Análisis de Arquitecturas de Software (Software Architecture Analysis Method, SAAM), el método de Revisión de Diseño Activo Intermedio (Active Intermediate Designs Review, ARID) y el Método de Análisis de Acuerdos de la Arquitectura (Architecture Trade-off Analysis Method, ATAM).

SAAM

Según Kazman, el Método de Análisis de Arquitecturas de Software (Software Architecture Analysis Method, SAAM) es el primero que fue ampliamente promulgado y documentado.

El método de evaluación SAAM se enfoca en la enumeración de un conjunto de escenarios que representan los cambios probables a los que estará sometido el sistema en el futuro. (5) SAAM constituye un método para la evaluación temprana y es aconsejable utilizarlo cuando el atributo de calidad modificabilidad es el de mayor interés.

Para la evaluación del método SAAM se proponen 6 pasos:

1. Desarrollo de escenarios.

Un escenario es una breve descripción de usos anticipados o deseados del sistema. De igual forma, estos pueden incluir cambios a los que puede estar expuesto el sistema en el futuro.

2. Descripción de la arquitectura.

La arquitectura debe ser descrita haciendo uso de alguna notación arquitectónica que sea común a todas las partes involucradas en el análisis. Deben incluirse los componentes de datos y conexiones relevantes, así como la descripción del comportamiento general del sistema.

El desarrollo de escenarios y la descripción de la arquitectura son usualmente llevados a cabo de forma intercalada, o a través de varias iteraciones.

3. Clasificación y asignación de prioridad de los escenarios.

La clasificación de los escenarios puede hacerse en dos clases: directos e indirectos. Un escenario directo es el que puede satisfacerse sin la necesidad de modificaciones en la arquitectura.

Un escenario indirecto es aquel que requiere modificaciones en la arquitectura para poder satisfacerse. Los escenarios indirectos son de especial interés para SAAM, pues son los que permiten medir el grado en el que una arquitectura puede ajustarse a los cambios de evolución que son importantes para los involucrados en el desarrollo.

4. Evaluación individual de los escenarios indirectos.

Para cada escenario indirecto, se listan los cambios necesarios sobre la arquitectura, y se calcula su costo. Una modificación sobre la arquitectura significa que debe introducirse un nuevo componente o conector, o que alguno de los existentes requiere cambios en su especificación.

5. Evaluación de la interacción entre escenarios.

Cuando dos o más escenarios indirectos proponen cambios sobre un mismo componente, se dice que interactúan sobre ese componente. Es necesario evaluar este hecho, puesto que la interacción de componentes semánticamente no relacionados revela que los componentes de la arquitectura efectúan funciones semánticamente distintas. De forma similar, puede verificarse si la arquitectura se encuentra documentada a un nivel correcto de descomposición estructural.

6. Creación de la evaluación global.

Debe asignársele un peso a cada escenario, en términos de su importancia relativa al éxito del sistema. Esta asignación de peso suele hacerse con base en las metas del negocio que cada escenario soporta. En el caso de la evaluación de múltiples arquitecturas, la asignación de pesos puede ser utilizada para la determinación de una escala general. (6)

Después de haber sido evaluada la arquitectura se obtiene una proyección sobre el diseño de los escenarios que representan los cambios posibles ante los que puede estar expuesto el sistema. Además del entendimiento de la funcionalidad del sistema, e incluso una comparación de múltiples arquitecturas con respecto al nivel de funcionalidad que cada una soporta sin modificación.

Este método a pesar de ser útil para evaluar de forma rápida distintos atributos de calidad, centra su evaluación en la modificabilidad, siendo esta su principal desventaja.

ARID

De acuerdo con Kazman el método Revisión de Diseño Activo Intermedio (Active Intermediate Designs Review, ARID) es conveniente para realizar la evaluación de diseños parciales en las etapas tempranas del desarrollo. En ocasiones, es necesario saber si un diseño propuesto es conveniente, desde el punto de vista de otras partes de la arquitectura. Según los autores, ARID es un híbrido entre Active Design Review (ADR) y Architecture Trade-off Analysis Method (ATAM). (7)

El método ARID propone pasos para su evaluación tales como:

Fase 1: Actividades Previas

1. Identificación de los encargados de la revisión.

Los encargados de la revisión son los ingenieros de software que se espera que usen el diseño y todos los involucrados en el diseño. En este punto, converge el concepto de encargado de revisión de ADR e involucrado del ATAM.

2. Preparar el informe de diseño.

El diseñador prepara un informe que explica el diseño. Se incluyen ejemplos del uso del mismo para la resolución de problemas reales. Esto permite al facilitador anticipar el tipo de preguntas posibles, así como identificar áreas en las que la presentación puede ser mejorada.

3. Preparar los escenarios base.

El diseñador y el facilitador preparan un conjunto de escenarios base. De forma similar a los escenarios del ATAM y el SAAM, se diseñan para ilustrar el concepto de escenario, que pueden o no ser utilizados para efectos de la evaluación.

4. Preparar los materiales.

Se reproducen los materiales preparados para ser presentados en la segunda fase. Se establece la reunión y los involucrados son invitados.

Fase 2: Revisión

5. Presentación del ARID.

Se explica los pasos del ARID a los participantes.

6. Presentación del diseño.

El líder del equipo de diseño realiza una presentación, con ejemplos incluidos. Se propone evitar preguntas que conciernen a la implementación o argumentación, así como alternativas de diseño. El objetivo es verificar que el diseño es conveniente.

7. Lluvia de ideas y establecimiento de prioridad de escenarios.

Se establece una sesión para la lluvia de ideas sobre los escenarios y el establecimiento de prioridad de escenarios. Los involucrados proponen escenarios a ser usados en el diseño para resolver problemas que esperan encontrar. Luego los escenarios son sometidos a votación y se utilizan los que resultan ganadores para hacer pruebas sobre el diseño.

8. Aplicación de los escenarios.

Comenzando con el escenario que contó con más votos, el facilitador solicita el pseudo-código que utiliza el diseño para proveer el servicio y el diseñador no debe ayudar en esta tarea. Este paso continúa hasta que ocurra alguno de los siguientes eventos:

- Se agota el tiempo destinado a la revisión.
- Se han estudiado los escenarios de más alta prioridad.
- El grupo se siente satisfecho con la conclusión alcanzada.

Puede suceder que el diseño presentado sea conveniente con la exitosa aplicación de los escenarios, o por el contrario, no conveniente cuando el grupo encuentra problemas o deficiencias.

9. Resumen

Al final, el facilitador recuenta la lista de puntos tratados, pide opiniones de los participantes sobre la eficiencia del ejercicio de revisión y agradece por su participación.

Aunque ARID es un híbrido tiene como desventaja que evalúa solamente la factibilidad de la arquitectura.

ATAM

Según Kazman, el Método de Análisis de Acuerdos de Arquitectura (Architecture Trade-off Analysis Method, ATAM) está inspirado en tres áreas distintas: los estilos arquitectónicos, el análisis de atributos de calidad y el método de evaluación SAAM, explicado anteriormente. 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 entre ellos; esto es, los tipos de acuerdos que se establecen entre ellos. (8)

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, entre otros. (9)

ATAM para su evaluación utiliza la técnica de:

Evaluación basada en escenarios

Un escenario es una breve descripción de la interacción de alguno de los involucrados en el desarrollo del sistema con este, que permite concretar y entender los atributos de calidad. Estos están constituidos en 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 y pruebas, cambios en el sistema y configuración. El contexto describe qué sucede en el sistema al momento del estímulo. La respuesta define, a través de la arquitectura, cómo debería responder el sistema ante el estímulo y este último elemento permite establecer cuál es el atributo de calidad asociado.

La técnica basada en escenarios está fundamentada por dos instrumentos de evaluación relevantes:

Árbol de utilidad (Utility Tree)

Un árbol de utilidad es un esquema en forma de árbol que presenta los atributos de calidad de un sistema de software, refinados hasta el establecimiento de escenarios que especifican con suficiente detalle el nivel de prioridad de cada uno.

La finalidad de su uso es la identificación de los atributos de calidad más significativos para un proyecto específico. No existe un conjunto preestablecido de estos, sino los que se definen al momento de la construcción del árbol por los involucrados en el desarrollo del sistema. El árbol de utilidad contiene como nodo raíz la utilidad general del sistema y los atributos de calidad asociados al mismo que conforman el segundo nivel del árbol se refinan hasta la obtención de un escenario lo suficientemente concreto para ser analizado y otorgarle una prioridad.

Perfiles (Profiles)

Un perfil es un conjunto de escenarios, generalmente con alguna importancia relativa asociada a cada uno de ellos. Su uso permite hacer especificaciones más precisas del requerimiento para un atributo de calidad. Los perfiles tienen asociados dos formas de especificación: completos y seleccionados.

Los perfiles completos definen todos los escenarios relevantes como parte del perfil. Esto permite al ingeniero de software realizar un análisis de la arquitectura para el atributo de calidad estudiado de una manera completa, puesto que incluye todos los posibles casos. Su uso se reduce a sistemas relativamente pequeños y sólo es posible predecir conjuntos de escenarios completos para algunos atributos de calidad. (10)

Los perfiles seleccionados se asemejan a la selección de muestras sobre una población en los experimentos estadísticos. Se toma un conjunto de escenarios de forma aleatoria, de acuerdo a algunos requerimientos. La aleatoriedad no es totalmente cierta por limitaciones prácticas, por lo que se fuerza la realización de una selección estructurada de elementos para el conjunto de muestra. Si bien es informal, permite hacer proposiciones científicamente válidas. (11)

La evaluación del método es realizada en etapas tempranas del desarrollo y es más profundo para evaluar aspectos más relacionados con la arquitectura, como el performance o la confiabilidad.

ATAM tiene establecido 9 pasos, estos agrupados en 4 fases:

Fase 1: Presentación

1. Presentación del ATAM.

El líder de evaluación describe el método a los participantes, trata de establecer las expectativas y responde las preguntas propuestas.

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.

3. Presentación de la arquitectura.

El arquitecto describe la arquitectura, enfocándose en cómo esta cumple con los objetivos del negocio.

Fase 2: Investigación y análisis

4. Identificación de los enfoques arquitectónicos.

Estos elementos son detectados, pero no analizados.

5. Generación del Árbol de utilidad.

Se muestran los atributos de calidad que engloban la utilidad del sistema (funcionalidad, eficiencia, portabilidad, usabilidad, confiabilidad y mantenibilidad), especificados en forma de escenarios. Se anotan los estímulos y respuestas, así como se establece la prioridad entre ellos.

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. En este paso se identifican riesgos arquitectónicos, puntos de sensibilidad y puntos de balance.

Fase 3: Pruebas

7. Lluvia de ideas y establecimiento de prioridad de escenarios.

Con la colaboración de todos los involucrados, se complementa el conjunto de escenarios.

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

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 como el documento de propuestas arquitectónicas, el conjunto de escenarios priorizados, el árbol de utilidad, los riesgos descubiertos, los no riesgos documentados y los puntos sensibles y puntos de intercambio encontrados.

Método de evaluación a utilizar

Se consideró optar por el método de evaluación ATAM, que utiliza la técnica basada en escenario y como instrumento de evaluación el árbol de utilidad, ya 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 entre ellos. Además el método se concentra en la identificación de los estilos o enfoques arquitectónicos utilizados, elementos que 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.

Para el desarrollo de la herramienta se tendrá como punto de partida el paso 5 del método de evaluación seleccionado porque los pasos anteriores serán realizados por el grupo de arquitectos antes de interactuar con el sistema.

Herramienta existente para la evaluación

ATAM Assistant

El Asistente de ATAM (ATAM Assistant) es una herramienta, desarrollada en la Universidad del Estado de California en los Estados Unidos, que semi-automatiza varias tareas implicadas en el proceso de evaluación. El objetivo de la herramienta es una sola aplicación para gestionar, visualizar e informar sobre todos los artefactos generados durante este proceso. Sin embargo al no ser una

herramienta de software libre, conlleva al pago de su licencia para poder utilizarla, además es propiedad de la universidad y para consultar la documentación es necesario abonar un crédito para pertenecer a la comunidad científica reconocida por esta universidad; razón que imposibilita realizar un estudio en profundidad.

1.3 Metodologías de desarrollo

Las metodologías de desarrollo están basadas en procedimientos, técnicas y documentación que son aplicadas a la Ingeniería de Software, sirven de ayuda para el desarrollo de productos software y son usadas para estructurar, planificar y controlar el proceso de desarrollo en sistemas de información.

Rational Unified Process (RUP)

El Proceso Unificado RUP, constituye la metodología estándar más utilizada para el análisis, diseño, implementación y documentación de sistemas orientados a objetos. Proporciona una aproximación disciplinada a la asignación de tareas y responsabilidades. RUP actúa como modelo y puede ser adaptado y extendido. (12) El proceso unificado conocido como RUP, es un modelo de software que permite el desarrollo de software a gran escala, mediante un proceso continuo de pruebas y retroalimentación, garantizando el cumplimiento de ciertos estándares de calidad. (13)

El proceso de desarrollo constituye un marco metodológico que define en términos de metas estratégicas, objetivos, actividades y artefactos (documentación) requerido en cada fase de desarrollo. Esto permite enfocar esfuerzo de los recursos humanos en términos de habilidades, competencias y capacidades a asumir roles específicos con responsabilidades bien definidas. (14) RUP define quien, cómo, cuándo y qué debe hacerse en un proyecto.

El ciclo de vida de RUP está estructurado en 4 fases:

- Fase de concepción o inicio.
- Fase de elaboración.
- Fase de construcción.
- Fase de transición.

Extreme Programming (XP)

Es una metodología ágil de desarrollo de software que tiene como objetivo fortalecer las relaciones interpersonales como eslabón para el éxito en el mismo, incentivar el trabajo en equipo y velar por el aprendizaje de desarrolladores para propiciar un buen clima de trabajo.

El ciclo de vida de un proyecto XP se puede separar en fases:

- Fase de exploración.
- Fase de planificación.
- Fase de iteraciones.
- Fase de puesta en producción.

Proceso Unificado Abierto (Open Unified Process, OpenUP)

OpenUP es un proceso de desarrollo unificado que está basado en Rational Unified Process (RUP). Mantiene las mismas características de RUP pues está dirigido por casos de uso, centrado en la arquitectura y además es iterativo e incremental. El ciclo de vida de un proyecto según la metodología OpenUP se divide en 4 fases fundamentales:

Concepción: En esta fase se pretende determinar los objetivos y establecer el alcance del proyecto.

Elaboración: El propósito de esta fase es establecer la línea base de la arquitectura del sistema y proporcionar una base estable para el desarrollo de la siguiente fase.

Construcción: Esta fase tiene como propósito completar el desarrollo del sistema basado en la arquitectura definida, enfocándose en el diseño, implementación y prueba de las funcionalidades para desarrollar un sistema completo.

Transición: En esta fase se asegura que el software esté listo para entregarse a los usuarios.

OpenUP propone 6 flujos de trabajo:

Requerimientos: Se realizan entrevistas con el cliente para comprender el problema a resolver y se definen los requerimientos.

Diseño: Se realiza el diseño de los requisitos que serán después implementados.

Implementación: Se realiza la implementación del sistema basándose en el diseño realizado.

Prueba: Busca los defectos a lo largo del ciclo de vida.

Gestión del Proyecto: Involucra actividades con las que se busca producir un producto que satisfaga las necesidades de los clientes.

Gestión de Configuración y Cambios: Describe cómo controlar los elementos producidos por todos los integrantes del equipo de proyecto en cuanto a utilización y actualización, control de versiones.

