Universidad de las Ciencias Informáticas
Facultad 4

Arquitectura de Software para la Plataforma de Gestión de Aprendizaje ZERA

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

Autores:
Ernesto Ruano Mamud
Yaismel Miranda Pons

Tutor:
MSc. Abel Ernesto Lorente Rodríguez

Co-Tutora:
Ing. Mairelis Gari Maribona

Ciudad de la Habana
Mayo 2011

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 2011.

_____

Ernesto Ruano Mamud

Firma del Autor

_____

Yaismel Miranda Pons

Firma del Autor

_____

MSc. Abel Ernesto Lorente Rodríguez

Firma del Tutor

_____

Ing. Mairelis Gari Maribona

Firma de la Co-Tutora

Agradecimientos

De Ernesto:

A toda mi familia por el apoyo incondicional que siempre me han brindado. A mis padres, mi abuelita Estrella, mi hermanito René, mi abuelo Neno, mis tíos Monguito y Nasdia, a Pepe y a Nana... gracias a todos por ayudarme a convertirme en la persona que hoy soy.

A la Revolución Cubana.

A la Facultad 4.

A los tutores Abel y Mairelis; y a otros que siempre nos ayudaron cada vez que los necesitabamos, especialmente Mayi, Tony y Alison.

A todos mis amigos con los que siempre he contado, al piquete de los Primones y mi piquete del PRE.

A la FEU y a Yusdel.

A toda aquella persona que de una forma u otra puso piedras en mi camino, gracias por ayudarme a convertirme en una persona más grande cada día.

De Yaismel:

A toda mi familia especialmente a mis padres y a mi hermanita, por el amor, el apoyo y la confianza que han depositado en mi.

A mi novia por estar a mi lado dándome confianza y apoyo, todos los días, ¡todas las noches!.

A los tutores Abel y Mairelis que estuvieron guiando el trabajo y siempre nos ayudaron cada vez que los necesitamos.

A los profesores Alison, Leonardo San Román, Héctor, la gente de Alfaomega y otros que han compartido conmigo todos estos años.

A la UCI por ser uno de los lugares en el que he vivido mis mejores experiencias.

A todas las personas que de una forma u otra hicieron posible la realización de este trabajo.

Dedicatoria

De Ernesto:

A mis padres y mi abuela Estrella, por todo el sacrificio incondicional que han realizado por mi a lo largo de tantos años, por apoyarme en todos mis actos y decisiones, y por estar siempre disponibles para mí en todos los momentos buenos y malos.

A mi hermano René que espero llegar a ser un ejemplo a seguir para él.

A la memoria de mi abuelito Carlos que ya no está junto a nosotros, lo recuerdo con mucho cariño.

De Yaismel:

A mi madre, que es la razón por la que me esfuerzo en ser cada día mejor, por estar siempre en los momentos difíciles y sentirme orgulloso de ser su hijo.

A mi padre por sus consejos, evitar que me convierta en una persona conformista y por enseñarme que para ser mejor no hay que estar en competencia con el mundo sino con uno mismo.

Resumen

Las plataformas de gestión de aprendizaje surgen con el desarrollo de las Tecnologías de la Información y las Comunicaciones en los últimos tiempos. En el desarrollo del presente trabajo se analizan los principales conceptos, estilos y patrones relacionados con la arquitectura de software, así como las metodologías, herramientas y tecnologías que se utilizarán para su desarrollo y definición.

La arquitectura expuesta en este documento sustenta el desarrollo de una plataforma de gestión del aprendizaje basada en el concepto: “hiperentornos de aprendizaje” los cuales se fundamentan en el modelo pedagógico cubano y desarrollado por el Ministerio de Educación de Cuba. Se realiza una estructuración del sistema que permite la adecuación de este modelo a otros entornos o sistemas educacionales tanto a nivel nacional como internacional, dados principalmente por el desarrollo de múltiples instrumentos de enseñanza, enfocados todos a la gestión de los recursos educativos.

Se presenta una estructuración de los subsistemas de diseño en base a las responsabilidades y funcionalidades que estos implementan. Se describen los principales componentes que conforman la línea base de la arquitectura definiendo además los casos de uso más significativos. Finalmente se realiza una evaluación de la propuesta en la cual se exponen las técnicas y métodos aplicados. Se hace un análisis de los resultados de la aplicación del método, obteniéndose un total de cinco no riesgos, cinco puntos de sensibilidad, un punto de intercambio y un riesgo.

Índice de contenido

Índice de ilustraciones

Índice de tablas

Introducción

Con el desarrollo tecnológico en los últimos tiempos las Tecnologías de la Información y las Comunicaciones (TIC) han llegado a ser uno de los pilares básicos de la sociedad, donde el sistema educativo no se ha quedado al margen de los nuevos cambios. Estas brindan nuevas posibilidades de instrumentación de los conocimientos que las tecnologías tradicionales no pueden cubrir. Además diversifican el conocimiento con el uso de herramientas telemáticas y de teleformación como las enciclopedias multimedia, los videos, el software educativo, la realidad virtual, etc. todo lo cual propicia una mayor calidad en el proceso de enseñanza-aprendizaje facilitando la tarea de difundir, transmitir y crear conocimientos.

Las TIC suscitan la colaboración en los alumnos, les ayuda a centrarse en el aprendizaje, mejoran la motivación y el interés, favorecen el espíritu de búsqueda, promueven la integración y estimulan el desarrollo de habilidades intelectuales tales como el razonamiento, la resolución de problemas, la creatividad y la capacidad de aprender a aprender [1]. Ortiz, plantea que con las nuevas TIC, que utilizan la red como medio de distribución de la información, surgen nuevos términos como son: *aprendizaje en red*, la *educación virtual*, la *teleducación* o el *e-learning [2]*. En este proceso han jugado un papel importante las plataformas de gestión del aprendizaje o los sistemas de administración del aprendizaje (LMS, por sus siglas en inglés), en los que se pueden organizar y distribuir los materiales de cursos, desarrollar foros de discusión, realizar tutorías, seguimientos y evaluación a los alumnos, entre muchas más actividades.

Desde el triunfo de la Revolución Cubana en 1959, Cuba se planteó un camino de desarrollo en el que se resolvieran los problemas materiales y espirituales de su población. Dentro de los innumerables cambios que tuvieron lugar, específicamente en la esfera de la educación se crearon varios programas con la meta de vincular las TIC a la educación en todos los niveles de enseñanza; objetivo que se ha ido cumpliendo en el transcurso de los años. En uno de estos programa denominado “la Batalla de Ideas” se crea la Universidad de las Ciencias Informáticas (UCI) con el objetivo de crear profesionales de alto nivel en la

rama de la Informática y comprometidos con su patria, vinculando la dinámica de la docencia con la producción y la investigación. De esta forma se aporta a la informatización de la sociedad cubana y a la economía del país a través de proyectos productivos en tiempo real.

En Marzo 6 de 2009 se firma un contrato entre la Empresa “Alfaomega Grupo Editor” y “Albet, Ingeniería y Sistemas”, para el desarrollo de una plataforma de gestión de aprendizaje, incentivados específicamente por la colección Futuro que el Ministerio de Educación de Cuba (MINED) ha desarrollado para estudiantes de la educación media, siendo la UCI y específicamente el Centro de Tecnologías para la Formación (FORTES) quien implementaría la solución. Más adelante esta plataforma fue nombrada como plataforma de gestión de aprendizaje ZERA.

En el desarrollo de un LMS, como en cualquier otro sistema de software se hace necesario garantizar ciertos modelos de calidad que se determinan mediante la selección de diversos criterios. Estos pueden ser seleccionados atendiendo a diferentes prioridades las cuales establecen a su vez los criterios de éxito de dicho sistema. Para ello se hace necesario la utilización de metodologías de desarrollo, las que guíen este proceso. Un elemento significativo en el proceso de desarrollo es la concepción de un diseño de alto nivel, que permita organizar y comprender la estructura del sistema que se desarrolla y brinde a todo el equipo una idea clara de lo que se está desarrollando [3].

Algunos de los criterios que determinan la calidad están definidos en los requerimientos no funcionales de todo el sistema, entre estos se encuentran los relacionados con la funcionalidad, fiabilidad, usabilidad, eficiencia, mantenibilidad y portabilidad.

Una de las principales limitaciones en el desarrollo de la plataforma ZERA, es precisamente la ausencia de definiciones que permitan a la misma alcanzar un nivel de calidad adecuado. Durante el proceso de desarrollo de software han existido deficiencias que limitan y dificultan el proceso en sí. Entre ellos se pueden ver: la escasa reutilización, la mala selección de elementos y componentes de software, errónea integración entre los subsistemas del software, carencia de estilos arquitectónicos, limitado uso de patrones y protocolos de comunicación, poca o ninguna interoperabilidad entre la plataforma y otros

sistemas, problemas de sincronización y acceso a datos, códigos lógicamente correctos pero mal estructurados y con poca organización y claridad. Donde la alternativa a todos estos problemas está en la correcta definición y descripción de una arquitectura de software para la plataforma ZERA.

La arquitectura de software surge de la necesidad de hacer los sistemas más modulares, que permitan la reutilización de componentes, por lo que su implementación reduce considerablemente el coste. 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 [3].

Con lo expuesto anteriormente se puede formular el siguiente **problema científico**: ¿Cómo establecer la organización lógica, funcional y física de la Plataforma de Gestión de Aprendizaje ZERA?

El **objeto de estudio** de la presente investigación consiste en el Proceso de Desarrollo de Software y como **objetivo general** desarrollar la propuesta de arquitectura de software para la plataforma ZERA. De este último se derivan los siguientes **objetivos específicos:**

1. Analizar el estado actual de las tendencias y tecnologías acerca de los modelos arquitectónicos para el desarrollo de software.
2. Elaborar la línea base de la arquitectura de software de la plataforma.
3. Evaluar la arquitectura definida.

El **campo de acción** se centra en la arquitectura de software para la plataforma ZERA.

Se plantea como **idea a defender** que si se define la línea base de la arquitectura de la plataforma ZERA, se garantizará un producto eficaz, seguro y escalable.

Los métodos científicos de investigación a utilizar son:

De los métodos teóricos:

- **Analítico - sintético**: A través de herramientas como el análisis y síntesis de la información, este método ayudará durante la investigación en el razonamiento y entendimiento del estudio teórico, permitiendo la extracción de los elementos más importantes que se relacionan con el objeto de estudio.
- **Inductivo - deductivo**: Una vez analizada y comprendida toda la información referente al objeto de estudio, se utilizará este método como base fundamental para la aplicación en el campo de acción de la información obtenida.
- **Modelación**: Este método es útil para realizar los modelos correspondientes a la descripción de la arquitectura.

Para dar cumplimiento al objetivo propuesto, se realizarán las siguientes **tareas**:

- Investigación de los diferentes estilos y patrones arquitectónicos y de diseño.
- Realización de un estudio arquitectónico de las principales Plataformas de Enseñanza.
- Comparación de las distintas propuestas arquitectónicas consultadas, para obtener la más adecuada para el desarrollo de la plataforma.
- Selección de los estilos y patrones a utilizar en la plataforma.
- Selección de las herramientas y tecnologías adecuadas que cumplan los requisitos para ser utilizadas en el desarrollo.
- Identificar los elementos y mecanismos de diseño.
- Integrar los elementos de diseño.
- Estudio y selección de los criterios de evaluación de la arquitectura.
- Evaluación de la arquitectura definida para la plataforma.
- Valoración de los resultados de la evaluación.

El contenido del trabajo está estructurado de la siguiente manera:

Capítulo 1: Fundamentación teórica: En este capítulo se aborda en detalle lo relacionado con la fundamentación teórica que sustenta el presente trabajo. Descripción de los conceptos básicos de la Arquitectura de Software. Selección de los patrones y estilos arquitectónicos a utilizar en el desarrollo de la arquitectura, artefactos, metodología y herramientas que se emplearán para el desarrollo.

Capítulo 2: Descripción de la arquitectura: Se explican las características del sistema propuesto. Se identifican los componentes arquitectónicos significativos y se representan las características del sistema en las vistas arquitectónicas.

Capítulo 3: Evaluación de la arquitectura: Se seleccionan los métodos de evaluación y atributos de calidad más adecuados. Se especifican las ventajas, riesgos de los diseños y se muestran los resultados de la evaluación luego de emplear tales métodos.

Capítulo 1: Fundamentación teórica

1.1 Introducción

En el presente capítulo se realiza el estudio del marco teórico referente al tema de la arquitectura de software para el desarrollo de una aplicación web, sus inicios, tendencias, así como una breve descripción de los principales conceptos asociados a esta. Se analizan y seleccionan las metodologías, herramientas y tecnologías a utilizar en el desarrollo de las tareas definidas para dar cumplimiento a los objetivos específicos. Además se realiza un estudio valorativo de los principales productos similares que existen en la actualidad.

1.2 Arquitectura de Software

1.2.1 Concepto y Definición de la Arquitectura de Software

La arquitectura de software es un concepto fácil de entender y que la mayoría de los ingenieros aprecian intuitivamente, sobre todo los que tienen un poco de experiencia, pero resulta difícil definirlo con precisión. En concreto, es difícil dibujar una línea precisa entre el diseño y la arquitectura, esta última es un aspecto de diseño que se concentra en algunas características específicas.

Existen gran variedad de conceptos y definiciones sobre qué es la Arquitectura de Software dado por disímiles autores y ninguno de estos es respaldado unánimemente por la totalidad de los arquitectos. En la presente investigación se analizan aquellos dados por organizaciones e instituciones reconocidas, así como de autores de prestigio mundial:

- Una definición reconocida es la de Paul Clements del Instituto de Ingeniería de Software de la Universidad de Carnegie Mellon [4]: “La arquitectura de software de un programa o sistema de cómputo es la estructura o estructuras del sistema, que comprende los componentes de software, sus propiedades externamente visibles, y las relaciones entre ellos”.

- Por otra parte la definición más reconocida internacionalmente por muchos arquitectos tributa a la industria y pertenece a la IEEE 1471, plantea lo siguiente: ”La Arquitectura de Software es la organización fundamental de un sistema encarnada en sus componentes, las relaciones entre ellos y el ambiente y los principios que orientan su diseño y evolución".
- La versión de la guía oficial de la corporación IBM sobre el Proceso Unificado de Rational del 2007 plantea que la arquitectura de software puede entenderse como aquella estructura del programa que cohesiona las funcionalidades más críticas y relevantes (necesarias para el sistema), y que sirve de soporte al resto de funcionalidades finales (necesarias para el usuario).

Para la presente investigación se asume la arquitectura de software según el concepto de la IEEE ya que este expresa de forma sencilla y en lenguaje coloquial la idea que quiere transmitir. Además la definición es amplia, abarcadora y proviene de una de las fuentes más confiables en el mundo en cuanto a estandarizaciones y definiciones se trata.

1.2.2 Estilos de Arquitectura de Software.

Varios autores establecen el razonamiento sobre estilos de arquitectura como uno de los aspectos fundamentales de la disciplina. Un estilo es un concepto descriptivo que define una forma de articulación u organización arquitectónica. El conjunto de los estilos cataloga las formas básicas posibles de estructuras de software, mientras que las formas complejas se articulan mediante composición de los estilos fundamentales [5].

Un estilo arquitectónico define tanto un vocabulario de tipos de componentes y conectores, como un conjunto de restricciones sobre cómo combinar esos componentes y conectores:

- Sirven para sintetizar estructuras de soluciones.
- Pocos estilos abstractos encapsulan una enorme variedad de configuraciones concretas.
- Permiten evaluar arquitecturas alternativas con ventajas y desventajas conocidas ante diferentes conjuntos de requerimientos no funcionales.

A diferencia de los patrones de diseños, los estilos se ordenan en seis o siete clases fundamentales y unos veinte ejemplares, como máximo. Es digno de señalarse el empeño por abarcar todas las formas existentes de aplicaciones en un conjunto de dimensiones tan modestas. Las arquitecturas complejas o compuestas resultan del agregado o la composición de estilos más básicos [5].

Algunos estilos arquitectónicos se muestran a continuación:

Estilos de Flujo de Datos

- Tubería y filtros

Estilos Centrados en Datos

- Arquitecturas de Pizarra o repositorio

Estilos de llamada y retorno

- Modelo-Vista-Controlador (MVC)
- Arquitectura en capas
- Arquitecturas orientadas a objetos
- Arquitecturas basadas en componentes

Estilos de código móvil

- Arquitectura de máquinas virtuales

Estilos heterogéneos

- Sistemas de control de procesos
- Arquitecturas basadas en atributos

Estilos Peer-to-Peer

- Arquitecturas basadas en eventos
- Arquitecturas orientadas a servicios
- Arquitecturas basadas en recursos

La elección de un estilo viene dado por la adaptabilidad de las características del sistema a el estilo en sí. Una vez seleccionado el estilo a usar, este define los patrones arquitectónicos que se deben implementar para cumplir con sus principios.

La plataforma ZERA está concebida para gestionar grandes volúmenes de información, ya sea desde un usuario del sistema hasta recursos como imágenes, videos, sonido, entre otros; donde toda esta información se necesita visualizar a los usuarios finales. Dadas estas características se seleccionó de los estilos estudiados el que se adaptaba a estas necesidades:

1. **Modelo-Vista-Controlador**

   Se utiliza principalmente cuando es necesario modularizar la interfaz de usuario, las reglas de negocio y el control de eventos. Separa el modelado del dominio, la presentación y las acciones basadas en datos ingresados por el usuario en tres clases diferentes:

   - **Modelo**: Administra el comportamiento y los datos del dominio de aplicación, responde a requerimientos de información sobre su estado. Mantiene el conocimiento del sistema. No depende de ninguna vista y de ningún controlador.
   - **Vista**: Maneja la visualización de la información.
   - **Controlador**: Interpreta las acciones del ratón y el teclado, informando al modelo y/o a la vista para que cambien según resulte apropiado. Tiene tres variantes principales: Activa, Pasiva y Documento-Vista.

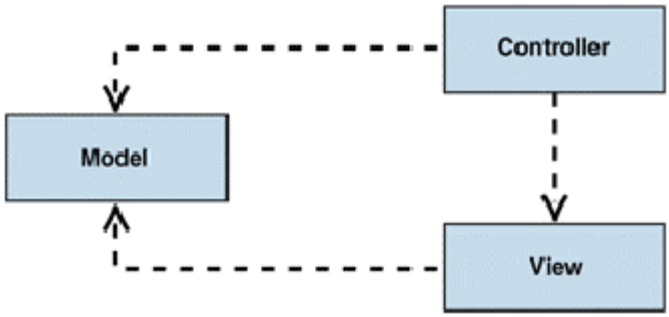


Fig. 1: Modelo-Vista-Controlador

Algunas de las ventajas que presenta este estilo se muestran a continuación [6]:

- **Soporte de vistas múltiples**: Dado que la vista se encuentra separada del modelo y no hay dependencia directa del modelo con respecto a la vista, la interfaz de usuario puede mostrar múltiples vistas de los mismos datos simultáneamente. Por ejemplo, múltiples páginas de una aplicación web pueden utilizar el mismo modelo de objetos, mostrado de maneras diferentes.

- **Adaptación al cambio**: Los requerimientos de interfaz de usuario tienden a cambiar con mayor rapidez que las reglas de negocios. Los usuarios pueden preferir distintas opciones de representación, o requerir soporte para nuevos dispositivos como teléfonos celulares. Dado que el modelo no depende de las vistas, agregar nuevas opciones de presentación generalmente no afecta al modelo.

1.2.3 Patrones

Existen buenas prácticas bien conocidas en el diseño arquitectural, especialmente en cuanto a la arquitectura lógica a gran escala, y estas prácticas se han escrito en forma de patrones [7]. El primer libro que se dedicó al tema de los patrones de arquitectura fue Pattern-Oriented Software Architecture (POSA).

En una mirada cercana a los patrones existentes demuestra que estos cubren varios rangos de escala y abstracción. Algunos patrones ayudan a estructurar sistemas de software en subsistemas. Otros por su lado, sustentan el refinamiento de subsistemas y componentes, o la relación existentes entre ellos. Algunos ayudan en la implementación de aspectos particulares del diseño en un específico lenguaje de programación [8].

A continuación se ofrece una clasificación bastante abarcadora de los patrones [9]:

- **Patrones de arquitectura:** Relacionados a la interacción de objetos dentro o entre niveles arquitectónicos. Proporcionan solución a problemas arquitectónicos, adaptabilidad a requerimientos cambiantes, rendimiento, modularidad, acoplamiento.
- **Patrones de diseño:** Fueron construidos dado los problemas con la claridad de diseño, multiplicación de clases, adaptabilidad a requerimientos cambiantes, proponiendo como solución el comportamiento de factoría y Clase-Responsabilidad-Contrato (CRC).
- **Patrones de análisis:** Usualmente específicos de aplicación, se aplican durante el análisis para solucionar problemas relacionados con el modelo de dominio, completitud, integración y equilibrio de objetivos múltiples, planeamiento para capacidades adicionales comunes, entre otros.
- **Patrones de proceso o de organización:** Tratan todo lo concerniente con el desarrollo o procesos de administración de proyectos, o técnicas, o estructuras de organización.
- **Patrones de idioma:** Reglan la nomenclatura en la cual se escriben, se diseñan y desarrollan nuestros sistemas.