OpenUP es un proceso ágil de desarrollo de software que incluye solo contenido fundamental. Por lo tanto, no proporciona orientación sobre muchos temas que los proyectos pueden tratar, tales como las situaciones contractuales, seguridad o aplicaciones de misión crítica, la tecnología de orientación específica. Sin embargo es completa en el sentido de que puede manifestarse como un proceso para construir un sistema. Para atender las necesidades que no están cubiertos en su contenido, OpenUP es extensible a utilizarse como base sobre la que es posible agregar o adaptar contenidos a un proceso, según sea necesario.

OpenUP como parte de las metodologías ágiles más reconocidas está destinado a conseguir en un equipo la comunicación entre los integrantes del mismo para promover una comprensión compartida del proyecto. Los métodos ágiles han llamado la atención hacia la importancia de comprender la coordinación, beneficiando a las partes interesadas sobre resultados productivos y la formalidad. OpenUP tiene las características esenciales de un proceso unificado de apoyo que se aplica iterativo y con enfoques graduales dentro de un ciclo de vida estructurado probado. Es basado en casos de uso y escenarios, la gestión del riesgo y tiene un enfoque centrado en la arquitectura para guiar el desarrollo.

Metodología de desarrollo a utilizar

Como metodología de desarrollo a utilizar en el proceso de construcción de la herramienta se escogió OpenUP por ser una metodología ágil usada por los desarrolladores de alto nivel por sus importantes características administrativas, al mismo tiempo es ligero y proporciona una comprensión detallada del proyecto favoreciendo a clientes y desarrolladores. Preserva la esencia del Proceso Unificado: desarrollo iterativo e incremental, dirigido por casos de uso y centrado en la arquitectura temprana para reducir al mínimo los riesgos y organizar el desarrollo. Además está pensada para proyectos pequeños.

1.4 Lenguaje de modelado

Un lenguaje de modelado es un conjunto de símbolos y reglas que están estandarizados y se utilizan para modelar parte de un diseño de software orientado a objetos. Comúnmente son utilizados en combinación con una metodología de desarrollo de software para llegar de una especificación inicial a la implementación. (15)

UML

Para el desarrollo del sistema se hará uso del Lenguaje de Modelado Unificado (UML) por ser el estándar más utilizado para especificar y documentar cualquier sistema de forma precisa. El mismo está diseñado para ser un lenguaje de modelado de propósito general proporcionando una gran flexibilidad y expresividad a la hora de modelar sistemas, puede ser utilizado para especificar la mayoría de los sistemas basados en objetos o en componentes y para modelar aplicaciones de muy diversos dominios de aplicación y plataformas de objetos distribuidos. UML es un lenguaje gráfico para especificar, construir y documentar los artefactos que modelan un sistema.

UML utiliza los diagramas gráficos para obtener estos distintos puntos de vista de un sistema:

- Diagramas de Implementación
- Diagramas de Comportamiento o Interacción
- Diagramas de Casos de uso
- Diagramas de Clases

1.5 Herramienta de modelado

Las herramientas CASE (Herramienta de Ingeniería de Software Asistida por Computación) proporcionan ayuda automatizada a los ingenieros, analistas y desarrolladores durante el ciclo de vida del desarrollo de un software, garantizando una mejora en la calidad del mismo. Facilitan la realización de prototipos, el desarrollo conjunto de aplicaciones y mejoran y estandarizan la documentación. Además aumentan la portabilidad de las aplicaciones, facilitan la reutilización de componentes y posibilitan un desarrollo y un refinamiento visual de las aplicaciones, mediante la utilización de gráficos.

Visual Paradigm 6.4

Visual Paradigm es una herramienta UML profesional que soporta el ciclo de vida del desarrollo de software: análisis y diseño orientados a objetos, construcción, pruebas y despliegue. Es una herramienta multiplataforma que se integra con varios IDEs (Ambiente Integrado de Desarrollo) y soporta múltiples usuarios trabajando sobre un mismo proyecto. Además importa y exporta diagramas en XML como imágenes (ya sea con extensiones jpg o png). Permite dibujar todos los tipos de diagramas de clases, código inverso, generar código desde diagramas y generar documentación. El software de modelado UML ayuda a la construcción de aplicaciones de calidad, mejores y a un menor coste.

Rational Rose

La herramienta CASE Rational Rose propone la utilización de cuatro tipos de modelo para realizar un diseño del sistema, utilizando una vista estática y otra dinámica de los modelos del sistema, uno lógico y otro físico. Permite crear y refinar estas vistas creando de esta forma un modelo completo que representa el dominio del problema y el sistema de software.

Características de la herramienta:

- Es compatible con el lenguaje UML (Unified Modeling Language) y es uno de los productos más completos de la familia Rational Rose.
- Soporta patrones de Análisis, ANSI C++, Rose J y Visual C++, Enterprise JavaBeans 2.0, e ingeniería directa e inversa para algunas de las construcciones más comunes de Java 1.5.
- Es capaz de analizar la calidad del código y de generar código gracias a las capacidades de sincronización configurable entre el modelo y el código, además de una gestión más detallada y el uso de modelos con la función de componentes de modelos controlables por separado.
- Permite el modelado UML para diseñar bases de datos, con la posibilidad de representar la integración de los requisitos de datos y aplicaciones mediante diseños lógicos y físicos.
- Sistemas operativos admitidos: Windows.

Herramienta de modelado a utilizar

La herramienta de modelado escogida fue Visual Paradigm porque constituye una herramienta profesional que soporta todo el ciclo de vida del desarrollo del software. Es capaz de generar código fuente en PHP, así como crear esquemas de clases, definiciones de bases de datos. La ventaja

principal de la misma radica en que es una herramienta multiplataforma y libre, por lo que no es necesario el pago de una licencia por su utilización.

1.6 IDE de desarrollo

Un entorno de desarrollo integrado (Integrated Development Environment, IDE), es un programa compuesto por un conjunto de herramientas para un programador. Un IDE es un entorno de programación que ha sido empaquetado como un programa de aplicación, es decir, consiste en un editor de código, un compilador, un depurador y un Constructor de Interfaz Gráfica (GUI). (16)

NetBeans 7.1

NetBeans IDE es un entorno de desarrollo, una herramienta para que los programadores puedan escribir, compilar, depurar y ejecutar programas. Está escrito en Java, pero puede servir para cualquier otro lenguaje de programación. Existe además un número importante de módulos para extender el NetBeans IDE. Es un producto libre y gratuito sin restricciones de uso. (17)

Es un reconocido entorno de desarrollo integrado disponible para Windows, Mac, Linux y Solaris. El proyecto NetBeans está formado por un IDE de código abierto y una plataforma de aplicación que permite a los desarrolladores crear con rapidez aplicaciones web, empresariales, de escritorio y móviles utilizando la plataforma Java, así como JavaFX, PHP, JavaScript y Ajax, Ruby y Ruby on Rails, Groovy and Grails y C/C++.

Zend Studio

Zend Studio consta de dos partes en las que se dividen las funcionalidades de parte del cliente y las del servidor. Las dos partes se instalan por separado, la del cliente contiene la interfaz de edición y la ayuda. Permite además hacer depuraciones simples de scripts, aunque para disfrutar de toda la potencia de la herramienta de depuración habrá que disponer de la parte del servidor, que instala Apache y el módulo PHP o, en caso de que estén instalados, los configura para trabajar juntos en depuración. También fue diseñado para usarse con el lenguaje PHP; sin embargo ofrece soporte básico para otros lenguajes Web, como HTML, JavaScript y XML.

IDE Seleccionado

Después de haber realizado esta comparación entre los IDE de desarrollo se puede apreciar que ambos son potentes en el desarrollo de aplicaciones web, sin embargo el ZendStudio tiene como desventaja que requiere de licencia de pago, por lo que se decidió utilizar NetBeans ya que es un reconocido entorno de desarrollo integrado disponible en múltiples plataformas, permite desarrollar aplicaciones web con rapidez, además es un producto libre y gratuito sin restricciones de uso.

1.7 Lenguaje de programación

Los lenguajes de programación son herramientas que permiten crear programas y software. Es un conjunto de símbolos y reglas sintácticas y semánticas que definen su estructura y el significado de sus elementos y expresiones. Además son utilizados para controlar el comportamiento físico y lógico de una máquina.

Como lenguajes a utilizar fueron escogidos PHP 5.3 y JavaScript para ser ejecutados en el servidor y el cliente respectivamente.

JavaScript

JavaScript es un lenguaje interpretado que no requiere compilación. El navegador del usuario se encarga de interpretar las sentencias JavaScript contenidas en una página HTML y ejecutarlas adecuadamente. Es un lenguaje orientado a eventos. Cuando un usuario da click sobre un enlace o mueve el puntero sobre una imagen se produce un evento. Mediante JavaScript se pueden desarrollar scripts que ejecuten acciones en respuesta a estos eventos, además es un lenguaje orientado a objetos. El modelo de objetos de este lenguaje está reducido y simplificado, pero incluye los elementos necesarios para que los scripts puedan acceder a la información de una página y puedan actuar sobre la interfaz del navegador.

PHP 5.3

PHP (Hipertexto Pre-Procesado) es un lenguaje de programación interpretado que posibilita la generación dinámica de contenidos en un servidor web. Se destacan entre sus características principales su alto rendimiento, potencia, facilidad de aprendizaje y los pocos recursos que consume.
(18)

Es un lenguaje de programación del lado del servidor gratuito e independiente de la plataforma, rápido, con una gran librería de funciones y mucha documentación.

Este lenguaje posee características que lo convierten en una potente herramienta tales como:

- Es un lenguaje multiplataforma.
- Soporte sólido para Programación Orientada a Objetos.
- Incorpora bibliotecas que contienen funciones integradas para realizar útiles tareas relacionadas con la Web.
- Es un software de código abierto.

PHP 5.3 se encuentra ampliamente distribuido, está perfectamente soportado por una comunidad de desarrolladores. Como producto de código abierto goza de la ayuda de un grupo de programadores, permitiendo que los fallos del funcionamiento se encuentren y se reparen rápidamente. Además el código se pone al día continuamente con mejoras y extensiones de lenguaje para ampliar las capacidades de PHP y permite codificar la lógica de negocio del sistema que se encuentra del lado del servidor.

1.8 Framework de desarrollo

Un framework es una estructura de soporte en la cual un proyecto de software puede ser organizado y desarrollado. Puede incluir soporte de programas, bibliotecas y un lenguaje interpretado entre otros software para ayudar a desarrollar y unir los diferentes componentes de un proyecto. Proporciona estructura al código fuente, forzando al desarrollador a crear código más legible y más fácil de mantener. Además de facilitar la programación de aplicaciones, ya que encapsula operaciones complejas en instrucciones sencillas.

Symfony

Symfony es un marco de desarrollo (framework) para PHP, libre, desarrollado completamente con PHP5 y diseñado para optimizar el desarrollo de las aplicaciones web. Para empezar, 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. (19)

Zend Framework

Zend Framework es un marco de desarrollo (framework) para PHP, libre, orientado a objetos que brinda una arquitectura flexible. Provee un núcleo para el manejo del patrón de diseño MVC y posee algunas librerías (RSS, PDF) incorporadas que lo hacen perder acoplamiento y ganar flexibilidad. (20) Su marco incluye objetos de la mayoría de las bases de datos lo que simplifica las consultas a su base de datos, sin la necesidad de realizar consultas SQL. Posee un avanzado soporte para internacionalización y cuenta con una completa documentación y test de alta calidad. Además el Zend Framework posee clases robustas para filtrado de entrada y autenticación. (21)

Framework a utilizar

Como framework de desarrollo fue escogido Symfony porque el mismo está desarrollado completamente con PHP5 y diseñado para optimizar el desarrollo de aplicaciones web. Soporta el Patrón de Diseño ORM (Mapeo Objeto-Relacional) e incorpora Motor de Plantillas (Layout), además incorpora un módulo para el manejo de autenticación de usuarios y su implementación es sencilla.

1.9 Gestor de base de datos

Los gestores de bases de datos son un tipo de software muy específico, dedicado a servir de interfaz entre la base de datos, el usuario y las aplicaciones que la utilizan. Controlan el almacenamiento de datos redundantes, los datos resultan independientes de los programas que los usan, se almacenan las relaciones entre los datos junto con estos y se puede acceder a los datos de diversas formas.

PostgreSQL 8.4

PostgreSQL es un sistema de gestión de bases de datos objeto-relacional (ORDBMS) basado en el proyecto POSTGRES. Incluye características de la orientación a objetos, como puede ser la herencia, tipos de datos, funciones, restricciones, disparadores, reglas e integridad transaccional. A pesar de esto, PostgreSQL no es un sistema de gestión de bases de datos puramente orientado a objetos.

Características del sistema de gestión de base de datos:

- Soporta distintos tipos de datos, dentro de los que se incluyen: datos de tipo fecha, monetarios, elementos gráficos, datos sobre redes, cadenas de bits y permite además la creación de tipos propios.
- Incorpora una estructura de datos array.

- Incluye herencia entre tablas (aunque no entre objetos, ya que no existen), por lo que a este gestor de bases de datos se le incluye entre los gestores objeto-relacionales.
- Permite la gestión de diferentes usuarios, como también los permisos asignados a cada uno de ellos.

PostgreSQL ofrece las siguientes posibilidades:

- Instalación ilimitada: con PostgreSQL, nadie puede ser demandado por violar acuerdos de licencia, puesto que no hay un costo asociado a la licencia del software.
- Soporte: existe una gran comunidad de profesionales y empresas que ofrecen soporte a PostgreSQL, de la cual el Grupo Global de Desarrollo de PostgreSQL puede obtener beneficios y contribuir en su mejoramiento.
- Estabilidad y Confiabilidad Legendarias: es extremadamente común que compañías reporten que PostgreSQL nunca ha presentado caídas en varios años de operación de alta actividad.
- Multiplataforma: se encuentra disponible para Linux y Windows.
- Herramientas gráficas de diseño y administración de bases de datos: existen varias herramientas gráficas de alta calidad para administrar las bases de datos (pgAdmin, pgAccess) y para hacer diseño de bases de datos.(22)

MySQL

MySQL (a veces denominado “monitor de terminal” o solamente “monitor”) es un programa interactivo que le permite conectarse a un servidor de bases de datos MySQL, ejecutar consultas, y ver los resultados. MySQL puede usarse también en modo por lotes: se colocan las consultas en un archivo previamente armado, y se le dice a MySQL que ejecute el contenido del archivo. (23)

El software MySQL proporciona un servidor de base de datos SQL (Structured Query Language) muy rápido, multi-threaded, multi usuario y robusto. El servidor MySQL está diseñado para entornos de producción críticos, con alta carga de trabajo así como para integrarse en software para ser distribuido. MySQL es una marca registrada de MySQL AB. (24)

Gestor de base de datos a utilizar

Después de haber realizado las comparaciones entre los gestores de base de datos se decidió utilizar para el desarrollo de la herramienta PostgreSQL 8.4 por ser de código abierto, libre, lo que garantiza que no sea necesario el pago de una licencia por su utilización. Posee además soporte

para distintos tipos de datos, incorpora estructuras de datos array entre otras características. Además implementa el uso de subconsultas y transacciones, haciendo su funcionamiento mucho más eficaz.