En líneas generales, un patrón sigue el siguiente esquema:

- **Contexto**: Es una situación de diseño en la que aparece un problema de diseño.
- **Problema**: Es un conjunto de fuerzas que aparecen repetidamente en el contexto.
- **Solución:** Es una configuración que equilibra estas fuerzas. Esta abarca:
    - Estructura con componentes y relaciones

- Comportamiento a tiempo de ejecución: aspectos dinámicos de la solución, como la colaboración entre componentes, la comunicación entre ellos, etc.

A continuación se dará paso al estudio de dos tipos de patrones: los arquitectónicos y los de diseño siendo estos los de mayor aporte a la arquitectura de software de un sistema.

1.2.3.1 Patrones Arquitectónicos

Las arquitecturas de software viables han sido construidas siguiendo algunos principios generales de estructuración. Estos principios son descritos con los patrones de arquitectura.

Un patrón arquitectónico expresa un diseño de organización estructural fundamental para sistemas de software. Este provee un conjunto de subsistemas predefinidos, con la especificación de sus responsabilidades e incluye las reglas y guías para organizar la relación entre ellos [8].

Estos patrones son plantillas para arquitecturas concretas de software. Estos especifican las propiedades estructurales de todo el sistema de una aplicación cualquiera, y poseen un impacto sobre la arquitectura de los subsistemas de esta aplicación. La selección de un patrón arquitectónico se convierte, por lo tanto, en una decisión fundamental de diseño cuando se está desarrollando el sistema.

Del estudio de patrones arquitectónicos se seleccionaron los que aparecen a continuación para ser utilizados en la definición de arquitectura de la plataforma ZERA. Estos implementan una serie de características factibles para el desarrollo de sistemas web mejorando en organización de la estructura del código, mayor flexibilidad y adaptabilidad al cambio, reducción del tiempo de mantenimiento, entre otras que se ajustan a la plataforma a desarrollar:

- **Modelo Vista Controlador (MVC):** Divide una aplicación interactiva en tres componentes. El modelo contiene los datos y funcionalidades de fondo. La vista exhibe información para el usuario. Los controladores manipulan las entradas del usuario. La vista y el controlador conjuntamente comprenden la interfaz de usuario. Un mecanismo de propagación de cambio asegura consistencia entre la interfaz de usuario y el modelo.

- **Front Controller (Controlador Frontal)**: Este elemento provee un controlador centralizado para gestionar las peticiones web a la aplicación. Un controlador frontal recibe todas las peticiones entrantes de los clientes, remitiendo a su vez cada petición al gestor de peticiones adecuado, que se encargará de gestionar la construcción de una respuesta adecuada al cliente. Son los puntos ideales para implementar servicios de seguridad, tratamiento de errores, y la gestión del control para la generación de contenidos.

1.2.3.2 Patrones de Diseño

Los subsistemas de una arquitectura de software, así como la relación entre ellos usualmente consiste en pequeñas unidades arquitectónicas. Estas unidades se describen mediante los patrones de diseño.

Un patrón de diseño provee un diseño para refinar los subsistemas o componentes de un software, o la relación que existe entre ellos. Estos describen estructuras comúnmente recurrentes de componentes de comunicación que solucionan un problema general de diseño en un contexto particular [8].

Estos patrones están relacionados con el diseño de los objetos y frameworks de pequeña y mediana escala. Aplicables al diseño de una solución para conectar los elementos de gran escala que se definen mediante los patrones de arquitectura, y durante el trabajo de diseño detallado para cualquier aspecto del diseño local. También se conocen como patrones de micro-arquitectura [7].

La aplicación de los patrones de diseños no tienen efectos sobre la estructura fundamental de un sistema de software, pero pueden tener una fuerte influencia sobre la arquitectura de un subsistema. Estos hacen más fácil reutilizar con éxito los diseños y arquitecturas, ayudan a los diseñadores a reutilizar con éxito diseños para obtener nuevos diseños.

Los patrones de diseño tiene una series elementos que los caracterizan:

- El nombre del patrón, describe el problema de diseño, su solución, y consecuencias en una o dos palabras. Tener un vocabulario de patrones nos permite hablar sobre ellos.

- El problema describe cuando aplicar el patrón. Se explica el problema y su contexto. Puede describir estructuras de clases u objetos que son sintomáticas de un diseño inflexible. Se incluye una lista de condiciones.
- La solución describe los elementos que forma el diseño, sus relaciones, responsabilidades y colaboraciones. No se describe un diseño particular. Un patrón es una plantilla.
- Las consecuencias son los resultados de aplicar el patrón.

El catálogo de patrones más famoso es el contenido en el libro “Design Patterns: Elements of Reusable Object-Oriented Software”, el de la banda de los cuatro, también conocido como el libro GOF (Gang-Of-Four Book).

Según este libro existen tres tipos de patrones de diseño [10]:

- **Creación**: Es la encargada de crear objetos cuando sus creaciones requieren tomar decisiones.
- **Estructurales**: Describen la forma en que diferentes tipos de objetos pueden ser organizados para trabajar unos con otros.
- **Comportamiento**: Organiza, maneja y combina comportamientos.

Cada patrón de estos presenta una gama de patrones que los encapsula, cumpliendo con las características de estos patrones.

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

Tabla 1: Ejemplos de patrones de diseño en cada estilo

En sistemas orientados a objetos existen otro conjunto denominado patrones GRASP, los cuales describen los principios fundamentales de la asignación de responsabilidades a objetos. GRASP es un acrónimo que significa General Responsibility Asignment Software Patterns (Patrones Generales de Software para Asignar Responsabilidades). El nombre se eligió para indicar la importancia de captar (grasping) estos principios, si se quiere diseñar eficazmente el software orientado a objetos. A continuación se describen los patrones básicos de asignación de responsabilidades [7]

- **Experto**: La responsabilidad de realizar una labor es de la clase que tiene o puede tener los datos involucrados (atributos). Una clase, contiene toda la información necesaria para realizar la labor que tiene encomendada.
- **Creador**: Este patrón como su nombre lo indica es el que crea, el guía la asignación de responsabilidades relacionadas con la creación de objetos, se asigna la responsabilidad de que una clase B cree un Objeto de la clase A solamente cuando : B contiene a A , B es una agregación

(o composición) de A , B almacena a A , B tiene los datos de inicialización de A (datos que requiere su constructor) y B usa a A.

- **Bajo Acoplamiento**: El acoplamiento es una medida de fuerza con que un elemento está conectado a, tiene conocimiento de, confía en, otros elementos. Este patrón es un principio que asigna la responsabilidad de controlar el flujo de eventos del sistema, a clases específicas. Esto facilita la centralización de actividades (validaciones, seguridad, etc.) . El controlador no realiza estas actividades, las delega en otras clases con las que mantiene un modelo de alta cohesión.
- **Alta Cohesión**: La cohesión es una medida de la fuerza con la que se relacionan las clases y el grado de focalización de las responsabilidades de un elemento. Cada elemento del diseño debe realizar una labor única dentro del sistema, no desempeñada por el resto de los elementos y auto-identificable, una clase con baja cohesión hace muchas cosas no relacionadas o hace demasiado trabajo.
- **Controlador:** Es un evento generado por actores externos. Se asocian con operaciones del sistema, como respuestas a los eventos de este, tal como se relacionan los mensajes y los métodos. Normalmente un controlador delega en otros objetos el trabajo que se necesita hacer; coordina o controla la actividad. No realiza mucho trabajo por sí mismo.

1.3 Plataformas de Gestión del Aprendizaje

Las plataformas de gestión de aprendizaje o plataformas de teleformación, cuyo término alternativo son: ambientes virtuales de aprendizaje (AVA), plataformas educativas [11] y entorno virtual de enseñanza/aprendizaje (EVE/A), son aplicaciones informáticas diseñadas para facilitar la comunicación pedagógica entre los participantes en un proceso educativo, sea este completamente a distancia, presencial, o de una naturaleza mixta que combine ambas modalidades en diversas proporciones [12]. Sus orígenes se deben a una especialización de los CMS (Content Management System) en sistemas orientados a la gestión de contenidos para el aprendizaje.

Cada vez son más usadas las plataformas de gestión del Aprendizaje por las ventajas que proponen a todos los participantes en un proceso de enseñanza-aprendizaje. En la actualidad existen infinidad de estos sistemas. Mientras que algunos están basados en código abierto, otros son soluciones propietarias, y para utilizarlos, los usuarios deben pagar para adquirir las licencias correspondientes. Entre los principales exponentes de los LMS basados en código abierto se encuentran: *Moodle*, *Dokeos*, *Claroline*, *Bazaar*, *Atutor*, *Sakai*, *Ilias* y *Spaghetti Learning* y entre las propietarias sobresalen por su relevancia: *BlackBoard*, *eCollege* y *Desire2Learn*.

También en Cuba se han desarrollado soluciones homólogas entre las que aparecen *aprenDIST*, desarrollada en el Instituto Superior Politécnico "José Antonio Echeverría" (CUJAE) para esta misma institución y con fines comerciales; y *SadHEA-WEB,* desarrollado por el Centro de Estudios de Software y sus Aplicaciones Docentes (CESOFTAD) de la Universidad de Ciencias Pedagógicas "José de la Luz y Caballero" de Holguín. Dentro de la Universidad de las Ciencias Informáticas no se tiene evidencia de que se haya desarrollado ningún sistema de este tipo, sólo personalizaciones usando como base la plataforma Moodle, ejemplo de ellas el Entorno Virtual de Aprendizaje (EVA) usado en la misma universidad y el Ambiente Digital de Aprendizaje (ADA) localizado para Venezuela. Además se han realizados módulos de apoyo a esta plataforma y otras para la gestión y almacenamiento de objetos de aprendizaje.

1.4 Arquitecturas de Plataformas de Gestión del Aprendizaje

Después de haber realizado un estudio sobre la Arquitectura de Software y sus principales características, definiciones y elementos que la componen se está en condiciones de analizar las principales cualidades arquitectónicas de los LMS más usados en el mundo.

En una búsqueda inicial fue posible constatar que los creadores de LMS con licencias comerciales no brindan información relacionada con el desarrollo de sus productos y herramientas. Seguidamente se dio paso a las plataformas de carácter libre. También se infiere que el tema en cuestión ha sido pobremente abordado, pues en la bibliografía consultada en las bases de datos referenciales y especializadas sobre el tema, no se encontraron muchas evidencias sobre el tema.

A continuación se expondrá el estudio realizado en base a características comunes de estos sistemas y sobre dos de los LMS libres más usados a nivel internacional.

La mayoría de las plataformas están subdivididas en módulos y subsistemas independientes que pueden ser agregados en diferentes momentos para adicionar funcionalidades a la plataforma. Comúnmente, un sistema de enseñanza virtual se compone de varios subsistemas principales que interaccionan entre ellos:

- **Subsistema de gestión y distribución de contenidos.** Permiten almacenar, organizar, recuperar y distribuir contenidos educativos y estructurarlos en contenidos de mayor complejidad y alcance temático.
- **Subsistema de administración de usuarios.** Facilitan el registro de los usuarios del sistema para el posterior control de acceso y presentación personalizada de los contenidos y cursos.
- **Subsistema de comunicación.** Chats, foros, correo electrónico, tableros de anuncios, permiten la comunicación entre estudiantes y tutores en una vía o en doble vía, sincrónica y asincrónicamente.
- **Subsistema de evaluación y seguimiento.** Apoyan la construcción y presentación de evaluaciones mediante la utilización de diferentes tipos de preguntas: abierta, falso o verdadero, selección múltiple, múltiple opción, completar y apareamiento entre otras. Algunas veces también permite la construcción de bancos de preguntas usados con frecuencia para seleccionar aleatoriamente preguntas para los estudiantes.

1.4.1 Claroline

Claroline es un sistema de administración del aprendizaje. Está diseñado para ayudar a los docentes e instructores a crear contenidos educacionales y supervisar las actividades de aprendizaje en la web. Permite, tanto a estudiantes como docentes, organizar el proceso de aprendizaje, comunicarse entre ellos, autoevaluarse y manejar un calendario de actividades [13]. Este LMS posibilita además la estructuración de cursos, colocación de material y administración de actividades. Da libertad a los docentes para

estructurar y colocar su material de estudio. El proceso de aprendizaje individual como está contemplado en esta herramienta utiliza lecturas y/o documentos en formatos diferentes.

Claroline ha sido desarrollado principalmente para apoyar un buen proceso de enseñanza y aprendizaje, no para sustituirlo. Busca ser útil, guiado por los requerimientos de los usuarios, abierto para permitir integrar servicios existentes y nuevas herramientas, adaptable a escenarios concretos para los cursos [13].

Debido a que Claroline es un software de código abierto, licenciado bajo GPL y basado en PHP3 y MySQL4, y modular, permite a un administrador agregar y modificar herramientas, cambiar la disposición, adaptar bases de datos, etc. Está traducida en treinta y cinco idiomas, lo que la convierte en una de las más usadas internacionalmente. Esta aplicación fue laureada por el premio UNESCO 2007.

Las principales características de Claroline son las siguientes:

- Gestión de agenda y anuncios, estos últimos con posibilidad de enviarse por correo a todos los usuarios
- Sistema de enlaces de hipertexto.
- Gestión de documentos de forma jerárquica -carpetas-.
- Sistema para que los alumnos envíen trabajos al profesor del curso a través de la web.
- Sistema de navegación de usuarios.
- Sistema de foros.
- Creación de ejercicios de Autoevaluación para alumnos (test, preguntas cortas, etc).
- Creación de grupos de alumnos (los cuales tienen foros y áreas de documentos individuales).
- Chat basado en texto, para lo comunicación sincrónica y asincrónica con el docente y otros alumnos.
- Posibilidad de añadir otras páginas en HTML propias o enlaces externos en la página principal.

- Potente sistema de estadísticas.

Brinda la posibilidad de ser configurada en varios idiomas. El multilenguaje es incorporado en cada módulo mediante unos archivos de lenguaje que contienen todas las cadenas de mensajes que se despliegan en cada uno de estos. Estos archivos se agrupan por carpetas dependiendo del idioma, todas las carpetas de idioma se agrupan en la dirección *claroline**lang*.

Las hojas de estilo (CSS) se agrupan en la carpeta *claroline**css*.

Este LMS posee una Interfaz de Programación de Aplicaciones o API (del inglés Application Programming Interface) que contiene todas las funciones necesarias para interactuar con la base de datos en las diferentes secciones del sistema, y desplegar tablas y texto acorde a las hojas de estilo que posea, al igual que poder desplegar la cabecera y el pié de la plataforma en cada uno de los módulos, dichas funciones se encuentran contenidas en los archivos de la carpeta *in**lib*\.

Los módulos agrupan funcionalidades y herramientas que aparecen en los cursos, tanto de administración como los que aparecen visibles para el alumno, todos ellos se encuentran agrupados en la carpeta *claroline*\. Las extensiones (o plugins) se agrupan en el directorio *claroline**plugins* donde cada carpeta representa a cada plugin.

La autenticación se gestiona mediante un módulo, el cual es el encargado de mostrar el formulario de inicio de sesión y de registrar las correspondientes variables de sesión; este se encuentra en el archivo *index.php* y en *claroline**auth*, donde se encuentran los scripts encargados del registro de usuarios y recuperación de contraseñas.

Por defecto Claroline crea varias bases de datos (dos para el programa principal y una para cada curso creado). Sin embargo, se puede indicar en el programa de instalación la opción de sólo una base de datos.

1.4.2 Moodle

Moodle es un completo sistema de administración de cursos. Su nombre es el acrónimo de Modular Object-Oriented Dynamic Learning Environment (Entorno de Aprendizaje Dinámico Orientado a Objetos y Modular). Funciona en servidores web con PHP combinado con MySQL, traducida a más de setenta idiomas y se distribuye gratuitamente como Software libre bajo la Licencia pública GNU. Ha sido desarrollado sobre Linux, Windows, y Mac OS X. Moodle también usa la librería ADOdb[1] para la abstracción de bases de datos, lo que significa que puede usar más de diez marcas diferentes de bases de datos (desafortunadamente, a pesar de ello, no puede aún crear tablas en todas esas bases de datos).

Algunas de sus principales características son:

- Tiene una interfaz de navegador de tecnología sencilla, ligera, eficiente y compatible.
- Es fácil de instalar en casi cualquier plataforma que soporte PHP. Sólo requiere que exista una base de datos (y la puede compartir).
- Con su completa abstracción de bases de datos, soporta las principales marcas de bases de datos (excepto en la definición inicial de las tablas).
- La lista de cursos muestra descripciones de cada uno de los cursos que hay en el servidor, incluyendo la posibilidad de acceder como invitado.
- Los cursos pueden clasificarse por categorías y también pueden ser buscados, un sitio Moodle puede albergar miles de cursos.
- Se ha puesto énfasis en una seguridad sólida en toda la plataforma. Todos los formularios son revisados, las cookies encriptadas, etc.
- La mayoría de las áreas de introducción de texto (recursos, mensajes de los foros etc.) pueden ser editadas usando el editor HTML, tan sencillo como cualquier editor de texto de Windows.
- Soporta un rango de mecanismos de autenticación a través de módulos de autenticación, que permiten una integración sencilla con los sistemas existentes.

1 Es un conjunto de bibliotecas de bases de datos para PHP y Python

- Método LDAP: las cuentas de acceso pueden verificarse en un servidor LDAP.
- IMAP, POP3, NNTP: las cuentas de acceso se verifican contra un servidor de correo o de noticias (news). Soporta los certificados SSL y TLS.
- Seguridad: los profesores pueden añadir una "clave de matriculación" para sus cursos, con el fin de impedir el acceso de quienes no sean sus estudiantes. Pueden transmitir esta clave personalmente o a través del correo electrónico personal, etc.
- Cada usuario puede elegir el idioma que se usará en la interfaz de Moodle (inglés, francés, alemán, español, portugués, etc.).
- Ofrece una serie flexible de actividades para los cursos: foros, glosarios, cuestionarios, recursos, consultas, encuestas, tareas, Chat y talleres.

Moodle sabe cuál es su versión (así como las versiones de todos los módulos) y se ha construido un mecanismo interno para que pueda actualizarse a sí misma de forma apropiada a las nuevas versiones (por ejemplo, puede renombrar las tablas de las bases de datos o añadir nuevos campos). Tiene una serie de características modulares, incluyendo temas, actividades, interfaces de idioma, esquemas de base de datos y formatos de cursos. Esto le permite a cualquier desarrollador añadir características al código básico principal o incluso distribuirlas por separado.

Una vez analizado las principales características arquitectónicas de los LMS, se puede concluir que la plataforma ZERA debe estar compuesta por varios subsistemas estrechamente relacionados, los cuales integren todas las funcionalidades que debe cubrir. Estos subsistemas deben poseer características modulares por las ventajas que esta presenta, observadas en la plataforma Moodle.

Otros temas que se deben definir es el uso de correos como vía de comunicación entre los partícipes del proceso de enseñanza-aprendizaje implicados en la plataforma, así como, la disponibilidad de varios lenguajes los cuales brinden más facilidades a los usuarios finales.

1.5 Entornos de Desarrollo

Los entornos de desarrollo de software son herramientas que ayudan a los programadores a desarrollar software sobre ambientes más amigables. Es decir, aquellos en los que el programador puede acceder con el menor esfuerzo a diferentes recursos como editores, compiladores, herramientas de análisis, etc [14]. A continuación se exponen las principales características de herramientas y tecnologías libres a utilizar.

1.5.1 Metodologías de desarrollo de software

El desarrollo de software no es una tarea fácil, lo demuestra la existencia de numerosas metodologías que influyen de diferentes formas en el proceso de desarrollo.

Una metodología es un conjunto de procedimientos, técnicas, herramientas y documentos auxiliares que ayuda a los desarrolladores a realizar un nuevo software. Seleccionar la metodología adecuada garantiza la construcción de un software de calidad, desarrollarlo en el tiempo planificado y sobre todo con los costes establecidos.

1.5.1.1 Proceso Unificado de Desarrollo(RUP)

Es un proceso para el desarrollo de un producto de software que define claramente quién, cómo, cuándo y qué debe hacerse en el proyecto. Proporciona un conjunto de técnicas que soportan el ciclo completo de desarrollo de software, cumpliendo con los requerimientos de los usuarios dentro de una planificación y presupuesto establecido. Está constituido por nueve flujos de trabajo (los seis primeros son conocidos como flujos de ingeniería y los tres últimos de apoyo): modelamiento del negocio, requisitos, análisis y diseño, implementación, prueba, instalación, gestión de configuración y cambios, gestión de proyectos, ambiente, y estos tienen lugar sobre cuatro etapas o fases: inicio, elaboración, construcción y transición.

Características definitorias de RUP

- **Dirigido por casos de uso**: Los casos de uso reflejan lo que los usuarios futuros necesitan y desean, lo cual se capta cuando se modela el negocio y se representa a través de los requerimientos. A partir de aquí los casos de uso guían el proceso de desarrollo ya que los modelos que se obtienen como resultado de los diferentes flujos de trabajo representan la realización de los casos de uso.

- **Centrado en la arquitectura:** La arquitectura muestra la visión común del sistema completo en la que el equipo de proyecto y los usuarios deben estar de acuerdo, por lo que describe los elementos del modelo que son más importantes para su construcción, los cimientos del sistema que son necesarios como base para comprenderlo, desarrollarlo y producirlo económicamente.

- **Iterativo e incremental**: Donde cada fase se desarrolla en iteraciones, de forma tal que se pueda dividir en pequeños proyectos mejorando su comprensión y desarrollo. RUP propone que cada fase se desarrolle en iteraciones. Una iteración involucra actividades de todos los flujos de trabajo, aunque desarrolla fundamentalmente unos más que otros [15].