1.10 Herramienta para realizar Pruebas de Software

QALoad

Puede emular la carga generada por cientos o miles de usuarios en la aplicación sin requerir la participación de los usuarios finales en sus equipos. Ofrece las estadísticas de rendimiento en tiempo real, y permite a los ingenieros de prueba insertar puntos de control dentro de los scripts para identificar y revisar áreas específicas del rendimiento del sistema. Aunque pudiera ser una herramienta candidata a seleccionar, constituye un software privativo, por tanto se precisa pagar para su adquisición y por consiguiente también su documentación.

LoadRunner

Herramienta de pruebas de carga más escalable que permite simular la actividad de miles de usuarios con los mínimos recursos de hardware. A pesar de tener todas esas características favorables, su adquisición es solo gratuita por 15 días y después de ser obtenida se debe pagar por su uso, de igual manera la documentación está sujeta a las mismas restricciones.

JMeter

Herramienta de software libre que ofrece la posibilidad de realizar pruebas de carga sobre diferentes aspectos de una aplicación web, inicialmente diseñada para pruebas de estrés en las mismas. Utilizar JMeter supone una mayor efectividad en el proceso y en la fiabilidad de los resultados. Dispone de varios componentes que facilitan la elaboración de los escenarios de prueba con la ventaja de simular para cada uno de esos escenarios miles de usuarios. De esta manera se verifica el rendimiento del sistema mediante las pruebas de Carga y Estrés. Una ventaja es que permite realizar pruebas de carga sobre cada una de las capas que conforman la aplicación a probar.

Herramienta seleccionada para realizar Prueba de Software

Después del análisis realizado de las herramientas se optó seleccionar a JMeter, por ser una herramienta dinámica y que además de evaluar el rendimiento mediante las pruebas de carga y estrés, utiliza la técnica de Caja Negra, probando todas las funcionalidades del software, y

proporcionando de esta manera una retroalimentación y reforzamiento en las pruebas funcionales ya realizadas con anterioridad. También su desarrollo es en el marco del software libre, política que apoya la universidad y que por sus características existe un por ciento superior de confianza en la utilización de la misma ya que se puede contar con todo el código que genera a la herramienta.

1.11 Conclusiones del Capítulo

En este capítulo se abordó lo referente a la arquitectura de software, su evaluación y la investigación de los métodos de evaluación, mediante la cual se ha seleccionado como mejor método de evaluación para la herramienta ATAM por revelar la forma en que una arquitectura específica satisface ciertos atributos de calidad. Además se estudiaron las tendencias y tecnologías actuales brindando las definiciones necesarias para comprender las herramientas que serán utilizadas en el desarrollo de la aplicación. Con el objetivo de garantizar un desarrollo exitoso del software y con el fin de hacerlo más eficiente se empleará como metodología de desarrollo, OpenUP, por ser una metodología ágil centrada en una arquitectura temprana para reducir al mínimo los riesgos y organizar el desarrollo. Se utilizará como lenguaje de modelado UML y para obtener un mejor diseño se hará uso de la herramienta de modelado Visual Paradigm 6.4. También para lograr una mayor facilidad a la hora de desarrollar la herramienta se utilizará PHP 5.3 como lenguaje de programación, donde el entorno de desarrollo a utilizar será NetBeans 7.1, a través del cual se hará uso del framework Symfony 1.4, además PostgreSQL 8.4 como gestor de base de datos y JMeter como herramienta para evaluar el rendimiento del sistema.

CAPÍTULO 2: ANÁLISIS Y DISEÑO DEL SISTEMA

En este capítulo se identifican los requisitos funcionales y no funcionales que se deben tener en cuenta para la implementación del sistema y se identifican los actores, casos de uso y las relaciones existentes entre ellos.

Breve descripción del sistema

Se propone una herramienta que asista a la evaluación de la arquitectura en el centro DATEC. Dicho sistema permite a los arquitectos trabajar con los escenarios y decisiones arquitectónicas insertadas, así como la visualización de reportes de evaluación, contribuyendo de esta forma a la toma de decisiones arquitectónicas.

2.1 Especificación de los Requisitos del Sistema

Los requisitos de un sistema describen los servicios que ha de ofrecer el sistema y las restricciones asociadas a su funcionamiento, no son más que propiedades o restricciones determinadas de forma precisa que deben satisfacerse. (25)

En este documento el artefacto de especificación de requisitos de software debe comprender una lista definida de los requisitos que debe cumplir la herramienta a desarrollar, donde los requisitos tienen que estar bien claros, completos, concretos y consistentes.

Requisitos Funcionales

Los requisitos funcionales son capacidades o condiciones que el sistema debe cumplir y se identifican a partir del método de evaluación seleccionado. Se mantienen invariables sin importar con qué propiedades o cualidades se relacionen. Además describen su funcionalidad, los servicios que de él se esperan, o los que proveerá y definen las funciones que el sistema será capaz de realizar.

- **RF 1: Autenticar usuario**
- **RF 2: Gestionar usuario**
 - RF 2.1:** Adicionar usuario
 - RF 2.2:** Visualizar usuario
 - RF 2.3:** Listar usuario
 - RF 2.4:** Modificar usuario

- RF 2.5: Eliminar usuario
- **RF 3: Visualizar reporte de evaluación**
- **RF 4: Gestionar escenario**
 - RF 4.1: Adicionar escenario
 - RF 4.2: Visualizar escenario
 - RF 4.3: Listar escenario (General)
 - RF 4.4: Listar escenario (Por Árbol)
 - RF 4.5: Modificar escenario
 - RF 4.6: Eliminar escenario
- **RF 5: Gestionar árbol de utilidad**
 - RF 5.1: Adicionar árbol de utilidad
 - RF 5.2: Visualizar árbol de utilidad
 - RF 5.3: Listar árbol de utilidad
 - RF 5.4: Modificar árbol de utilidad
 - RF 5.5: Eliminar árbol de utilidad
- **RF6: Administrar decisión arquitectónica**
 - RF 6.1: Adicionar decisión arquitectónica
 - RF 6.2: Visualizar decisión arquitectónica
 - RF 6.3: Listar decisión arquitectónica
 - RF 6.4: Modificar decisión arquitectónica
- **RF7: Gestionar decisión-escenario**
 - RF 7.1: Adicionar decisión-escenario
 - RF 7.2: Visualizar decisión-escenario
 - RF 7.3: Listar decisión-escenario
 - RF 7.4: Modificar decisión-escenario
 - RF 7.5: Eliminar decisión-escenario
- **RF8: Graficar reporte de evaluación**
- **RF9: Exportar reporte de evaluación**

Requisitos No Funcionales

Los requisitos no funcionales son propiedades o cualidades que el producto debe tener, que hacen al producto atractivo, usable, rápido o confiable. Normalmente están vinculados a los requisitos funcionales y son fundamentales en el éxito del producto.

Usabilidad

➤ **RNF1: Interacción del usuario con el sistema.**

El tiempo de entrenamiento requerido para los usuarios será aproximadamente 15 días.

La herramienta es 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.

Rendimiento del sistema

➤ **RNF2: Disponibilidad**

El sistema debe poder permanecer online en los horarios de trabajo establecidos por las entidades, excepto cuando sea necesario reiniciarlo o detenerlo por tareas de mantenimiento o cambio de configuración.

➤ **RNF3: Seguridad**

Los usuarios para la entrada al sistema deben ser autenticados, de no ser así no pueden trabajar con el sistema hasta que no sean registrados por el Administrador, esto garantiza la confiabilidad y confidencialidad de los datos a través de la asignación de roles.

Eficiencia

➤ **RNF4: Cantidad de usuarios conectados simultáneamente.**

El sistema debe permitir la concurrencia de al menos 10 usuarios.

Como promedio el sistema debe poseer un tiempo de respuesta de 5 segundos.

Restricciones de diseño

➤ **RNF5: Lenguaje y marco de trabajo para el desarrollo del sistema del lado del servidor.**

El sistema se implementará con el lenguaje de programación PHP versión 5.3. Como framework de desarrollo se usará Symfony 1.4 el cual propone una arquitectura modular en tres capas: el modelo, la vista y el controlador.

➤ **RNF6: Requerimientos de Hardware**

Cliente

- Ordenador con un microprocesador Pentium IV a 1.7 GHz o superior.
- Memoria RAM mínimo 512 MB.

Servidor

- Ordenador con un microprocesador Pentium IV a 3.0 GHz o superior.

- Memoria RAM mínimo 2Gb.
- Disco Duro con 5Gb de capacidad para instalar el sistema.

➤ **RNF7: Requerimientos de Software**

Cliente

- Navegador compatible con CSS 3.

Servidor

- Sistema operativo GNU/Linux Ubuntu 10.10 o versión superior.
- Servidor de aplicaciones Apache, versión 2.
- Lenguaje de programación PHP y librerías, versión 5.3.
- Sistema Gestor de Base de Datos PostgreSQL, versión 8.4 o superior.

2.2 Modelo de casos de uso del sistema

El modelo de casos de uso del sistema está compuesto por actores, casos de uso y la relación entre ellos. Constituye un esquema que recoge las funcionalidades del sistema que se han de automatizar y describen bajo la forma de acciones y reacciones, el comportamiento de un sistema desde el punto de vista del usuario.

Definición de los casos de uso del sistema

Un caso de uso es una descripción de la secuencia de interacciones que se producen entre un actor y el sistema, cuando el actor usa el sistema para llevar a cabo una tarea específica. Expresa una funcionalidad y se representa en el diagrama de casos de uso mediante una elipse con el nombre del caso de uso en su interior reflejando la tarea específica que el actor desea llevar a cabo, además este proporciona uno o más escenarios que indican cómo debería interactuar el sistema con el usuario o con otro sistema para conseguir un objetivo.

Definición de los actores del sistema

Los actores representan un tipo de usuario del sistema. Se entiende como usuario cualquier elemento externo que interactúa con el sistema es decir, una persona (identificada por un rol), un sistema informatizado u organización. Además son generalmente responsables de realizar actividades que serán automatizadas en el sistema.

Tabla 1: Actores del sistema.

Actor	Descripción
Administrador	Es el Jefe de Proyecto encargado de administrar y darles los permisos a los usuarios que interactúan con el sistema.
Arquitecto	Representa al usuario capacitado en la evaluación que va a interactuar en el sistema con las funcionalidades que le corresponde.
Usuario	Rol que representa a los usuarios del sistema que se han autenticado y pueden acceder a los recursos que le son permitidos.

Diagrama de casos de uso del sistema

Los diagramas de casos de uso sirven para especificar la comunicación y el comportamiento de un sistema mediante su interacción con los usuarios y/u otros sistemas, mostrando la relación entre los actores y los casos de uso en un sistema.

Diagrama de caso de uso del sistema

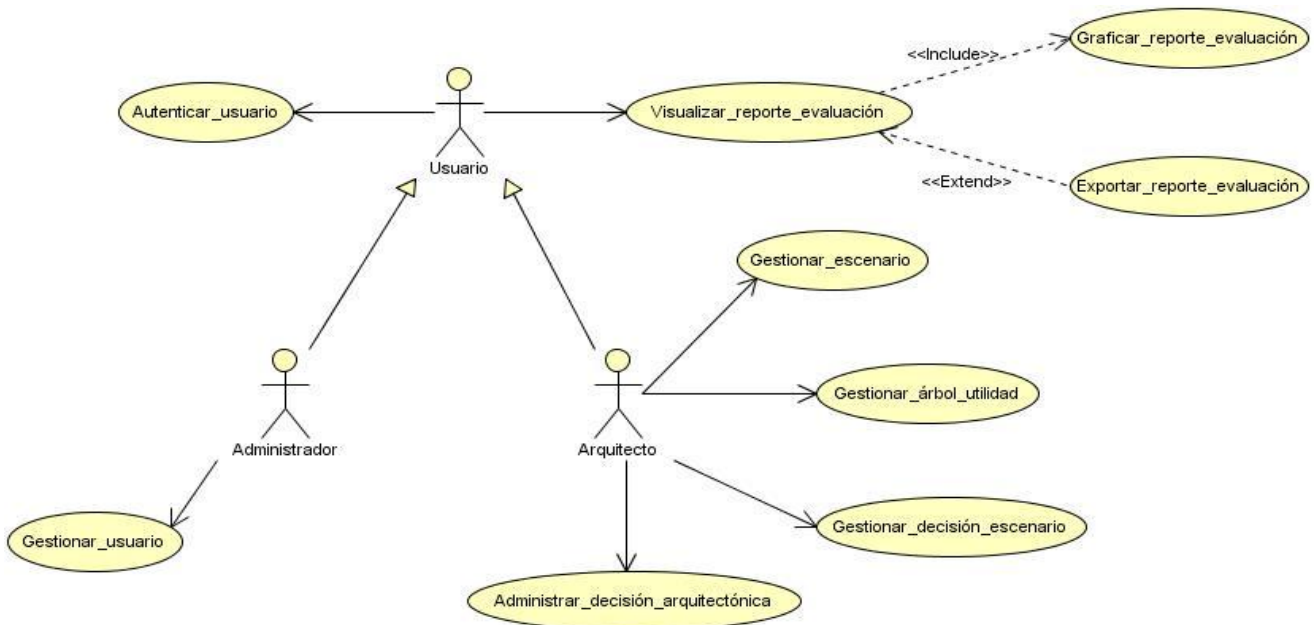


Figura 1: Diagrama de caso de uso del sistema.

Los patrones de casos de uso permiten reflejar con precisión los requerimientos funcionales del sistema, ayudan a describir qué es lo que el sistema debe hacer, es decir, describen el uso del

sistema y cómo este interactúa con los usuarios. Con la utilización de ellos se pueden lograr mejores resultados de forma más rápida.

Existen diversos patrones de casos de uso, entre ellos se utilizó el patrón CRUD Completo, consiste en un caso de uso para gestionar la información (Gestionar_usuario, Gestionar_escenario y Gestionar_árbol_utilidad, Gestionar_decisión_escenario), que permite modelar las diferentes operaciones para gestionar una entidad de información, tales como adicionar, eliminar, modificar y visualizar la información, además se usó el patrón CRUD Parcial, que indica la ausencia de algunas de las cuatro operaciones del CRUD Completo, consiste en un caso de uso para Administrar la información (Administrar_decisión_arquitectónica), que permite modelar las operaciones para administrar una entidad de información, tales como adicionar, modificar y visualizar la información y se utilizó el patrón Múltiples Actores, específicamente la categoría Roles Comunes. Este patrón se empleó debido a que tres actores (Usuario, Administrador y Arquitecto) inicializan un mismo caso de uso, por tal motivo se representa un actor (Usuario), que inicialice el caso de uso común (Autenticar_usuario y Visualizar_reporte_evaluación) y de él heredan dos actores (Administrador y Arquitecto).

Descripción textual de los casos de uso del sistema

Cada caso de uso se detalla mediante una descripción textual que describe la funcionalidad que se construirá en el componente propuesto. A continuación se muestra un ejemplo:

Descripción del caso de uso Gestionar_escenario

Tabla 2: Descripción del caso de uso Gestionar_escenario.

Caso de Usos:	Gestionar_escenario	
Actor:	Arquitecto	
Resumen	En este caso de uso el Arquitecto es el encargado de adicionar, modificar, eliminar, listar los escenarios.	
Precondiciones:	El Arquitecto debe estar autenticado.	
Referencia:	RF4	
Prioridad	Crítico	
Flujo Normal de Eventos		
	Acción del Actor	Respuesta del Sistema
	1. El Arquitecto selecciona del menú	2. El sistema ejecuta una de las siguientes

<p>Gestionar Escenario una de las siguientes opciones:</p> <ul style="list-style-type: none"> • Adicionar • Modificar/Eliminar • Listar. 	<p>secciones:</p> <ul style="list-style-type: none"> • Si seleccionó la opción Adicionar ir a la sección Adicionar escenario. • Si seleccionó la opción Modificar/Eliminar ir a la sección Modificar escenario si va a modificar. • Si seleccionó la opción Modificar/Eliminar ir a la sección Eliminar escenario si va a eliminar. • Si seleccionó la opción Listar ir a la sección Buscar y visualizar escenario.
---	---

Sección “Adicionar escenario” Flujo Normal de Eventos

Acción del Actor	Respuesta del Sistema
1. El Arquitecto escoge la opción de Adicionar.	2. El sistema muestra los árboles existentes para que el Arquitecto escoja a cual le adicionará el escenario.
3. El Arquitecto selecciona el árbol.	4. El sistema muestra una ventana de diálogo con un formulario para que el Arquitecto introduzca los datos del escenario.
5. El Arquitecto introduce los datos del escenario y oprime el botón Guardar.	6. El sistema valida los datos. 7. Adiciona el nuevo escenario en la base de datos. 8. El sistema muestra la lista de escenarios, terminado así el caso de uso.

Sección “Adicionar escenario” Flujo Alterno

Acción del Actor	Respuesta del Sistema
	7.1 El sistema detecta errores en los datos introducidos o la ausencia de ellos e indica al Arquitecto donde existe el error. (Volver al paso 5)

Sección “Modificar escenario” Flujo Normal de Eventos

Acción del Actor	Respuesta del Sistema
1. El Arquitecto escoge la opción de Modificar/Eliminar.	2. El sistema muestra los árboles existentes para que el Arquitecto escoja que escenario modificará.
3. El Arquitecto selecciona el árbol.	4. El sistema muestra una ventana con los escenarios existentes del árbol seleccionado.

5. El Arquitecto selecciona el escenario a modificar.	6. El sistema muestra una ventana de diálogo con el formulario de los datos del escenario para que el Arquitecto los modifique.
7. El Arquitecto modifica los datos del escenario y oprime el botón Guardar.	8. El sistema valida los datos. 9. Modifica los datos del escenario en la base de datos. 10. El sistema muestra el listado de los escenarios, terminando así el caso de uso.
Sección “Modificar escenario” Flujo Alterno	
Acción del Actor	Respuesta del sistema
	9.1 El sistema detecta errores en los datos introducidos o la ausencia de estos y le indica donde existe el error. (Volver al paso 7)
Sección “Eliminar escenario” Flujo Normal de Eventos	
Acción del Actor	Respuesta del sistema
1. El Arquitecto escoge la opción de Modificar/Eliminar.	2. El sistema muestra los árboles existentes para que el Arquitecto escoja de cual eliminará un escenario.
3. El Arquitecto selecciona el árbol.	4. El sistema muestra una ventana con los escenarios existentes del árbol seleccionado.
5. El Arquitecto selecciona el escenario a eliminar.	6. El sistema muestra una ventana con los datos del escenario seleccionado.
7. El Arquitecto oprime el botón Eliminar.	8. El sistema muestra mensaje de confirmación: ¿Está seguro?
9. El Arquitecto oprime el botón Aceptar.	10. El sistema elimina el escenario de la base de datos. 11. El sistema muestra el árbol con los escenarios existentes, terminado así el caso de uso.
Sección “Eliminar escenario” Flujo Alterno	
Acción del Actor	Respuesta del sistema
9.1 El Arquitecto oprime el botón Cancelar.	10.1 El sistema cierra el mensaje de confirmación.
Sección “Buscar y visualizar escenario” Flujo Normal de Eventos	

Acción del Actor	Respuesta del sistema
1. El Arquitecto escoge una opción de Listar General / Listar Por Árbol.	2. El sistema muestra una ventana con los árboles y escenarios asociados a ellos. / El sistema muestra una ventana con los árboles de utilidad existentes.
3. El Arquitecto selecciona el escenario que desea ver. / El Arquitecto selecciona el árbol de utilidad del cual desea ver los escenarios.	4. El sistema busca el escenario en la base de datos. / El sistema muestra una ventana con los escenarios existentes del árbol seleccionado.
5. / El Arquitecto selecciona el escenario que desea ver.	5 / 6. El sistema muestra el escenario con los datos correspondientes a él, terminado así el caso de uso.
Poscondiciones	Queda adicionado, modificado o eliminado un escenario.

2.3 Diagrama conceptual de la herramienta

El diagrama conceptual puede ser adquirido como punto de partida para el diseño del sistema y tiene como propósito contribuir a la comprensión del sistema.

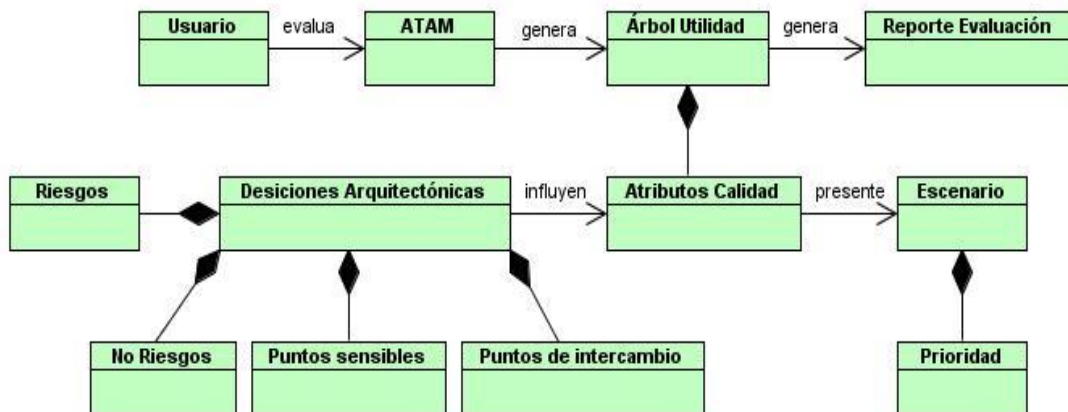


Figura 2: Diagrama conceptual de la herramienta.

Definición de clases del diagrama conceptual

- **Usuario:** Son las personas encargadas de interactuar con el sistema.

- **ATAM:** Método de evaluación 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. Además identifica riesgos, puntos sensibles y puntos de intercambio.
- **Árbol Utilidad:** Proveen un mecanismo para la traducción directa y eficiente de las necesidades del negocio de un proyecto en atributos de calidad. Es un esquema en forma de árbol generado a partir de los atributos de calidad del sistema, el cual indica al equipo evaluador dónde ocupar su tiempo, probar aproximaciones y riesgos arquitectónicos.
- **Atributos Calidad:** Los atributos de calidad son aspectos del sistema que no afectan directamente a la funcionalidad necesitada, sino que definen la calidad y las características que el sistema debe soportar. Engloban la utilidad del sistema, ejemplo de estos atributos son: confidencialidad, eficiencia, portabilidad, modificabilidad, usabilidad y funcionalidad.
- **Prioridad:** Es la que se le otorga a los escenarios de acuerdo con su importancia.
- **Escenarios:** Es una breve descripción de la interacción de algunos de los involucrados en el desarrollo del sistema, consta de tres partes: el estímulo, el contexto y la respuesta.
- **Reporte Evaluación:** Muestra los resultados de la evaluación de arquitectura.
- **Decisiones Arquitectónicas:** Decisiones tomadas por parte del equipo de desarrollo que se ejecuta con el objetivo de mitigar los riesgos en la arquitectura desde el punto de vista de los atributos de calidad analizados, para proceder a la construcción del sistema.
- **Riesgos:** Son decisiones arquitectónicamente importantes que no han sido tomadas o decisiones que han sido tomadas pero las consecuencias no han sido entendidas a plenitud.
- **No Riesgos:** Son buenas decisiones de arquitectura que suelen estar implícitas.
- **Puntos sensibles:** Son las propiedades de los componentes que son críticos para alcanzar un atributo de calidad.
- **Puntos de intercambio o trade-off:** Es descubierto en la arquitectura cuando un parámetro de construcción arquitectural es común para más de un punto de sensibilidad, donde los atributos de calidad medibles son afectados indistintamente por cambios en el parámetro.

2.4 Diseño del sistema

El diseño del sistema define la arquitectura y estructura de hardware, software, componentes, módulos y datos de un sistema de cómputo. Se centra en la transformación de los requisitos de datos y la arquitectura del software. Además de ser una presentación detallada del informe de terminación del análisis de sistema.

El diseño es el núcleo técnico de la Ingeniería del Software. Durante el diseño se desarrollan, revisan y documentan los refinamientos progresivos de la estructura de datos, arquitectura, interfaces y datos procedimentales de los componentes del software. El diseño da como resultado representaciones del software para evaluar la calidad. (26)

Para la realización del diseño se tendrá en cuenta un conjunto de patrones, que constituyen una guía para resolver problemas comunes en programación que generalmente están presentes en el diseño de un programa. Cada patrón explica cómo resolver un determinado problema, bajo determinadas circunstancias.

Patrones Arquitectónicos

Los patrones arquitectónicos especifican un conjunto predefinido de subsistemas con sus responsabilidades y una serie de recomendaciones, para organizar los distintos componentes.

Para el desarrollo de la herramienta se utilizará el patrón Modelo Vista Controlador (Model-View-Controller-MVC), por sus principales ventajas:

- **Soporte para múltiples vistas:** puesto que la vista se separa del modelo y no hay ninguna dependencia directa entre vista y modelo, la interfaz de usuario puede mostrar múltiples vistas de los mismos datos al mismo tiempo.
- **Mayor soporte a los cambios:** los requisitos de interfaz tienden a cambiar más rápidamente que las reglas de negocio. Puesto que el modelo no depende de las vistas, la adición de nuevos tipos de vista al sistema generalmente no afectan al modelo. Por tanto, el ámbito del cambio se limita a la vista.

Además se emplea este patrón porque el framework de desarrollo Symfony, toma lo mejor de esta arquitectura MVC e implementa de forma que el desarrollo de aplicaciones sea rápido y sencillo representándolo a través de tres elementos fundamentales: el modelo, la vista y el controlador.

- **Modelo:** define la lógica de negocio, la base de datos pertenece a esta capa. Encapsula los datos y las funcionalidades. Es independiente de cualquier representación de salida y/o comportamiento de entrada. Es la representación específica de la información con la cual el sistema opera. La lógica de datos asegura su integridad y permite derivar nuevos datos.
- **Vista:** Es lo que utilizan los usuarios para interactuar con la aplicación, los gestores de plantillas pertenecen a esta capa. En Symfony la capa de la vista está formada principalmente por plantillas en PHP. Este presenta el modelo en un formato adecuado para interactuar, usualmente la interfaz de usuario. Muestra la información al usuario. Pueden existir múltiples vistas del modelo. Cada vista tiene asociado un componente controlador.
- **Controlador:** Recibe las entradas, traducidas a solicitudes de servicio para el modelo. Es un bloque de código que realiza llamadas al modelo para obtener los datos y se los pasa a la vista para que los muestre al usuario. Este responde a eventos, usualmente acciones del usuario e invoca cambios en el modelo y probablemente en la vista. Los eventos son traducidos a solicitudes de servicio para el modelo o la vista.

Patrones de diseño

Symfony está constituido de forma tal que permite la utilización de diferentes patrones de diseño. Los que se emplearán en la solución del sistema pertenecen al conjunto de patrones GRASP, ya que estos ayudan a refinar el diseño y a asignar las responsabilidades de las distintas clases de diseño, haciéndolas más sencillas, reutilizables y encapsuladas.

También describen los principios fundamentales de diseño de objetos para la asignación de responsabilidades. Constituyen un apoyo para la enseñanza que ayuda a entender el diseño de objeto esencial y aplica el razonamiento para el diseño de una forma sistemática, racional y explicable.

Patrones GRASP

Patrón Experto

En la arquitectura de Symfony, específicamente en el modelo, existen dos tipos de clases fundamentales:

- Las clases encargadas de la abstracción de datos (responsables de realizar todas las operaciones con la BD).
- Las de acceso a datos (responsables de interactuar con las clases de abstracción de datos, devuelven los objetos que necesitan los controladores en su forma original).

Symfony genera 3 clases por cada tabla de la BD, por ejemplo en la presente aplicación se tiene una tabla denominada Escenarios, se generan las siguientes clases: Escenarios, BaseEscenarios, EscenariosTable. De estas tres clases, se percibe que la clase que trabaja directamente con la BD, es la clase EscenariosTable, esta clase es la encargada de hacer las consultas a la BD utilizando Doctrine y la clase BaseEscenarios es la clase de abstracción de datos que tienen los atributos necesarios para realizar dicha función, por tanto deben implementar la responsabilidad de realizar las acciones directamente con la Base de Datos y aquí es donde se aplica el patrón Experto.

Patrón Creador

En Symfony hay clases que aplican el patrón Singleton y por tanto la creación de los objetos de estas clases se hacen una sola vez y los métodos de estas clases son estáticos, un ejemplo de esto es en las clases “action” cuando desea crear una instancia de una clase del Modelo, por ejemplo se tiene en la clase “action” del módulo gestionarEscenarios, la acción BuscarEscenarios toma el Id de un escenario y desea ver los datos del mismo, se necesita crear una instancia de ese escenario. En este caso el “action” usa un objeto de la clase EscenariosTable, por tanto puede crear instancias de ella evidenciándose este patrón.

Bajo Acoplamiento

Este patrón se manifiesta dentro del framework Symfony en la capa modelo ya que las clases de acceso a los datos son independientes de las clases de abstracción de datos. Hay poca dependencia entre esas clases lo que permite una mayor reutilización.

Alta Cohesión

Una de las características importantes del framework Symfony es la organización del trabajo en cuanto a la estructura del proyecto, lo cual permite crear y trabajar con clases de una alta cohesión. Por ejemplo, la clase actions del módulo gestionarEscenarios contiene varias funcionalidades con un propósito único, no desempeñado por el resto de los elementos, siendo estas funcionalidades las

encargadas de controlar las acciones de las plantillas. Esto garantiza que el software sea flexible a cambios sustanciales con mínimo efecto.

Patrón Controlador

Un ejemplo del patrón Controlador se puede apreciar en las clases `sfFrontController`, `sfFrontWebController`, `sfContext`. En la arquitectura del framework (MVC) hay una capa específicamente para los controladores, que son el núcleo del mismo. Symfony aplica el patrón "Front Controller" (Controlador frontal) y por tanto tiene una estructura bien organizada de controladores, que parte desde el "index.php" del ambiente y terminan en los "actions". Aquí cada clase en esta capa tiene su responsabilidad y es única, hay controladores que se encargan de la seguridad del sistema trabajando con ficheros YML, otros solo velan por identificar mediante unos datos las clases que deben realizar determinadas tareas.

Patrones GoF

Patrón Decorador

El uso de este patrón se pone de manifiesto en la relación que se establece entre el layout o plantilla global y las diferentes plantillas que forman parte de la vista. El archivo llamado `layout.php` que contiene el layout de la página, almacena el código HTML que es común a todas las páginas de la aplicación, para no tener que repetirlo en cada página. El contenido de la plantilla se integra en el Layout, o si se mira desde otro punto de vista, el Layout decora la plantilla.

Singleton (Instancia única)

Este patrón evidencia en el método `getInstance` de la clase `sfRouting`, el cual garantiza que esa clase solo tenga una única instancia, proporcionando un punto de acceso global a la misma.