Modelo 4 +1 Vistas

Esta metodología describe la Arquitectura de Software mediante cinco vistas concurrentes propuestas en el Modelo 4+1 Vistas. Este modelo permite a los involucrados en el proyecto encontrar lo que necesitan en la Arquitectura de Software. Los clientes y especialistas pueden aproximarse a los datos de la vista lógica, los ingenieros de sistemas pueden inicialmente abordar la vista física e ir al proceso a continuación, los administradores de proyectos y las personas de configuración del software pueden enfocarse en la vista de desarrollo. Estas cuatro vistas están guiadas por la vista de casos de uso que describe las funcionalidades del sistema que más inciden sobre su arquitectura:

- **Vista de Casos de Uso:** Esta vista representa un subconjunto del artefacto Modelo de casos de uso y lista los casos de uso o escenarios más significativos, con las funcionalidades centrales del

sistema. Si el sistema se hace extenso entonces se debería organizar en paquete, lo cual facilitaría la comprensión de la vista de casos de uso.

- **Vista Lógica:** Esta vista representa un subconjunto del artefacto Modelo de diseño, en él se representan los elementos de diseño más significativos para la arquitectura del sistema. Describe las clases más importantes, su organización en paquetes y subsistemas, y estos a su vez en capas.

- **Vista de Implementación:** En esta sección se describe la estructura general del modelo de implementación, en correspondencia con la vista lógica, se debe representar la descomposición del software en capas y paquetes importantes para la arquitectura.

- **Vista de Despliegue:** Esta vista suministra una base para la comprensión de la distribución física de un sistema a través de nodos, es decir, modela la configuración en funcionamiento del sistema (software y hardware) y las relaciones entre sus componentes. Suele utilizarse cuando el sistema está distribuido.

- **Vista Concurrente o de Procesos:** Describe el diseño de concurrencia y aspectos de sincronización. Especifica las líneas de mando que ejecutan cada operación en cada una de las clases señaladas en la vista lógica. Los diseñadores realizan esta vista en varios niveles de abstracción, además de dividir el software en conjuntos independientes de tareas, es decir, se empaqueta en pequeños programas o librerías del subsistema.

1.5.1.2 Programación Extrema (XP)

La Programación Extrema es una metodología ligera de desarrollo de software que se basa en la simplicidad, la comunicación y la reutilización del código desarrollado.

Esta metodología trata de minimizar la complejidad de un proyecto y de enfocarse directamente hacia el objetivo, hace uso de las relaciones interpersonales y la rapidez de reacción. Se caracteriza por llevar a

cabo pruebas unitarias, basadas en pruebas hechas a los procesos de mayor importancia, también la refactorización y la programación en parejas. El ciclo de vida ideal de XP consta de 6 fases: exploración, planificación de la entrega, iteraciones, producción, mantenimiento y muerte del proyecto.

Propuestas de la Programación Extrema

- Inicia desde la base o lo pequeño y agrega funcionalidades con retroalimentación continua.
- El cliente final forma parte del equipo de trabajo.
- El control del cambio es una de sus bases fundamentales.
- No introduce funcionalidades antes que sean necesarias.

Algunos de los inconvenientes de esta metodología es que la constante refactorización[2] trae consigo que cuando se usan diagramas UML estos generalmente tienden a estar poco actualizados, y otra de las cosas que más se menciona de los proyectos con XP es que es difícil predecir costos y tiempo de desarrollo.

1.5.2 El Lenguaje Unificado de Modelado (UML)

UML es un lenguaje de modelado visual que permite visualizar, especificar, construir y documentar los artefactos del sistema de software [15].

Ofrece soporte para clases, clases abstractas, relaciones, comportamiento por iteración, empaquetamiento, entre otros. Estos elementos se pueden representar mediante nueve tipos de diagrama, que son: de clases, de objetos, de casos de uso, de secuencia, de colaboración, de estados, de actividades, de componentes y de desarrollo [16].

UML presenta características generales y razones por las que resulta interesante su aplicación para efectos de la representación de una arquitectura de software. Permite el soporte para algunos de los conceptos asociados a las arquitecturas de software, como los componentes, los paquetes, las librerías y

2 Es el proceso de reestructuración u optimización del código fuente sin cambiar su comportamiento.

la colaboración. Además, admite la descripción de componentes en la arquitectura de software en dos niveles; se puede especificar solo el nombre del componente o especificar las clases o interfaces que implementan estos.

1.5.3 Herramientas CASE

Las herramientas CASE (Computer Aided Software Engineering, Ingeniería de Software Asistida por Computadora) son variadas aplicaciones informáticas destinadas a la automatización o apoyo de una o más fases del ciclo de vida de desarrollo de software. Brindan al equipo de desarrollo una gama de componentes que facilitan tanto el desarrollo de prototipos como el modelado de sistemas.

- **ArgoUML:** Aplicación desarrollada en Java que permite crear diagramas UML, puede usarse en cualquier plataforma que soporte este lenguaje (Java). Además de incluir soporte para generar código en lenguajes como: Java, PHP, C++, Python, permite desde diagramas la generación de ficheros PNG, GIF, JPG.

  Un principal inconveniente de esta aplicación, es la carencia de soporte completo para algunos tipos de diagramas incluyendo los Diagramas de secuencia y los de colaboración.

- **Visual Paradigm:** Es una herramienta de diseño que soporta todos los diagramas UML, diagramas de SysML y diagrama entidad – relación. Visual Paradigm para UML ofrece amplias características de modelado de casos de uso incluyendo la función completa de UML, Diagrama de casos de uso, flujos de evento editor, caso de uso/ Red de actor y la generación de un diagrama de actividad.Visual Paradigm produce la documentación del sistema con diseñador de plantilla. Los analistas del sistema pueden estimar las consecuencias de los cambios en los diagramas de análisis de impacto, tales como la matriz y el diagrama de análisis.

  Esta herramienta permite además la generación de código Java y generar reportes en PDF, EXCEL y MS WORD.

1.5.4 Lenguajes de Desarrollo

Un lenguaje es un sistema de comunicación que tiene forma, contenido y uso. La programación es, en informática, el proceso de escritura del código fuente de un software. De esta forma, la programación le señala al programa informático qué tiene que hacer y cómo realizarlo.

1.5.4.1 Lenguajes del lado del cliente

Son los lenguajes que basan su procesamiento en el cliente web, es decir que se ejecutan en el navegador del usuario.

- **XHTML**: Es muy similar al lenguaje HTML. De hecho, XHTML no es más que una adaptación de HTML al lenguaje XML. Técnicamente, HTML es descendiente directo del lenguaje SGML (Lenguaje de Marcado Generalizado), mientras que XHTML lo es de XML (que a su vez también es descendiente de SGML). De forma sencilla, “HTML es lo que se utiliza para crear todas las páginas web”. Más concretamente, HTML es el lenguaje con el que se “escriben” la mayoría de las páginas web. El propio W3C define el Lenguaje de Marcado de Hipertexto como “un lenguaje reconocido universalmente y que permite publicar información de forma global”. Desde su creación, el lenguaje HTML ha pasado a ser un lenguaje utilizado exclusivo para crear documentos electrónicos a ser un lenguaje que se utiliza en muchas aplicaciones electrónicas como buscadores, tiendas en línea y banca electrónica [17].

- **CSS**: Es un lenguaje de hojas de estilo creado para controlar el aspecto o presentación de los documentos electrónicos definidos con HTML y XHTML. CSS (Hojas de estilo en cascada) es la mejor forma de separar los contenidos y su presentación y es imprescindible para crear páginas web complejas. Separar la definición de los contenidos y la definición de su aspecto presenta numerosas ventajas, ya que obliga a crear documentos HTML/XHTML bien definidos y con significado completo (también llamados “documentos semánticos”). Además, mejora la accesibilidad del documento, reduce la complejidad de su mantenimiento y permite visualizar el mismo documento en infinidad de dispositivos diferentes [18].

- **JavaScript**: Es un lenguaje de programación que se utiliza principalmente para crear páginas web dinámicas. Una página web dinámica es aquella que incorpora efectos como texto que aparece y desaparece, animaciones, acciones que se activan al pulsar botones y ventanas con mensajes de aviso al usuario. Técnicamente, JavaScript es un lenguaje de programación interpretado, por lo que no es necesario compilar los programas para ejecutarlos. En otras palabras, los programas escritos con JavaScript se pueden probar directamente en cualquier navegador sin necesidad de procesos intermedios [19].

- **XML**: Son las siglas en inglés de Extensible Markup Language (lenguaje de marcas extensible), una especificación/lenguaje de programación desarrollada por el W3C. XML es una versión de SGML, diseñado especialmente para los documentos de la web. Permite que los diseñadores creen sus propias etiquetas, permitiendo la definición, transmisión, validación e interpretación de datos entre aplicaciones y entre organizaciones [20].

- **Ajax**: En realidad el término es un acrónimo de Asynchronous JavaScript + XML, que se puede traducir como “JavaScript asíncrono + XML”. En el artículo publicado por Jesse James Garrett en el 2005 define Ajax como: “Ajax no es una tecnología en sí misma. Las tecnologías que forman Ajax son XHTML y CSS para crear una presentación basada en estándares, DOM para la interacción y manipulación dinámica de la presentación. XML, XSLT y JSON para el intercambio y la manipulación de información, XMLHttpRequest para el intercambio asíncrono de la información y JavaScript para unir todas las demás tecnologías [21].

1.5.4.2 Lenguaje del lado del servidor

Son los lenguajes que se procesan en el lado del servidor y que generan la página antes de enviarla al cliente.

- **Python:** Fue creado en el año 1990 por Guido van Rossum y surgió como sucesor del lenguaje ABC. El principal objetivo que persigue es la facilidad, tanto de lectura como de diseño. Su entorno de ejecución detecta muchos de los errores de programación que escapan al control de los

compiladores, esto proporciona información valiosa para detectar esos errores y corregirlos. Además de incluir varias bibliotecas estándar puede integrarse con otros lenguajes y herramientas.

- **PHP:** Es un lenguaje de script interpretado en el lado del servidor utilizado para la generación de páginas web dinámicas, similar al ASP de Microsoft o el JSP de Sun, embebido en páginas HTML y ejecutado en el servidor. La mayor parte de su sintaxis ha sido tomada de C, Java y Perl con algunas características específicas de sí mismo. La meta del lenguaje es permitir rápidamente a los desarrolladores la generación dinámica de páginas. No es un lenguaje de marcas como podría ser HTML, XML o WML. Al ser un lenguaje libre dispone de una gran cantidad de características que lo convierten en la herramienta ideal para la creación de páginas web dinámicas:
    - Ofrece una solución simple y universal para las paginaciones dinámicas del web de fácil programación.
    - Integración con varias bibliotecas externas, permite generar documentos en PDF (documentos de Acrobat Reader) hasta analizar código XML.
    - Soporte para una gran cantidad de bases de datos: MySQL, Oracle, PostgreSQL, MS SQL Server, entre otras.
    - Perceptiblemente más fácil de mantener y poner al día que el código desarrollado en otros lenguajes.
    - Soportado por una gran comunidad de desarrolladores, como producto de código abierto, PHP goza de la ayuda de un gran grupo de programadores que permite que los fallos de funcionamiento se encuentren y reparen rápidamente.
    - El código se pone al día continuamente con mejoras y extensiones de lenguaje para ampliar las capacidades de PHP, a partir de la versión 5.0.0, se incluyeron funcionalidades que permiten aprovechar las facilidades de la programación orientada a objetos.

- Con PHP se puede hacer cualquier cosa que se pueda realizar con un script CGI, como el procesamiento de información en formularios, blogs y foros de discusión, manipulación de cookies y páginas dinámicas [22].

1.5.5 Marcos de Desarrollos (Framework)

Un framework es una estructura de soporte definida en la cual otro proyecto de software puede ser organizado y desarrollado. Puede incluir soporte de programas, librerías y un lenguaje de script entre otros software para ayudar a desarrollar y unir los diferentes componentes de un proyecto. Proporciona una estructura al código y hace que los desarrolladores escriban código mejor, más entendible y mantenible, hace la programación más fácil, convirtiendo complejas funciones en sencillas instrucciones.

1.5.5.1 Framework para el cliente

La mayoría de los frameworks Javascript proveen componentes totalmente compatibles entre los distintos navegadores, esto aumenta la portabilidad de la aplicación. Poseen capacidad de manipulación del DOM, manejo de XML, CSS, JSON, eventos y otras características para aumentar la interactividad del usuario. Entre ellos se destacan:

- **Dojo Toolkit:** Está compuesto por widgets que son componentes de código en Javascript pre-empaquetados que pueden ser utilizados para enriquecer sitios web con varias características que trabajan a través de la mayoría de los navegadores, tales como: menús, tabs, tooltips y tablas ordenables.
- **ExtJS:** Incluye interoperabilidad con jQuery. Posee controles para campos de textos, incluyendo áreas de texto, controladores selectores de fecha, campos numéricos para radiobutton y checkbox. También componentes para crear y manipular datagrids donde goza de cierta ventaja sobre otros frameworks.
- **jQuery:** Es muy rápida, ligera y simplifica el desarrollo de la parte del cliente de las aplicaciones web. Es un nuevo tipo de bibliotecas de Javascript que permite simplificar la manera de interactuar

con los documentos HTML. Existen gran número de plugins que extienden su funcionalidad y cualquiera puede crear sus propios plugins. Además, tiene un módulo de widgets que proporciona componentes predefinidos y efectos visuales a la interfaz de usuario.

1.5.5.2 Framework para el servidor

- **Zend Framework:** Usa código 100% orientado a objetos y la estructura sus componentes es algo único; cada componente está construido con una baja dependencia de otros componentes. Esta arquitectura débilmente acoplada permite a los desarrolladores utilizar los componentes por separado. Aunque se pueden utilizar de forma individual, los componentes de la biblioteca estándar de Zend Framework conforman un potente y extensible framework de aplicaciones web al combinarse. Ofrece un gran rendimiento, una robusta implementación MVC y una abstracción de base de datos fácil de usar.

- **Symfony:** Es un framework PHP que facilita el desarrollo de las aplicaciones web. Se encarga de todos los aspectos comunes de las aplicaciones, dejando que el programador se dedique a aportar valor desarrollando las características únicas de cada proyecto [23]. Además Symfony posee una extensa documentación, ya que cuenta con miles de páginas de documentación distribuidas en varios libros gratuitos y decenas de tutoriales.

  Está desarrollado solamente con PHP 5 para aprovechar las principales características de PHP y obtener su máximo rendimiento. Es multiplataforma y compatible con la mayoría de los gestores de base de datos como: Postgres, MySQL, Oracle y SQL Server de Microsoft. Es flexible y extensible mediante un completo mecanismo de plugins. Además de adaptarse a las políticas y arquitecturas propias de cada empresa, es lo suficientemente estable como desarrollar aplicaciones a largo plazo.

  Symfony se basa en el patrón MVC, toma lo mejor de la arquitectura que este propone y la implementa de forma que el desarrollo de aplicaciones sea rápido y sencillo. Está diseñado con el

patrón Controlador Frontal, el cual es el punto de entrada a la aplicación, carga la configuración, verifica las restricciones de seguridad y determina la acción a ejecutarse.

Para el desarrollo de nuevos sistemas se debería utilizar Symfony 1.4, que es la última versión estable y ha eliminado las características obsoletas de las versiones anteriores. Además de ser rápida y limpia, tiene soporte por el equipo de desarrollo por 3 años (hasta finales del 2012).

1.5.6 Ambiente de desarrollo integrado (IDE)

- **NetBeans IDE:** Es una herramienta para programadores con el objetivo de escribir, compilar y depurar programas. El IDE NetBeans es un producto libre y gratuito sin restricción de uso. Brinda soporte nativo a los frameworks Symfony y jQuery. Permite a los desarrolladores crear rápidamente aplicaciones utilizando lenguajes como Java, PHP, C / C++, Javascript, JavaFX.

  La integración con Symfony garantiza el desarrollo de sistemas de forma sencilla, esto es posible porque desde el mismo IDE se pueden ejecutar todas las tareas de Symfony, incluyendo los comandos para crear proyectos y aplicaciones.

1.5.7 Sistemas Gestores de Base de Datos (SGBD)

- **MySQL:** Es uno de los SGBD más usado en todo el mundo. Proporciona sistemas transaccionales y no transaccionales de almacenamiento transaccionales. Características como fiabilidad, facilidad y sobre todo velocidad elimina los problemas más importantes relacionados con el tiempo de inactividad. Es multiplataforma y presenta API disponibles para Java, C / C++, PHP y otros.

  Gracias a la colaboración de muchos usuarios, la base de datos se ha ido mejorando optimizándose en velocidad. Por eso es una de las bases de datos más usadas en Internet.

- **PostgreSQL:** Es un poderoso sistema de base de datos objeto-relacional de código abierto. Cuenta con más de quince años de desarrollo activo y una arquitectura probada que se ha ganado

una sólida reputación de confiabilidad, integridad de datos y corrección. Funciona en todos los principales sistemas operativos, incluyendo Linux, UNIX (AIX, BSD, HP-UX, SGI IRIX, Mac OS X, Solaris, Tru64), y Windows. Es totalmente compatible con ACID, tiene soporte completo para claves foráneas, uniones, vistas, disparadores y procedimientos almacenados (en varios idiomas). También es compatible con el almacenamiento de objetos binarios, incluyendo imágenes, sonidos o vídeos. Tiene interfaces de programación nativo de C/C++, Java, NET, Perl, Python, Ruby, Tcl, ODBC, entre otros, y la documentación de carácter excepcional. Es altamente escalable, tanto en la enorme cantidad de datos que puede manejar como en el número de usuarios concurrentes que puede permitir. Hay sistemas activos de PostgreSQL en entornos de producción que manejan más de cuatro terabytes de datos. Algunos de los límites generales de PostgreSQL se incluyen en la tabla de abajo.

Es una base de datos empresarial multiplataforma altamente escalable, tanto en la enorme cantidad de datos que puede manejar y en el número de usuarios concurrentes que puede acomodar. Es un sistema que cuenta con sofisticadas funciones de replicación asincrónica, backup en línea, permite consultas muy complejas y tiene un sofisticado optimizador para estas, es compatible con un conjunto de caracteres internacionales, cuenta con vistas, integridad referencial, triggers y posee control de versionado concurrente.

PostgreSQL 8.4 incorpora nuevos operadores y el tipo de datos XML, esto permite operar con documentos XML como lo que son: documentos de texto estructurado, por lo que se puede comprobar fácilmente si un valor de tipo XML está bien formado. Además, se ha adaptado el gestor para trabajar a alta velocidad en escenarios de alta carga de trabajo.

1.5.8 Servidor Web

Apache: es el servidor web por excelencia, con algo mas de un 60% de los servidores de internet confiando en él, por su robustez y estabilidad. Algunas de sus características son:

- **Fiabilidad**: Alrededor del 90% de los servidores con más alta disponibilidad funcionan con Apache.
- **Gratuidad**: Apache es totalmente gratuito, y se distribuye bajo la licencia Apache Software License, que permite la modificación del código.
- **Extensibilidad**: Se pueden añadir módulos para ampliar las ya de por si amplias capacidades de Apache. Hay una amplia variedad de módulos, que permiten desde generar contenido dinámico (con PHP, Java, Perl, Python...), monitorizar el rendimiento del servidor, atender peticiones encriptadas por SSL, hasta crear servidores virtuales por IP o por nombre (varias direcciones web son manejadas en un mismo servidor) y limitar el ancho de banda para cada uno de ellos. Dichos módulos incluso pueden ser creados por cualquier persona con conocimientos de programación.

Lighttpd: servidor web diseñado para ser rápido, seguro, flexible, y fiel a los estándares. Está optimizado para entornos donde la velocidad es muy importante, y por eso consume menos CPU y memoria RAM que otros servidores. Por todo lo que ofrece, lighttpd es apropiado para cualquier servidor que tenga problemas de carga. Además, tiene soporte para PHP, Python, Ruby y otros, permite crear servidores virtuales y cifrado SSL. Se puede integrar con módulos externos lo que garantiza que puedan usarse lenguajes en prácticamente cualquier lenguaje de programación.

1.5.9 Selección de las Herramientas y Tecnologías.

Se propone trabajar con tecnología web, por las ventajas que proporciona en cuanto a soporte, disponibilidad y eficiencia, pues facilita la actualización y mantenimiento de aplicaciones web sin distribuir e instalar software a miles de usuarios potenciales, y además, la estructura de almacenamiento y centralización de la información, permite incrementar el número de usuarios.

Para realizar el proceso de selección se priorizarán las herramientas y tecnologías que sean de código abierto o pertenecientes al software libre y que contribuyan el desarrollo de aplicaciones web. Teniendo en cuenta lo anterior, se determinó:

Utilizar RUP como metodología de desarrollo, porque propone el modelo 4+1 vistas para la descripción de la arquitectura, este permite representar diferentes aspectos y características de los elementos significativos del software en diferentes vistas, y cada una brinda una descripción completa de un sistema desde una perspectiva particular. Además, los clientes potenciales de la aplicación en cuestión no forman parte del equipo de trabajo.

Se decide trabajar con la herramienta CASE Visual Paradigm, esta utiliza UML como lenguaje de modelado y soporta todos sus diagramas. Además de tener bastante documentación en Internet, es multiplataforma y posee buena aceptación en la comunidad de desarrollo de software.