La clase `sfRouting` es una de las más utilizadas por el controlador frontal (`sfFrontWebController`), porque es la encargada de enrutar todas las peticiones que se hagan a la aplicación. La clase `sfRouting` define además otros métodos muy útiles para la gestión manual de las rutas: `clearRoutes()`, `hasRoutes()`, `getRoutesByName()`.

Modelo de Diseño

El modelo de Diseño se utiliza para representar y documentar el diseño de una aplicación, comprende representaciones de datos, arquitectura, interfaces y componentes. Es un modelo de objeto que describe la realización de casos de uso, y sirve como una abstracción del modelo de Implementación y del código fuente. Se utiliza como entrada esencial para actividades de los flujos de trabajo Implementación y Prueba. Además de ser un artefacto integral que abarca todas las clases de diseño, subsistemas, paquetes, colaboraciones y las relaciones entre ellos.

Diagramas de clases del diseño

Un diagrama de clases del diseño es una representación concreta de lo que se debe implementar. Estos diagramas representan la parte estática del sistema a través de la representación de las clases y sus relaciones.

Para simplificar el desarrollo de la aplicación se utilizó uno de los patrones más utilizados en el framework Symfony: el patrón Modelo-Vista-Controlador (MVC), que está constituido por tres niveles. El Modelo representa la información con la que trabaja la aplicación, es decir, su lógica de negocio. La Vista transforma el modelo en una página web que permite al usuario interactuar con ella y el Controlador se encarga de procesar las interacciones del usuario y realiza los cambios apropiados en el modelo o en la vista.

A continuación se muestra un ejemplo de los diagramas de clases del diseño (Ver expediente de proyecto):

Diagrama de clases del diseño CU Gestionar_escenario

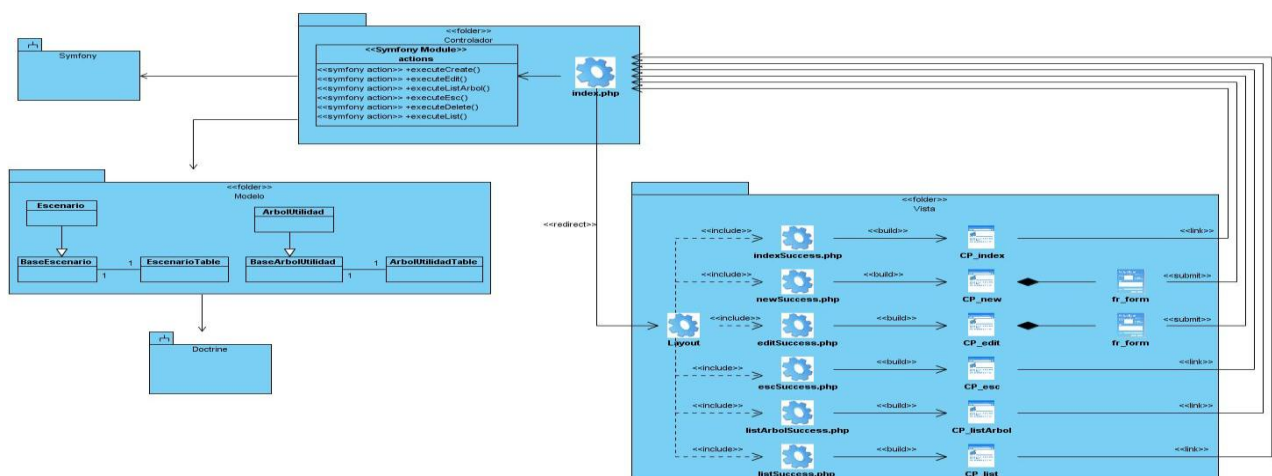


Figura 1: Diagrama de clases del diseño CU Gestionar_escenario.

Como se trata de una aplicación web, es necesario considerar fundamentalmente los lenguajes que se encontrarán del lado del servidor entre ellos está PHP y por lado del cliente JavaScript. En el framework los servicios se definen como las acciones (vistas con el estereotipo “symfony action”) agrupadas en el correspondiente módulo (clase con el estereotipo “symfony module”) al que pertenecen, se destaca la relación que existe de flujo de información entre el componente módulo del lado del servidor. En Symfony la capa del controlador, que contiene el código que une la capa de negocio con la de presentación, está dividida en varios componentes que se utilizan para diversos propósitos, aquí es donde aparece como primer elemento el controlador frontal, que es el único punto de entrada a la aplicación, carga la configuración y determina la acción a ejecutarse. Además esta capa cuenta con las acciones, las cuales contienen la lógica de la aplicación, verifica la integridad de las peticiones y preparan los datos requeridos por la capa de presentación, utilizando los componentes del framework. Después de asociar la acción, esta se comunica con la capa donde radica el paquete de clases con sus métodos y funciones necesarios para responder las peticiones del usuario, donde esta nueva capa interactúa con el modelo para el trabajo con los datos.

El modelo se encarga de la abstracción de la lógica relacionada con los datos, haciendo que la vista y las acciones sean independientes del tipo de gestor de bases de datos utilizado por la aplicación.

Las clases de la capa del modelo se generan automáticamente, en función de la estructura de datos de la aplicación. La librería Doctrine se encarga de esta generación automática, ya que crea el esqueleto o estructura básica de las clases y genera automáticamente el código necesario. Cada tabla del modelo de datos genera 3 clases, pero las verdaderas clases presentan la estructura de BaseClase.php y ClaseTable.php.

Diagramas de interacción del diseño. Secuencia

Los diagramas de interacción muestran las interacciones entre objetos mediante transferencia de mensajes entre objetos o subsistemas, por lo que son empleados para modelar aspectos dinámicos del sistema. Los diagramas de colaboración y de secuencia son dos tipos de diagramas de interacción semánticamente equivalentes y se puede pasar de uno a otro sin pérdida de información, pero sin embargo los diagramas de colaboración destacan la relación entre los objetos que interactúan y los de secuencia destacan el orden temporal de los mensajes. Para la realización de los casos de uso del diseño es más factible el empleo de los diagramas de secuencia ya que representan con más claridad el flujo de las acciones que debe realizar el sistema.

A continuación se muestran ejemplos de los diagramas de secuencias (Ver expediente de proyecto):

Diagrama de secuencia CU Adicionar escenario

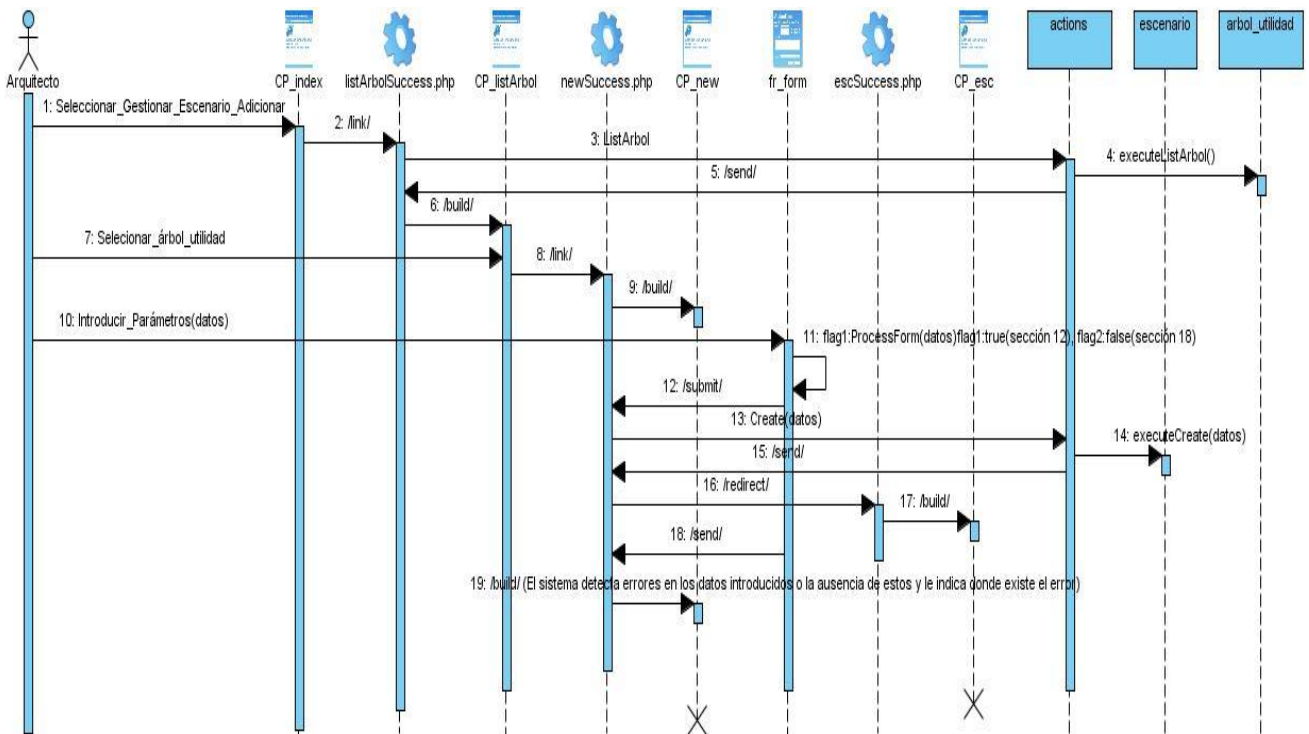


Figura 2: Diagrama de secuencia CU Adicionar escenario.

El diagrama de secuencia adicionar escenario representa las acciones que se ejecutan para realizar dicha acción. El Arquitecto selecciona el árbol de utilidad al que pertenecerá el escenario, luego introduce los datos del mismo, si los datos no son válidos o existen campos vacíos, el sistema indica donde se encuentran los errores. De no existir ningún error se verifica que el escenario a adicionar no se encuentre en la base de datos, de no cumplirse la condición anterior se muestra un mensaje de notificación indicando la existencia del mismo; en caso contrario se adiciona el nuevo escenario.

Diagrama de secuencia CU Eliminar escenario

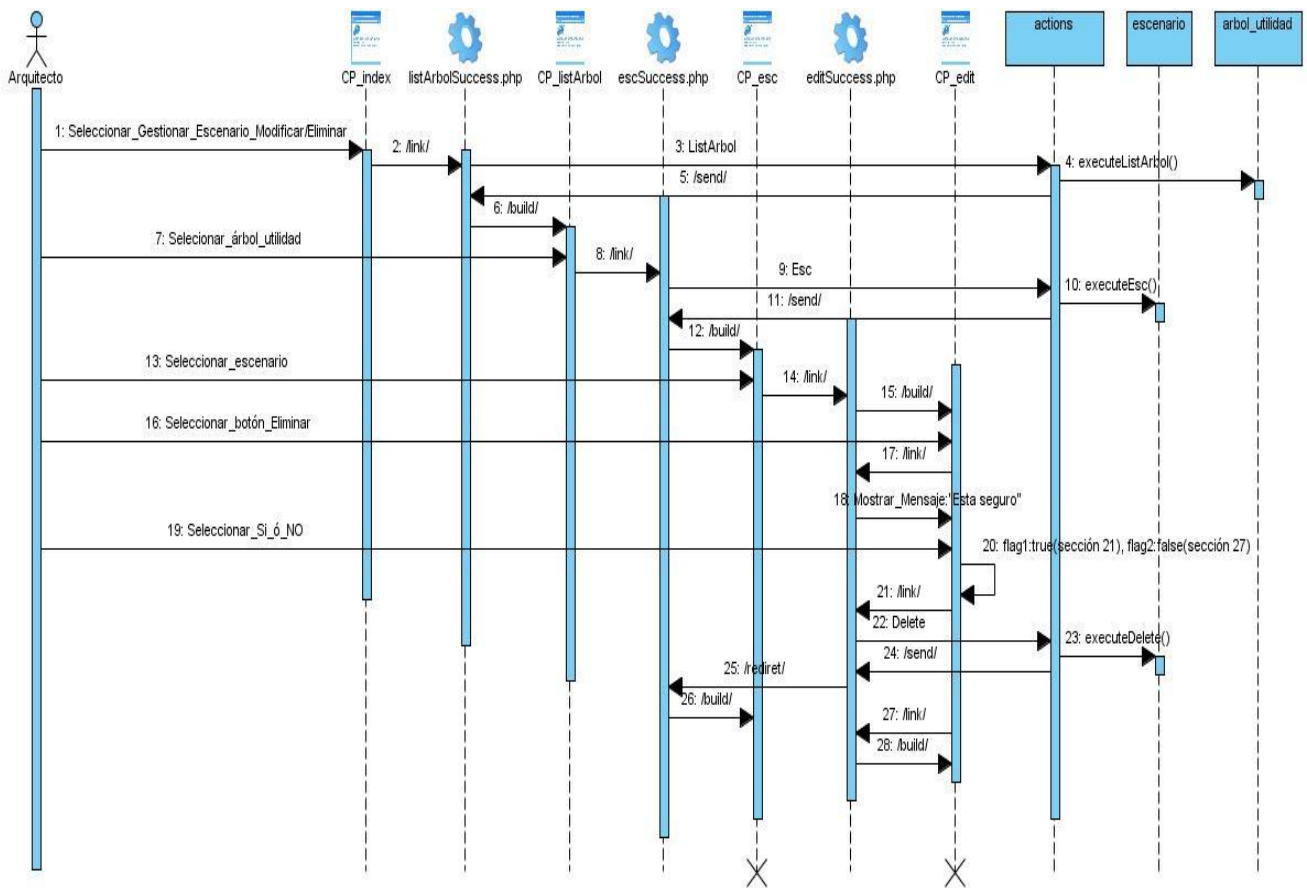


Figura 3: Diagrama de secuencia CU Eliminar escenario.

La figura 5 muestra el diagrama de secuencia eliminar escenario donde el Arquitecto selecciona el árbol de utilidad al que pertenece el escenario que desea eliminar. Se muestra el listado de escenarios correspondientes a dicho árbol, luego de seleccionado el escenario, el sistema muestra los datos asociados al mismo. El Arquitecto oprime el botón Eliminar, a continuación es mostrado un mensaje de confirmación y en caso de escoger la opción Aceptar el escenario es eliminado.

Diagrama de clases persistentes

La persistencia es la capacidad de un objeto de mantener su valor en el espacio y en el tiempo. Las clases persistentes representan información de larga duración o que persiste en el tiempo, es decir, es la información que se requiere almacenar para gestionarla en cualquier momento. Describe la estructura de un sistema mostrando sus clases, atributos y las relaciones entre ellos. Generalmente, estas clases tienen como origen las clases clasificadas como entidad porque ellas modelan la información del sistema y el comportamiento asociado de algún fenómeno o concepto (27).

Diagrama de clases persistentes

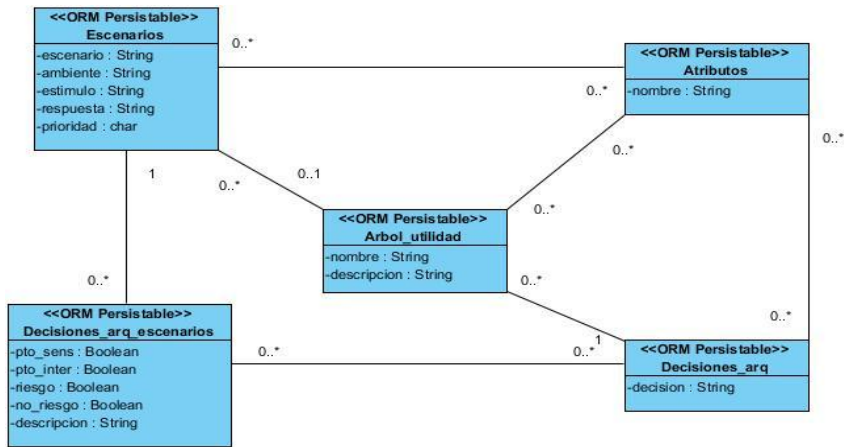


Figura 4: Diagrama de clases persistentes.

El diagrama de clases persistentes está constituido por las clases que poseen los datos que se gestionan en el sistema, los atributos que existen en cada una y las relaciones entre ellas.

Modelo de datos

El modelo de datos es un lenguaje utilizado para la descripción de una base de datos. Por lo general, un modelo de datos permite describir el tipo de datos que incluye la base y la forma en que se relacionan, así como las restricciones de integridad y las operaciones de manipulación de los datos. El modelo de datos describe la representación lógica y física de los datos persistentes.

Modelo de datos

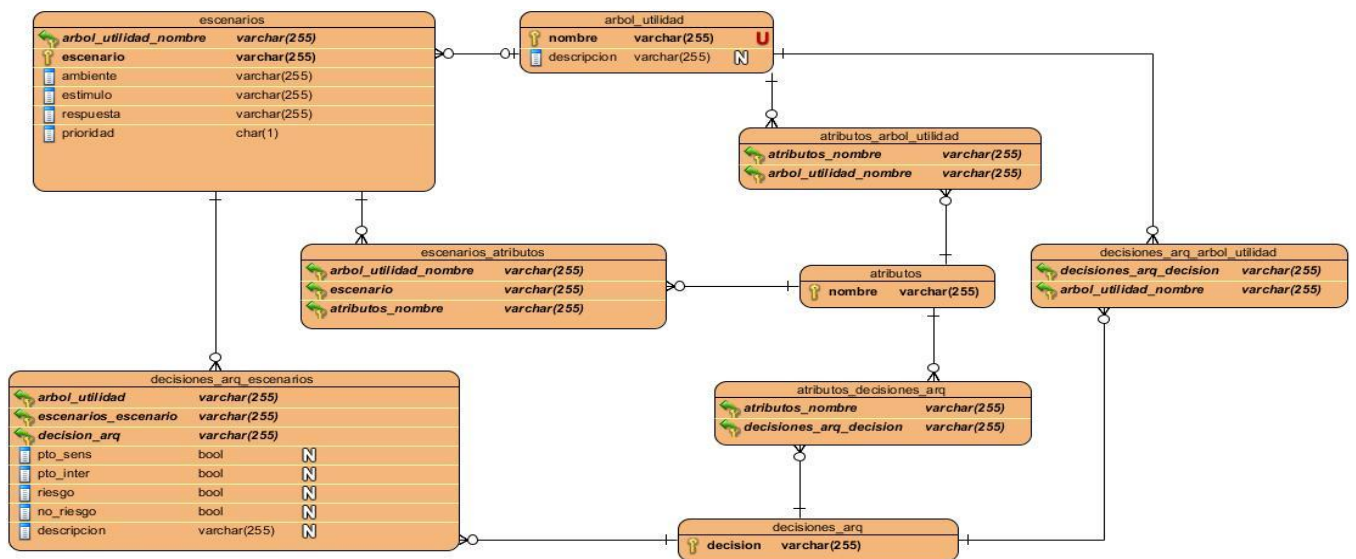


Figura 5: Modelo de datos.

En la figura 10 se muestra el modelo de datos asociado a la herramienta, teniendo como principales tablas involucradas en el proceso de evaluación: escenario, arbol_utilidad, decisiones_arq, decisiones_arq_escenarios, así como las relaciones existentes entre ellas.

2.5 Conclusiones del Capítulo

En este capítulo se definieron los requisitos funcionales y no funcionales que permitieron identificar las funcionalidades con las que contará el sistema, las cuales darán respuesta a las necesidades del problema. Los casos de uso fueron relacionados mediante un diagrama de casos de uso del sistema y se les realizó una descripción detallada logrando así un mayor acercamiento a lo que el sistema deberá cumplir. Además, en este capítulo se hizo uso del patrón MVC, al igual que la aplicación de patrones de diseño, se generaron los diagramas de clases del diseño y los diagramas de secuencia para cada uno de los escenarios, brindando una visión de cómo el usuario interactúa con el mismo y a partir del diseño de clases persistentes y la estructuración del diagrama de clases persistentes se obtuvo el modelo de datos, donde este describe la representación lógica y física de los datos persistentes.

CAPÍTULO 3: IMPLEMENTACIÓN Y PRUEBA

En este capítulo se describe la implementación del sistema, se analizan los principales artefactos como el modelo de implementación que incluye componentes, subsistemas de implementación y diagramas de componentes. Se comienza a implementar el sistema mediante las clases del diseño y se realizan pruebas a la herramienta con el objetivo de obtener funcionalidades con la menor cantidad de errores posibles.

3.1 Modelo de Despliegue

El Modelo de Despliegue se utiliza para capturar los elementos de configuración del procesamiento y las conexiones entre esos elementos. También se utiliza para visualizar la distribución de los componentes de software en los nodos físicos.

El diagrama de despliegue realizado representa tres nodos principales. El nodo PC_Cliente que requiere de un navegador que soporte JavaScript, el nodo Servidor_Web, en el cual debe estar instalado el servidor Apache y en el nodo Servidor_BD debe estar instalado el Sistema Gestor de Base de Datos PostgreSQL.

El nodo PC_Cliente estará conectado mediante el Protocolo de Transferencia de Hipertexto HTTP al nodo procesador que representa al Servidor Web. La conexión entre el Servidor Web y el Servidor de Base de Datos se realizará mediante el protocolo de comunicación TCP/IP.

Modelo de Despliegue

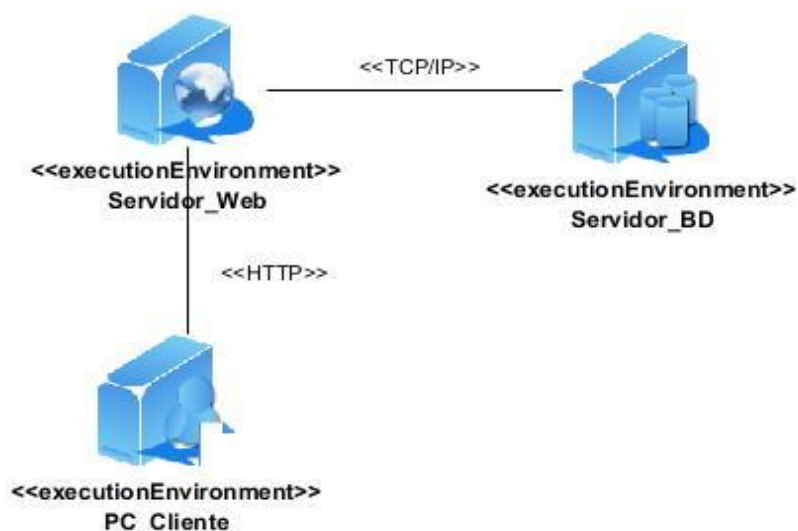


Figura 6: Modelo de despliegue.

3.2 Modelo de Implementación

El modelo de implementación describe cómo los elementos del modelo de diseño, las clases, se implementan en términos de componentes, ficheros de código fuente, ejecutables. Es considerado el artefacto más importante del flujo de trabajo de Implementación por su importancia para la comprensión del funcionamiento del sistema desde el punto de vista de componentes y sus relaciones. Representa la composición física de la implementación en términos de subsistemas de implementación y elementos de implementación y describe como los elementos de diseño se implementan en componentes.

Diagrama de componentes

Los diagramas de componentes se encuentran dentro del modelo de Implementación. Un componente es el empaquetamiento físico de los elementos de un modelo, como son las clases en el modelo de diseño, muestra las organizaciones y dependencias lógicas entre componentes software, sean éstos componentes de código fuente, binarios o ejecutables. El diagrama de componentes describe los elementos físicos del sistema y sus relaciones, cómo se organizan los componentes de acuerdo con los mecanismos de estructuración disponibles en el entorno de implementación y en el lenguaje o lenguajes de programación utilizados y cómo dependen los componentes unos de otros.

A continuación se muestra un ejemplo de los mismos (Ver expediente de proyecto):

Diagrama de componentes CU Gestionar_escenario

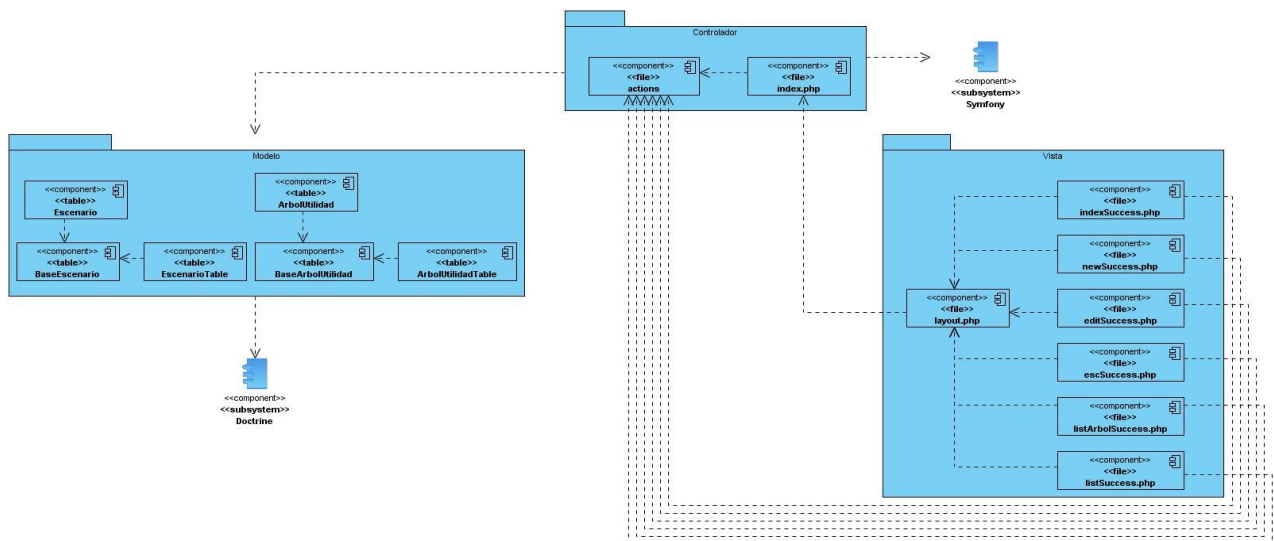


Figura 7: Diagrama de componentes CU Gestionar_escenario.

En la figura 12 se muestra el diagrama de componentes asociados al CU Gestionar_escenario donde se encuentra:

- Vista que contiene las interfaces de usuario, convertidas en componentes de tipo <<file>>.
- Controlador que contiene las acciones que dan respuesta a las peticiones del usuario que llegan a través de index.php, convertidas en componentes de tipo <<file>>. El Controlador tiene relación de dependencia con el <<subsystem>> Symfony.
- Modelo que contiene las tablas de la base de datos, convertidas en componentes de tipo <<table>>. El Modelo tiene relación de dependencia con el <<subsystem>> Doctrine.
- Symfony: paquete que se utiliza en la capa Controlador para la implementación de las acciones, convertido en un componente de tipo <<subsystem>>.
- Doctrine: paquete que se utiliza en la capa Modelo para convertir las tablas de la base de datos en clases, convertido en un componente de tipo <<subsystem>>.

3.3 Código Fuente

El código fuente es un conjunto de líneas de texto que son las instrucciones que debe seguir la computadora para ejecutar dicho programa. Por tanto, en el código fuente de un programa está descrito por completo su funcionamiento. Estas instrucciones son escritas en un lenguaje de programación que consiste en un conjunto de símbolos, reglas sintácticas y semánticas que definen su estructura y el significado de sus elementos y expresiones.

Estándares de Codificación

Un estándar de codificación completo comprende todos los aspectos de la generación de código. Usar técnicas de codificación sólidas y realizar buenas prácticas de programación con vistas a generar un código de alta calidad, es de gran importancia para la calidad del software y para obtener un buen rendimiento.

Estilo de Codificación Utilizado

- Todas las etiquetas php deben ser completas (<?php?>)... no reducidas (<? ?>).
- Los bloques de código siempre deben estar encerrados por llaves (incluso si solo constan de una línea).

- Tamaño = 4 (espacios) para:
 - Declaraciones dentro de las clases.
 - Enunciado dentro de métodos y funciones.
 - Enunciados dentro de bloques de comandos.

Ejemplo de Código Fuente

La mayoría de los métodos implementados responden a funciones básicas de inserción, actualización, eliminación, que se vuelven realmente sencillos mediante el uso que hace Symfony de Doctrine para realizar el mapeo objeto-relacional. Genera la mayor parte del código de acceso a datos proporcionando una abstracción al punto que permite que los implementadores tengan que utilizar muy pocas consultas a la base de datos.

A continuación se muestra el fragmento de código donde se muestran los métodos necesarios para adicionar escenario y se ofrece una breve descripción del mismo.

```
class gestEscActions extends sfActions {
    public function executeNew(sfWebRequest $request) {
        $esc = new Escenarios();
        $esc->setArbolUtilidadNombre($request->getParameter('nombre'));
        $this->form = new escenariosForm($esc);
    }

    public function executeCreate(sfWebRequest $request) {
        $this->forward404Unless($request->isMethod(sfRequest::POST));
        $esc = new Escenarios();
        $esc->setArbolUtilidadNombre($request->getParameter('nombre'));
        $this->form = new escenariosForm($esc);
        $this->processForm($request, $this->form);
        $this->setTemplate('new');
    }

    protected function processForm(sfWebRequest $request, sfForm $form) {
        $form->bind($request->getParameter($form->getName()), $request->getFiles($form->getName()));
        $form_project = $request->getParameter('escenarios');
        $arr_tributos = $form_project['atributos'];
        $arbol = $form_project['arbol_utilidad_nombre'];
        $escenario = $form_project['escenario'];

        if ($form->isValid()) {
            $esc = Doctrine::getTable('escenarios')->find(array($arbol, $escenario));
            if ($esc == null) {
                $escenarios = $form->save();
                foreach ($arr_tributos AS $idatributos) {
                    $this->addEscenariosAtributos($arbol, $escenario, $idatributos);
                }
                $this->redirect('gestEsc/show?arbol_utilidad_nombre=' . $escenarios->getArbolUtilidadNombre() . '&escenario=' . $escenarios->getEscenario());
            }
            $this->redirect('gestEsc/error?error=0');
        }
    }
}
```

Figura 8: Ejemplo de Código Fuente: Método Adicionar escenario.

El método `executeNew()` se encarga de crear el formulario para introducir los datos de escenario, el método `executeCreate()` obtiene los datos y los envía al método `processForm()` que es el encargado de validar los datos y adicionar el nuevo escenario.

3.4 Pruebas

Las pruebas constituyen el instrumento adecuado para determinar el estado de la calidad de un producto de software. En este proceso se ejecutan pruebas dirigidas a componentes del software o al sistema de software en su totalidad, con el objetivo de medir el grado en que el software cumple con los requerimientos. El objetivo de ellas es diseñar casos de pruebas, especificados de forma estructurada mediante técnicas de prueba, que sistemáticamente identifiquen diferentes clases de errores, realizándolo con la menor cantidad de tiempo y esfuerzo.

La disciplina de pruebas es iterativa e incremental. Se producen en cada iteración del ciclo de vida, comenzando con las primeras versiones de sistema. Esta disciplina desafía los supuestos riesgos e incertidumbres inherentes en el desarrollo de artefactos altamente técnicos.