Como lenguaje de desarrollo se seleccionó PHP, pues cuenta con abundante documentación, incorpora las buenas prácticas de la programación orientada a objetos, y lo apoya una enorme comunidad que le aporta funcionalidades y nuevas soluciones.

Tomando PHP como punto de partida, se selecciona Symfony como marco de trabajo del lado del servidor, está entre los mejores documentados, es fácil de aprender, es libre para uso comercial, incorpora buenas prácticas de desarrollo de software mediante el uso de patrones como MVC y Controlador Frontal.

El marco de trabajo del lado del cliente que se decide utilizar es jQuery, por ser el más conocido por el equipo de desarrollo, es eficiente e incorpora componentes predefinidos y efectos visuales que aumentan la interactividad del usuario.

Apache es la elección principal como servidor web, porque es totalmente gratuito, soporta la mayoría de los lenguajes de desarrollo del lado del servidor, gestores de base de datos, y encabeza la lista de los servidores web más utilizados. Se propone además el servidor Lighttpd, este brinda opciones de balanceo de carga, es compatible con muchas herramientas y está optimizado para entornos donde la velocidad es muy importante.

El gestor de base de datos que se escoge es PostgreSQL, por estar entre los primeros gestores de código abierto, incorpora funcionalidades y mejoras que le permiten trabajar a alta velocidad en escenarios de alta carga de trabajo.

El IDE Netbeans satisface las principales necesidades del equipo de desarrollo, se integra a la gestión de configuración y a las herramientas de control de versiones, es un producto libre y gratuito, sin restricción de uso y brinda soporte nativo a los frameworks previamente seleccionados, jQuery y Symfony.

A partir de lo anterior, queda resumido el perfil tecnológico de la siguiente forma:

- **Metodología de desarrollo**: RUP
- **Herramienta CASE:** Visual Paradigm
- **Lenguaje de programación:** PHP 5.0 o superior
- **Framework del lado del cliente:** jQuery-1.5.2
- **Framework del lado del servidor:** Symfony 1.4.11
- **Ambiente de desarrollo integrado:** Netbeans IDE 6.8 o superior
- **Gestor de base de datos:** PostgreSQL 8.4
- **Servidor web:** Apache 2.0 y Lighttpd 1.4

Capítulo 2: Descripción de la arquitectura

2.1 Introducción

Cuando se diseña un sistema informático es importante que los involucrados dominen o tengan claro los principales conceptos y relaciones con los que deberá trabajar la aplicación. Por esta razón, en el presente capítulo se realiza la descripción de la arquitectura propuesta. Se identifican además los requisitos no funcionales y los casos de uso arquitectónicamente significativos que guiarán el desarrollo del sistema.

En el capítulo se realizan los Diagramas de Casos de Uso, estos muestran las relaciones existentes entre los actores del sistema y las secuencias de acciones en las que están involucrados. También, se realiza el diseño de la aplicación y otros aspectos relevantes que tributarán a la arquitectura del software.

2.2 Descripción del Sistema Propuesto

Teniendo en cuenta el estudio realizado hasta el momento, se determinaron las principales características y funcionalidades que formarán parte del sistema. Fueron definidos seis subsistemas: *administration*, *learning*, *content*, *resources*, *reports* y *bachelor*, los cuales serán los encargados de encapsular las funcionalidades que debe cumplir la plataforma, basados en tecnologías web, desarrollados con herramientas libres y capaces de ejecutarse en cualquier plataforma.

El subsistema de administración (*administration)* es el encargado de toda la gestión de usuarios, roles, permisos y acciones, además de la gestión de las escuelas, de las réplicas y de la instalación de la plataforma. También debe ser capaz de administrar y configurar los elementos que se gestionan en el resto del sistema, así como garantizar la seguridad e integridad de la misma.

Los principales procesos del subsistema de administración del aprendizaje (*learning)* están relacionados con las comunicaciones en la plataforma como son: la gestión del foro, la mensajería, la agenda, las

noticias, los avisos, los anuncios, además de la gestión de los sistemas educativos, los períodos lectivos así como la configuración de la plataforma en las escuelas.

El subsistema de gestión de materias (*content)* tiene como responsalibilidad fundamental agrupar las funcionalidades correspondientes a la creación cada materia (hiperentorno) y su respectivo macro índice, de manera que se pueda garantizar su correcta administración, permitiendo de esta forma la creación de contenidos que luego serán mostrados y utilizados por los docentes y estudiantes en cada uno de los hiperentornos.

Los recursos interactivos son un apoyo fundamental e indispensable para enriquecer los contenidos que se crean de las diferentes materias y que luego son visualizados en los hiperentornos. Estos están constituidos por: páginas, cuestionarios, recursos interactivos, medias, recursos estructurales, materiales del docente y glosario de términos. Todos estos elementos anteriormente mencionados son gestionados en el subsistema *de gestión de recursos (resources),* donde se pueden asociar los recursos a cada una de las materias.

El subsistema de gestión de reportes (*reports)* permite obtener un grupo de informes, con el objetivo de supervisar las acciones llevadas a cabo por otros usuarios en la plataforma.

Los hiperentornos son la base principal de la actividad académica, todo lo gestionado a través del resto de los subsistemas confluye en un hiperentorno. Precisamente de esto se encarga el subsistema bachiller (bachelor), el cual además proporciona a los estudiantes de una metáfora de trabajo similar a los libros de textos tradicionales, permite la toma de notas, mediante el resalte de textos y brindando la posibilidad de búsquedas y modificación de estas. También propone funcionalidades para las prácticas y simulaciones, lo que posibilita a docentes y estudiantes, crear y desarrollar respectivamente actividades de evaluación y autoevaluación.

2.3 Requisitos del Software

Los requerimientos para un sistema de software determinan lo que hará el sistema y definen las restricciones de su operación e implementación.

2.3.1 Requisitos No Funcionales (RNF)

Los requerimientos no funcionales son propiedades o cualidades que el producto debe tener. Representan las características del producto a desarrollar.

2.3.1.1 Seguridad

La plataforma y sus subsistemas deberán cubrir los siguientes elementos de seguridad:

- Base de datos de elementos multimedia y contenidos: encriptada.
- Acceso mono-usuario a la plataforma, estableciendo sesiones de trabajo.

La plataforma debe asegurar en todo momento, la seguridad de los contenidos:

- Asegurando el encriptado de algunos de los contenidos, dentro de los cuales se encuentran las imágenes, documentos, archivos, animaciones swf y applets.
- Manteniendo la individualidad del contenido y datos para los estudiantes independientes, así como para los grupos. Es decir, un estudiante no sabrá si se le ha asignado una tarea diferente que a sus compañeros, ni un grupo sabrá del trabajo de otro grupo.

2.3.1.2 Usabilidad

Los elementos gráficos como los iconos deberán contar con un tooltype o mensaje flotante que señalen el tipo de recurso al que se refiere.

2.3.1.3 Eficiencia

Cuando un usuario, sin importar su rol, permanece inactivo durante 20 minutos, se cierra la sesión automáticamente.

2.3.1.4 Especificaciones de los servidores

Servidor Central	Administración Gestión del Aprendizaje Gestión de Materias Gestión de Recursos Bachiller Gestión de Reportes
Servidor Local	Gestión del Aprendizaje Gestión de Recursos Bachiller Gestión de Reportes

Tabla 2: Distribución de los subsistemas en los servidores

Los servidores deberán cubrir las siguientes características o contar con una variante equivalente a:

Tipo de Servidor	Cantidad	Especificaciones
Servidores web	2 o más	**Procesador**: Intel Xeon de 4 núcleos. La cantidad de núcleos depende de la cantidad de usuarios conectados. (Ejemplo: 80,000 usuarios – 7 núcleos) **RAM**: 8 GB o más, con posibilidades de expansión en caso de ser necesario.
Servidor de Base de Datos	2	**Procesador**: Intel Xeon de 4 núcleos. La cantidad de núcleos depende de la cantidad de usuarios conectados. (Ejemplo: 80,000 usuarios – 7 núcleos) **RAM**: 8 GB o más, con posibilidades de expansión en caso de ser necesario.
Servidor de Almacenamiento	1	300 GB, inicialmente para el almacenamiento de todas las medias.

Tabla 3: Especificaciones de los Servidores centrales

Servidores Locales

En las escuelas en dependencia de la cantidad de usuarios pudiese elegirse usar un solo servidor para la web, Base de Datos y Almacenamiento, o de forma separada en dos: uno para la web y otro para la BD.

Pueden ser servidores normales, o máquinas mejoradas en caso de escuelas pequeñas de menos de 200 usuarios:

- 1 Procesador CoreDuo
- 2GB de RAM
- 120 GB de HDD mínimo, aunque puede ser configurable de acuerdo a la cantidad de contenidos que se vayan contratando.

Equipos de usuario final

Los usuarios finales deberán contar como mínimo con:

- Procesador Pentium II o superior.
- 1GB de RAM.
- 20GB de HDD.
- Bocinas.

Si no cuentan con servidor local: conexión de banda ancha 256 bps como mínimo.

Si cuenta con servidor local: acceso a red interna.

2.3.1.5 Soporte

La plataforma y sus contenidos deben cumplir con los siguientes puntos de accesibilidad:

- Ser generado en tecnología web para ser accesible a través de Internet.
- Ejecutar sobre cualquier navegador, siendo como mínimo que sea compatible con:
    - Internet Explorer 7.0 o superior.
    - Mozilla Firefox 3.6 o superior.
    - Opera 10.6 o superior.

- Chrome 10 o superior.
- Safari 5 o superior.

2.3.1.6 Restricciones de diseño

Las tecnologías y lenguajes utilizados para la integración de la plataforma se especifican a continuación:

Nombre y versión	Sitio web	Incluida[3]	Distribuida[4]
Symfony v1.4	http://www.symfony-project.org	Sí	Sí
Postgres SQL v8.4	www.postgresql.org	Sí	Sí
PHP v5.2	http://www.php.net/	Sí	Sí
JQuery v1.5	http://jquery.com/	Sí	Sí
Java Development Kit v6u20	http://java.sun.com/	No	No
Java Runtime Environment v6u20	http://java.sun.com/	No	No
RED5 v0.7	http://osflash.org/red5	Sí	Sí
Apache v2.2	http://www.apache.org/	Sí	Sí
Ubuntu v10.04 LTS	http://www.ubuntu.com/	Sí	Sí
SymmetricDS v1.7	http://symmetricds.codehaus.org/	Sí	Sí

Tabla 4: Tecnologías y lenguajes utilizados

El sistema usará como motor de Base de Datos a PostgreSql. Siendo necesario la elaboración de una copia de rescate de las tablas para no afectar la información almacenada antes de poner en total funcionamiento al sistema.

3 Se refiere a si el componente forma parte del producto final, ya sea en forma de código fuente, como librería de enlace dinámico o framework.