En la evaluación de la herramienta se aplicó el nivel de trabajo Pruebas de Desarrollador, el cual está diseñado e implementado por el equipo de desarrollo. Cada nivel de prueba engloba una técnica de prueba específica según los atributos de calidad que se deseen verificar con las pruebas al software. Para evaluar los servicios que ofrece el sistema se utilizó el tipo de Prueba funcional, que tiene como objetivo asegurar el trabajo apropiado de los requisitos funcionales, incluyendo la navegación, entrada de datos, procesamiento y obtención de resultados.

Mediante la técnica de Caja Negra, también conocido como Pruebas de Comportamiento, se probará usando datos válidos e inválidos por cada requisito funcional, para confirmar que los resultados esperados ocurran cuando se usen datos válidos y se desplieguen los mensajes apropiados de error.

Se comprobó la correcta validación de los campos, verificando que cada cual aceptará exclusivamente los caracteres válidos, de esta forma se aplicó simultáneamente el método de Partición de Equivalencia de la técnica de Caja Negra, con la comprobación de las variables válidas e inválidas. En el proceso de pruebas se detectaron 10 No Conformidades, las cuales fueron corregidas y gestionadas correctamente.

Tabla 3: No Conformidades por casos de uso en cada iteración.

Fecha	Iteración	Caso de Prueba	Tipo de error	Descripción
17/05/2012	1	CU Gestionar usuario.	Validación	Validar el campo del correo.

17/05/2012	1	CU Gestionar escenario.	Recomendación	Cuando se adiciona un escenario mostrar la lista de escenarios adicionados.
17/05/2012	1	CU Gestionar árbol de utilidad.	Recomendación	Definir si las palabras árbol de utilidad se escriben en mayúsculas o minúsculas.
17/05/2012	1	CU Gestionar árbol de utilidad.	Ortografía	Cuando se adiciona un árbol que ya está en el sistema en el mensaje que indica el error la palabra árbol lleva tilde.
17/05/2012	1	CU Administrar decisión arquitectónica.	Funcional	Los botones azules deberían realizar las mismas funciones que por el menú.
17/05/2012	1	CU Administrar decisión arquitectónica.	Recomendación	Cuando se adiciona una decisión arquitectónica mostrar la lista de las mismas adicionadas.
24/05/2012	2	CU Visualizar reporte de evaluación.	Ortografía	Las palabras Decisiones Arquitectónicas están mal escritas.
24/05/2012	2	CU Visualizar reporte de evaluación.	Recomendación	No escribir las palabras en abreviaturas.
24/05/2012	2	CU Visualizar reporte de evaluación.	Ortografía	La palabra Descripción está mal escrita.
24/05/2012	2	CU Exportar	Recomendación	Cuando se exporta el pdf

		reporte de evaluación.		debería aparecer las palabras Decisión Arquitectónica no solo Decisión.
--	--	------------------------	--	---

La tabla muestra la cantidad de No Conformidades que se identificaron en cada iteración asociadas a cada caso de uso, en la primera y segunda iteración se encontraron 6 y 4 respectivamente, con un total de 10 No Conformidades.

La siguiente tabla es un ejemplo de una sección del Caso de Prueba realizado al caso de uso Gestionar_escenario. (Ver expediente de proyecto)

Tabla 4: Caso de Prueba realizado al caso de uso Gestionar_escenario.

Nombre de la sección	Escenarios de la sección	Descripción de la funcionalidad	V1	V2	V3	V4	V5	V6	Flujo central
SC1: Adicionar escenario.	EC 1.1 Adicionar escenario. Correctamente	Se adiciona el escenario satisfactoriam ente en el sistema.	V	V	V	V	V	V	<ol style="list-style-type: none"> 1. El Arquitecto escoge la opción de Adicionar. 2. El sistema muestra los árboles existentes para que el Arquitecto escoja a cual le adicionará el escenario. 3. El Arquitecto selecciona el árbol. 4. El sistema muestra una ventana de diálogo con un formulario para que el Arquitecto introduzca los datos del escenario. 5. El Arquitecto introduce los datos del escenario y oprime el botón Guardar.

										<p>6. El sistema valida los datos.</p> <p>7. Adiciona el nuevo escenario en la base de datos.</p> <p>8. El sistema muestra la lista de escenarios.</p>
EC	1.2	No se I	adiciona el en	V	V	V	V	V	V	<p>1. El Arquitecto escoge la opción de Adicionar.</p> <p>2. El sistema muestra los árboles existentes para que el Arquitecto escoja a cual le adicionará el escenario.</p> <p>3. El Arquitecto selecciona el árbol.</p> <p>4. El sistema muestra una ventana de diálogo con un formulario para que el Arquitecto introduzca los datos del escenario.</p> <p>5. El Arquitecto introduce los datos del escenario y oprime el botón Guardar.</p> <p>6. El sistema valida los datos.</p> <p>7. El sistema detecta errores en los datos introducidos o la ausencia de ellos e indica al Arquitecto donde existe el error.</p>

											(Volver al paso 5)
EC	1.3	Se	El Arquitecto	cancela la operación.	NA	NA	NA	NA	NA	NA	<ol style="list-style-type: none"> 1. El Arquitecto escoge la opción de Adicionar. 2. El sistema muestra los árboles existentes para que el Arquitecto escoja a cual le adicionará el escenario. 3. El Arquitecto selecciona el árbol. 4. El sistema muestra una ventana de diálogo con un formulario para que el Arquitecto introduzca los datos del escenario. 5. El Arquitecto decide cancelar la operación de adicionar el escenario volviendo a la lista de escenarios.

Para medir el rendimiento del sistema se utilizó el tipo de Pruebas de Carga y Estrés, que se realiza generalmente para observar el comportamiento de una aplicación bajo una cantidad de peticiones esperadas. La carga puede ser el número esperado de usuarios concurrentes utilizando la aplicación y que realizan un número específico de transacciones durante el tiempo que dura la carga. Esta prueba puede mostrar los tiempos de respuesta de todas las transacciones importantes de la aplicación.

El entorno para la ejecución de las pruebas con la herramienta JMeter estuvo determinada por:

- PC Cliente: Pentium(R) 4, 3.0Ghz, memoria RAM de 1.0GB, disco duro de 15GB.
- Servidor de BD: Pentium(R) 4, 3.0Ghz, memoria RAM de 1.0GB, disco duro de 15GB.
- Funcionamiento de la red: 100 Mbps de velocidad de enlace.

En las siguientes figuras se muestra las pruebas realizadas en la aplicación JMeter a la herramienta:

Ver Resultados en Árbol

Nombre: Ver Resultados en Árbol

Comentarios

Escribir todos los datos a Archivo

Nombre de archivo Log/Display Only: Escribir en Log Sólo Errores Successes

Muestra #	Start Time	Thread Name	Label	Tiempo de Muestra (ms)	Status	Bytes
1	12:01:46.146	Grupo de Hilos 1-1	Petición HTTP	2338		20407
2	12:01:46.235	Grupo de Hilos 1-2	Petición HTTP	2515		20407
3	12:01:46.340	Grupo de Hilos 1-3	Petición HTTP	2621		20407
4	12:01:46.439	Grupo de Hilos 1-4	Petición HTTP	2950		20407
5	12:01:46.540	Grupo de Hilos 1-5	Petición HTTP	2887		20407
6	12:01:46.645	Grupo de Hilos 1-6	Petición HTTP	2696		20407
7	12:01:46.645	Grupo de Hilos 1-6	Petición HTTP	3053		20407
8	12:01:46.854	Grupo de Hilos 1-8	Petición HTTP	2879		20407
9	12:01:46.754	Grupo de Hilos 1-7	Petición HTTP	3035		20407
10	12:01:47.054	Grupo de Hilos 1-10	Petición HTTP	2754		20407

No. de Muestras 10 Última Muestra 2754 Media 2772

Figura 9: Resultado de la prueba de Carga.

Como resultado de la prueba de Carga se obtuvo para una muestra de 10 usuarios conectados a la herramienta el tiempo de respuesta fue de 2.7 segundos, que comparado con el tiempo promedio de respuesta, definido en el requisito no funcional de Eficiencia, es menor por lo que se cumple el requisito descrito. Además se puede apreciar que no hubo ningún error en el resultado de las peticiones por HTTP.

Ver Resultados en Árbol

Nombre: Ver Resultados en Árbol

Comentarios

Escribir todos los datos a Archivo

Nombre de archivo Log/Display Only: Escribir en Log Sólo Errores Successes

Muestra #	Start Time	Thread Name	Label	Tiempo de Muestra (ms)	Status	Bytes
1	16:00:55.051	Grupo de Hilos 1-1	Petición HTTP	1526		19960
2	16:00:55.144	Grupo de Hilos 1-2	Petición HTTP	2300		19960
3	16:00:55.211	Grupo de Hilos 1-3	Petición HTTP	3184		19960
4	16:00:55.278	Grupo de Hilos 1-4	Petición HTTP	3260		19960
5	16:00:55.344	Grupo de Hilos 1-5	Petición HTTP	3333		19960
6	16:00:55.745	Grupo de Hilos 1-11	Petición HTTP	2954		19960
7	16:00:55.478	Grupo de Hilos 1-7	Petición HTTP	3443		19960
8	16:00:55.679	Grupo de Hilos 1-10	Petición HTTP	3340		19960
9	16:00:55.545	Grupo de Hilos 1-8	Petición HTTP	3500		19960
10	16:00:55.612	Grupo de Hilos 1-9	Petición HTTP	3440		19960
11	16:00:55.411	Grupo de Hilos 1-6	Petición HTTP	3663		19960
12	16:00:55.812	Grupo de Hilos 1-12	Petición HTTP	3696		19960
13	16:00:55.879	Grupo de Hilos 1-13	Petición HTTP	3679		19960
14	16:00:55.945	Grupo de Hilos 1-14	Petición HTTP	3662		19960
15	16:00:56.013	Grupo de Hilos 1-15	Petición HTTP	3653		19960

No. de Muestras 15 Última Muestra 3653 Media 3242

Figura 10: Resultado de la prueba de Estrés.

Como resultado de la prueba de Estrés se obtuvo para una muestra mayor que la definida en los requisitos funcionales de 10 usuarios conectados a la herramienta el tiempo de respuesta fue de 3.6 segundos y además no hubo ningún error en el resultado de las peticiones por HTTP, por lo que el sistema es capaz de soportar más usuarios que los definidos.

Prueba de Aceptación

Las pruebas de aceptación se realizan con el objetivo de validar la solución a través de un encuentro con el cliente, el cual comprueba que el sistema cumple con el funcionamiento esperado y da muestra de su conformidad en el documento oficial Carta de Aceptación del cliente donde quedan plasmados los resultados. ([Ver anexo 2](#))

3.5 Conclusiones del Capítulo

En este capítulo se realizó el diagrama de despliegue que propició un enfoque de como está distribuido el sistema físicamente. Se estructuraron las clases del diseño en paquetes de componentes mostrando la organización y las dependencias entre los componentes que conforman el sistema. Además se realizaron pruebas para validar que los requisitos fueron implementados correctamente a través la técnica de Caja Negra, aplicando el método de Partición de Equivalencia, para medir el rendimiento del sistema se realizaron pruebas de Carga y Estrés mediante la herramienta JMeter y se llevó a cabo la prueba de Aceptación por parte del cliente.

CONCLUSIONES

Una vez culminado el trabajo es posible afirmar que se cumplieron los objetivos trazados para el mismo:

- Se definieron las funcionalidades de la herramienta informática para asistir el proceso de evaluación de la arquitectura que se lleva a cabo en el laboratorio de Soluciones Integrales de DATEC, que permitirá elevar la eficiencia de este.
- El proceso de análisis y diseño de la herramienta permitió obtener los artefactos necesarios para el desarrollo de esta.
- La implementación de la herramienta permitió cumplir con todas las funcionalidades identificadas durante la etapa de análisis y diseño, así como la utilización de herramientas, lenguajes y tecnologías propuestas por el grupo de arquitectura del departamento del Centro.
- Se diseñaron casos de prueba basados en los casos de uso del Sistema, además se realizaron pruebas de Caja Negra para verificar la eficacia de la aplicación, de Carga y Estrés para medir el rendimiento de la misma y se llevó a cabo la prueba de Aceptación por parte del cliente.

RECOMENDACIONES

Después de haber alcanzado los objetivos trazados y como el producto informático se encuentra en su primera versión se plantea las siguientes recomendaciones:

- Implementar una nueva versión de la herramienta que incluya la realización de un mercado de datos a la fuente de información almacenada.
- Incorporar a la herramienta otros métodos de evaluación de arquitectura.

REFERENCIAS BIBLIOGRÁFICAS

1. **Reynoso, Carlos Billy.***Introducción a la Arquitectura de Software.* 2004.
2. **Reynoso, Carlos Billy.***Introducción a la Arquitectura de Software.* 2004.
3. **Reynoso, Carlos Billy.***Introducción a la Arquitectura de Software.* 2004.
4. **Brey, Andrés, Gustavo, Escobar, Gastón, Passerini, Nicolas y Arias, Juan.** *Arquitectura de Proyectos de IT. Evaluación de Arquitecturas.* Buenos Aires: Universidad Tecnológica Nacional. Facultad Regional de Buenos Aires - Departamento de Sistemas. 2005.
5. **Erika Camacho, Fabio Cardeso, Gabriel Nuñez.***Arquitecturas De Software.* 2004.
6. **Erika Camacho, Fabio Cardeso, Gabriel Nuñez.***Arquitecturas De Software.* 2004.
7. **Erika Camacho, Fabio Cardeso, Gabriel Nuñez.***Arquitecturas De Software.* 2004.
8. **Erika Camacho, Fabio Cardeso, Gabriel Nuñez.***Arquitecturas De Software.* 2004.
9. **Erika Camacho, Fabio Cardeso, Gabriel Nuñez.***Arquitecturas De Software.* 2004.
10. **Bosch, J.** *Design & Use of Software Architectures.* Addison-Wesley.2000.
11. **Bosch, J.** *Design & Use of Software Architectures.* Addison-Wesley.2000.
12. **Jacobson, I y G, Booch.***El proceso unificado de desarrollo de software.* La Habana : Félix Varela, 2004.
13. **Zaragoza, María de Lourdes Santiago.**<http://www.utvm.edu.mx>. [En línea] <http://www.utvm.edu.mx/OrganoInformativo/orgJul07/RUP.htm>.
14. **Zaragoza, María de Lourdes Santiago.** <http://www.utvm.edu.mx>. [En línea] <http://www.utvm.edu.mx/OrganoInformativo/orgJul07/RUP.htm>.
15. **Lisdanay Domínguez Medina, Edgar Rill Arencibia.***SISTEMA DE GESTIÓN DEL CAPITAL HUMANO.* 2010.
16. **Ileana Juez Aldana, Aliuska Gámez Lambert.***SISTEMA PARA EL CONTROL DEL CONSUMO DE COMBUSTIBLE.* 2010.
17. <http://www.netbeans.org>. [En línea] http://www.netbeans.org/index_es.html.
18. **Lisdanay Domínguez Medina, Edgar Rill Arencibia.***SISTEMA DE GESTIÓN DEL CAPITAL HUMANO.* 2010.
19. **Cirley Tupe, Juan Cisneros.***Evaluación y Selección de Framework de Desarrollo PHP: Symfony, Kumbia, CakePHP y Zend.* 2008.
20. **Cirley Tupe, Juan Cisneros.***Evaluación y Selección de Framework de Desarrollo PHP: Symfony, Kumbia, CakePHP y Zend.* 2008.
21. **Lisdanay Domínguez Medina, Edgar Rill Arencibia.***SISTEMA DE GESTIÓN DEL CAPITAL HUMANO.* 2010.

- 22. Osvaldo Ernesto Stable Vilches y Yandris Mata Cabrera.** *Desarrollo de un módulo para la validación de modelos estadísticos en el Paquete de Ayuda a la Toma de Decisiones y Soluciones Integrales (PATDSI).* 2010.
- 23.** [http://dev.mysql.com. \[En línea\] http://dev.mysql.com/doc/refman/5.0/es/tutorial.html](http://dev.mysql.com/doc/refman/5.0/es/tutorial.html).
- 24.** [http://dev.mysql.com. \[En línea\] http://dev.mysql.com/doc/refman/5.0/es/introduction.html](http://dev.mysql.com/doc/refman/5.0/es/introduction.html).
- 25. Agut, R. M.** *Especificación de Requisitos Software según el estándar IEEE 830.* 2001.
- 26. Pressman, R. S.** *Ingeniería Del Software. Un Enfoque Práctico.* 2002.
- 27. Castillo, I Perez y Mesa, R Valiente.** *Lims de Calidad del Centro de Ingeniería Genética y Biotecnología: Desarrollo de la Base de Datos del Módulo Sección de Mejoramiento de la Calidad.*2009.

BIBLIOGRAFÍA

1. **Reynoso, Carlos Billy.** *Introducción a la Arquitectura de Software*. 2004.
2. **Brey, Andrés, Gustavo, Escobar, Gastón, Passerini, Nicolas y Arias, Juan.** *Arquitectura de Proyectos de IT. Evaluación de Arquitecturas*. Buenos Aires: Universidad Tecnológica Nacional. Facultad Regional de Buenos Aires - Departamento de Sistemas. 2005.
3. **Bosch, J.** *Design & Use of Software Architectures*. Addison-Wesley.2000.
4. **Erika Camacho, Fabio Cardeso, Gabriel Nuñez.** *Arquitecturas De Software*. 2004.
5. **Jacobson, I y G, Booch.** *El proceso unificado de desarrollo de software*. La Habana : Félix Varela, 2004.
6. **Zaragoza, María de Lourdes Santiago.** <http://www.utvm.edu.mx>. [En línea] <http://www.utvm.edu.mx/Organoinformativo/orgJul07/RUP.htm>.
7. **Joskowicz, José.** *Reglas y Prácticas en*. 2008.
8. **Lisdanay Domínguez Medina, Edgar Rill Arencibia.** *SISTEMA DE GESTIÓN DEL CAPITAL HUMANO*. 2010.
9. **Ileana Juez Aldana, Aliuska Gámez Lambert.** *SISTEMA PARA EL CONTROL DEL CONSUMO DE COMBUSTIBLE*. 2010.
10. <http://www.netbeans.org>. [En línea] http://www.netbeans.org/index_es.html.
11. **Cirley Tupe, Juan Cisneros.** *Evaluación y Selección de Framework de Desarrollo PHP: Symfony, Kumbia, CakePHP y Zend*. 2008.
12. **Oswaldo Ernesto Stable Vilches y Yandris Mata Cabrera.** *Desarrollo de un módulo para la validación de modelos estadísticos en el Paquete de Ayuda a la Toma de Decisiones y Soluciones Integrales (PATDSI)*. 2010.
13. <http://dev.mysql.com>. [En línea] <http://dev.mysql.com/doc/refman/5.0/es/tutorial.html>.
14. **Agut, R. M.** *Especificación de Requisitos Software según el estándar IEEE 830*. 2001.
15. **Pressman, R. S.** *Ingeniería Del Software. Un Enfoque Práctico*. 2002.
16. **Castillo, I Perez y Mesa, R Valiente.** *Lims de Calidad del Centro de Ingeniería Genética y Biotecnología: Desarrollo de la Base de Datos del Módulo Sección de Mejoramiento de la Calidad*.2009.
17. <http://dev.mysql.com>. [En línea] <http://dev.mysql.com/doc/refman/5.0/es/introduction.html>.
18. <http://es.scribd.com/doc/297224/RUP>
19. <http://www.utim.edu.mx/~mgarcia/DOCUMENTO/ADSI2/RUP.pdf>
20. <http://iie.fing.edu.uy/~josej/docs/XP%20-%20Jose%20Joskowicz.pdf>
21. <http://www.eclipse.org/epf/general/OpenUP.pdf>

22. http://media.wiley.com/product_data/excerpt/49/07645260/0764526049.pdf
23. <http://www.lcc.uma.es/~av/Publicaciones/04/UMLProfiles-Novatica04.pdf>
24. <http://semanatecnologica.fordes.co.cu/ocs-2.3.2/public/site/242.pdf>
25. <http://www.visual-paradigm.com>
26. <http://www.rational.com.ar/herramientas/roseenterprise.html>
27. <http://www.monografias.com/trabajos5/insof/insof.shtml>
28. <http://tallertematico2010.fordes.co.cu/public/site/160.pdf>
29. <http://dev.mysql.com/doc/refman/5.0/es/features.html>

ANEXOS

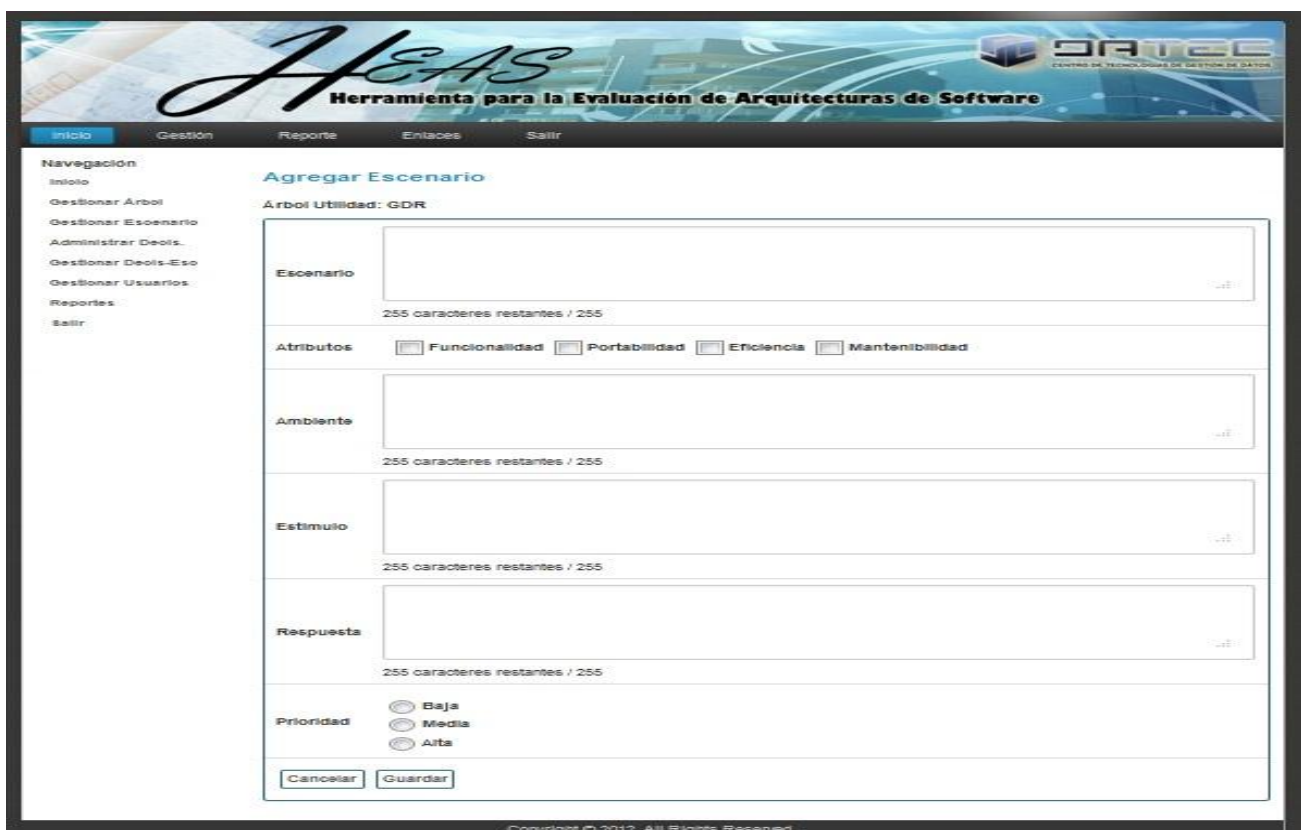
Anexo 1: Interfaces principales de la aplicación

A continuación se muestran algunas pantallas de la aplicación:



The screenshot shows the login page for HEAS. At the top, there is a banner with the HEAS logo and the text "Herramienta para la Evaluación de Arquitecturas de Software". To the right of the banner is the DATEC logo and the text "CENTRO DE TECNOLOGÍAS DE GESTIÓN DE DATOS". Below the banner is a navigation menu with "Inicio", "Reporte", "Enlaces", and "Acceder". On the left side, there is a "Navegación" sidebar with "Inicio", "Reportes", and "Acceder" (highlighted). The main content area contains a login form with fields for "Usuario" and "Contraseña", and a "Iniciar Sesión" button. To the right of the form is an illustration of a padlock with two people. At the bottom, there is a copyright notice: "Copyright © 2012. All Rights Reserved."

Figura 11: Interfaz de autenticación.



The screenshot shows the "Agregar Escenario" page. At the top, there is a banner with the HEAS logo and the text "Herramienta para la Evaluación de Arquitecturas de Software". To the right of the banner is the DATEC logo and the text "CENTRO DE TECNOLOGÍAS DE GESTIÓN DE DATOS". Below the banner is a navigation menu with "Inicio", "Gestión", "Reporte", "Enlaces", and "Salir". On the left side, there is a "Navegación" sidebar with "Inicio", "Gestionar Árbol", "Gestionar Escenario", "Administrar Datos", "Gestionar Datos-Eso", "Gestionar Usuarios", "Reportes", and "Salir". The main content area contains a form titled "Agregar Escenario" with the following fields: "Escenario" (text input, 255 caracteres restantes / 255), "Atributos" (checkboxes for "Funcionalidad", "Portabilidad", "Eficiencia", and "Mantenibilidad"), "Ambiente" (text input, 255 caracteres restantes / 255), "Estimulo" (text input, 255 caracteres restantes / 255), "Respuesta" (text input, 255 caracteres restantes / 255), and "Prioridad" (radio buttons for "Baja", "Media", and "Alta"). At the bottom of the form are "Cancelar" and "Guardar" buttons. At the bottom of the page, there is a copyright notice: "Copyright © 2012. All Rights Reserved."

Figura 12: Interfaz de Adicionar un escenario.

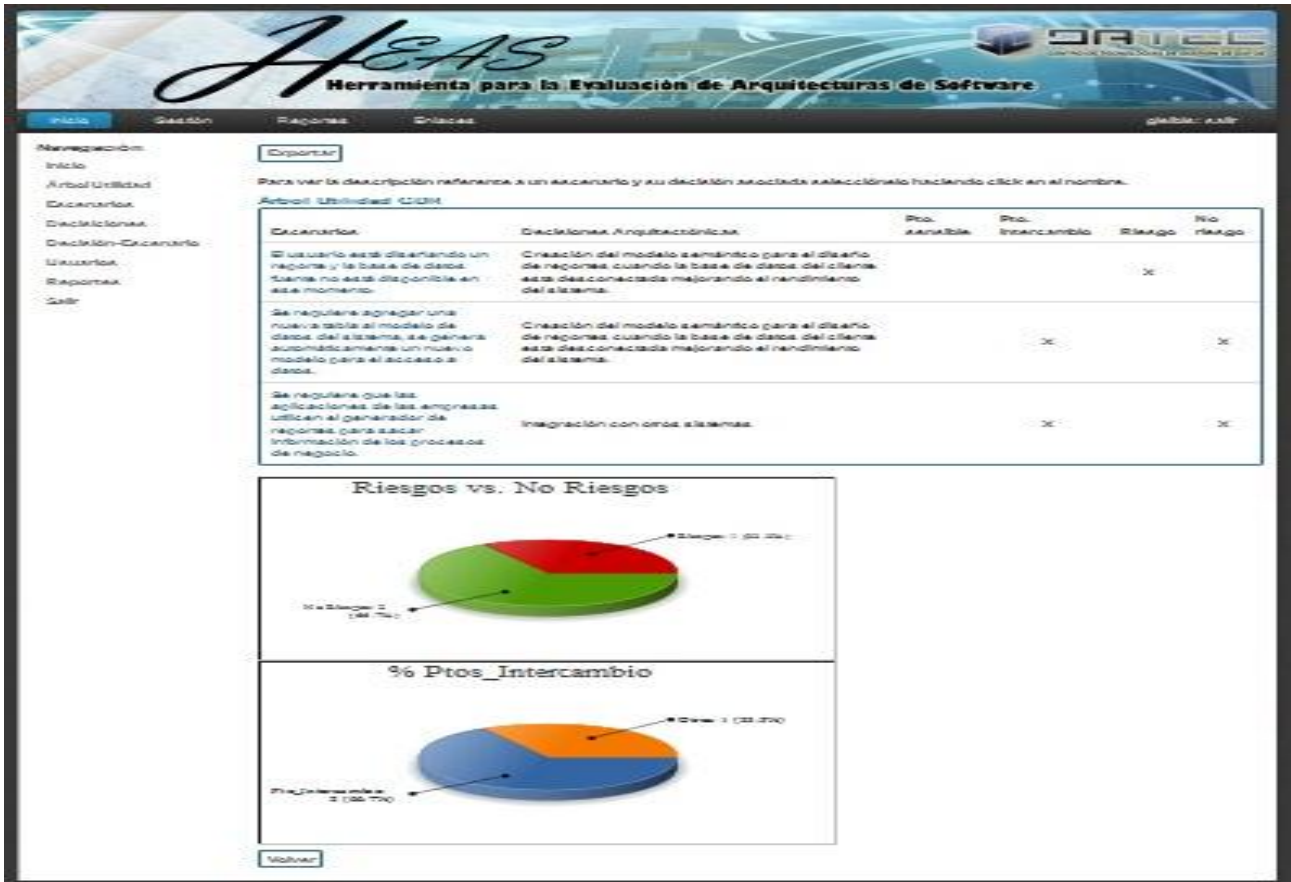


Figura 13: Interfaz de Visualizar reporte de evaluación.

Anexo 2: Carta de Aceptación



Figura 14: Carta de Aceptación.

GLOSARIO

1. AS: Arquitectura del Software
2. UCI: Universidad de las Ciencias Informáticas
3. DATEC: Centro de Tecnologías de Gestión de Datos
4. I+D: Investigación y Desarrollo
5. ADL: Lenguajes de Descripción Arquitectónica
6. SAAM: Método de Análisis de Arquitecturas de Software
7. ATAM: Método de Análisis de Acuerdos de la Arquitectura
8. ARID: Revisión de Diseño Activo Intermedio
9. ADR: Revisión de Diseño Activo.
10. RUP: Rational Unified Process
11. XP: Extreme Programming
12. OpenUP: Proceso Unificado Abierto
13. UML: Lenguaje de Modelado Unificado
14. PHP: Hipertexto Pre-Procesado
15. MVC: Modelo Vista Controlador
16. CU: Caso de Uso