4 Se refiere a las herramientas que deben ser distribuidas con el producto final, dígase un compresor de ficheros, un reproductor de video, un gestor de base de datos.

Se configurará un respaldo para la información de la plataforma en otro servidor que se ejecutará cada 60 minutos. Los respaldos en las escuelas se configurarán una vez a la semana en el propio servidor local. Los datos se encuentran respaldados por el proceso de réplica a nivel central.

2.4 Características utilizadas de Symfony

El proyecto consta de seis subsistemas o aplicaciones Symfony, *administration*, *learning*, *bachelor*, *content*, *resources y services.* En correspondencia con el alcance de la investigación, se describirán *administration,* en el que se realizan las principales acciones de gestión de la plataforma incluyendo la administración de usuarios, *learning* encargado de la gestión del aprendizaje, *content* donde se administran los contenidos educativos o materias, y *resources* encargado de gestionar los recursos interactivos, como imágenes, videos y otros.

Symfony utiliza el patrón decorador para la vista, en la que cada plantilla mostrada por la ejecución de cualquier acción es decorada con un layout común en cada aplicación. El proyecto contará con cinco layouts, el primero para decorar la página principal de la plataforma, uno por cada aplicación destinada a la gestión, y el último para decorar el asistente de instalación que se usará en la creación de las escuelas.

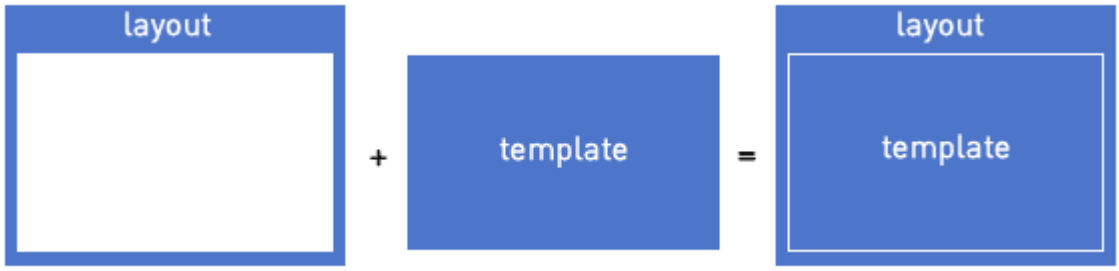


Fig. 2: Representación del patrón Decorador usado en Symfony

El framework Symfony tiene soporte para todas las base de datos soportadas por la capa de abstracción de base de datos que integra PHP (PDO). Para mapear[5] los registros de las tablas con los objetos, el ORM Doctrine genera tres clases por cada tabla en el modelo, una clase base que funciona como entidad,

5 Técnica de programación para convertir datos entre el sistema de tipos utilizado en un lenguaje de programación orientado a objetos y el utilizado en una base de datos relacional.

la cual posee los atributos, relaciones y restricciones de la tabla, no se puede personalizar porque es sobrescrita cada vez que se genera el modelo. La otra clase es una especialización de la base, y un objeto de esta representa un único registro de la tabla, por lo que mediante el uso de algunos métodos get y set se pueden manipular todos sus atributos los cuales corresponden a los campos de la tabla en la base de datos. La última clase define funciones que generalmente retornan colecciones o listas de objetos de la clase anterior.

2.4.1 Tema para los módulos de administración

En las aplicaciones web se pueden realizar operaciones de accesos a datos como insertar, actualizar, obtener y eliminar registros, conocidas por el acrónimo CRUD (iniciales de sus nombres en inglés). Symfony permite generar automáticamente el código de las operaciones CRUD, lo que facilita el desarrollo de las tareas administrativas.

Los módulos de administración utilizan un archivo de configuración especial denominado *generator.yml* que puede ser modificado para controlar la forma visual de los módulos y ampliar los componentes generados automáticamente.

Para conseguir un aspecto visual y funcionalidades comunes para los módulos encargados de la gestión, se crea un tema general de administración (zera_admin) que es activado desde el archivo *generator.yml* de cada uno.

```
generator:
  class: sfDoctrineGenerator
  param:
    ...
    theme: zera_admin
    …
```

En el tema se crean y redefinen funcionalidades que facilitan los procesos de administración, como la edición rápida y acciones de retroceso, brindando a los desarrolladores la posibilidad de crear elementos

estándares y reutilizables. Para incluir la edición rápida en un módulo se añade en el archivo *generator.yml* las siguientes líneas de código:

```
config:
  list:
    object_actions:
      _edit: ~
      _fastedit:
          label: Fast Edit
      _delete: ~
```

2.4.2 Librerías

Existen numerosas técnicas para ganar en optimización y rendimiento de las aplicaciones, que van desde la optimización de los servidores, la capa del modelo, las vistas, hasta incluso el código fuente. Una forma de optimizar las vistas es acelerar el proceso de carga de las páginas, teniendo un solo archivo con todo el código javascript. Sin embargo, durante el desarrollo y mantenimiento de las aplicaciones podría surgir el inconveniente de que los ficheros tendrían demasiadas líneas de código. Para solucionar esto, se crea un helper[6] el cual, en modo de producción, se encarga de unir automáticamente en un solo archivo todos los ficheros javascript de las páginas, y después cargarlo en caché. Esto último garantiza que una página que se haya cargado anteriormente ya tenga su archivo con todo el código javascript, evitando así repetir todo el proceso de unión de los ficheros.

La forma de ejecutar muchas de las funciones y componentes de symfony resulta tedioso para algunos desarrolladores, y no siempre cubren todas las funcionalidades requeridas. Es por ello que se crearon un conjunto de librerías que facilitan el desarrollo de la plataforma:

- **sfAOLink:** Permite redireccionarse de una aplicación symfony a otra o a una acción específica de un módulo. Por ejemplo, para ir a la aplicación *learning* basta con *sfAOLink::toApp('learning')*; para ejecutar la acción index del módulo principal que se encuentra en la aplicación *learning* se logra con *sfAOLink::toApp('learning', 'principal/index')*.

6 Función definida por Symfony, que puede tener parámetros y devolver código HTML.

- **sfAOTask:** Permite ejecutar comandos symfony en tiempo de ejecución. Por ejemplo, al llamar la función sfAOTask::clearCache() se limpia la caché.
- **sfUserTools:** Con esta librería se puede manipular y acceder a los datos del usuario que haya accedido a la aplicación. Además, permite almacenar o remover datos de la sesión del usuario y verificar si este ha iniciado sesión en la plataforma. Ejemplo *sfUserTools::getConnectedUser()* devuelve verdadero si el usuario está autenticado y falso en caso contrario.
- **sfMaterials:** Permite el trabajo con ficheros de configuración *yml y* se utiliza para almacenar datos de configuración de las materias que se crean en tiempo de ejecución. Ejemplo: *sfMaterials::get_materials_by_referent($referencias)* se utiliza para obtener los datos de configuración de una materia dada la referencia de la materia.
- **sfNomenclature:** Esta librería representa una capa de abstracción para el uso de los nomencladores de los diferentes tipos de recursos existentes, asignando la responsabilidad del manejo de estos a la librería y no al desarrollador ganando en usabilidad y mantenimiento. Ejemplo: *sfNomenclature::getImage()* se utiliza para obtener el nomenclador correspondiente al recurso imagen y *sfNomenclature::getImage(true)* para obtener el identificador de la tabla correspondiente a ese nomenclador.

2.4.3 Plugins

Un plugin es una extensión encapsulada para un proyecto symfony. Además de facilitar la reutilización de código, permiten aprovechar los desarrollos realizados por otros programadores y añadir al núcleo de symfony nuevas extensiones. La forma en la que Symfony carga los plugins permite que los proyectos puedan utilizarlos como si fueran parte del propio framework [23].

Para estandarizar el desarrollo de plugins en el proyecto se define la siguiente estructura básica de archivos y directorios:

```
sfNombrePlugin/
    config/
        app.yml
        config.php                  //  Configuración específica del plugin
        doctrine/
            nombre.schema.yml       //  Esquema de datos (en caso de ser necesario)
    lib/                            //  Librerías, formularios, clases del modelo
    modules/
    web/                            // Archivos javascript, css, imágenes
    README
    LICENSE
```

Los archivos de configuración en los plugins de Symfony repercuten en todas las aplicaciones. Un posible error en algunos de estos ficheros puede afectar todo el sistema. Para mitigar este riesgo se definen las siguientes restricciones:

- No usar los archivos routing.yml, filters.yml, factories.yml .
- Si se incluye el fichero settings.yml, las variables globales que se declaren en este no pueden coincidir con ninguna definida por el framework. Como alternativa se propone el uso del archivo app.yml.

En el Anexo 1 se muestra una descripción de algunos de los plugins esenciales en el desarrollo del proyecto.

2.5 Vista de Casos de Uso

2.5.1 Casos de uso arquitectónicamente significativos

Los casos de uso arquitectónicamente significativos son aquellos que representan las partes más críticas del sistema y se utilizan para la validación de la arquitectura a lo largo del desarrollo del software. A continuación se muestran los diagramas de casos de uso agrupados por paquetes lo que facilita la comprensión de los mismos.

2.5.2 Diagrama de casos de uso del sistema arquitectónicamente significativos

2.5.2.1 Paquete Escuela

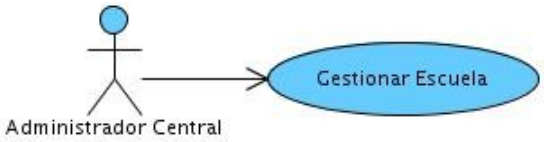


Fig. 3: Diagrama de CUS arquitectónicamente significativos del paquete Escuela

2.5.2.2 Paquete Sistema Educativo

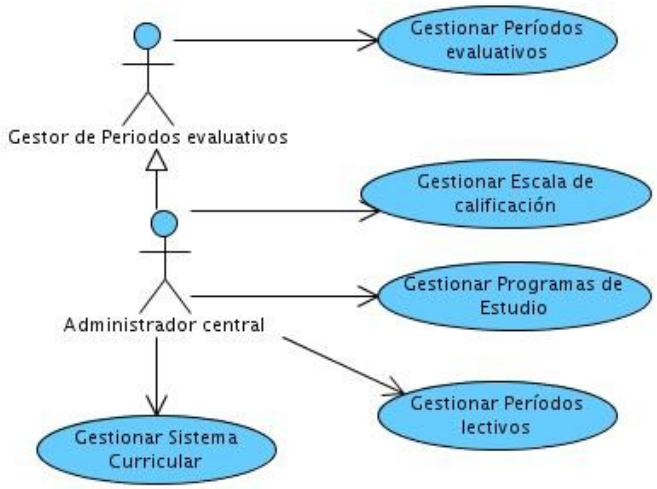


Fig. 4: Diagrama de CUS arquitectónicamente significativos del paquete Sistema Educativo

2.5.2.3 Paquete Usuarios

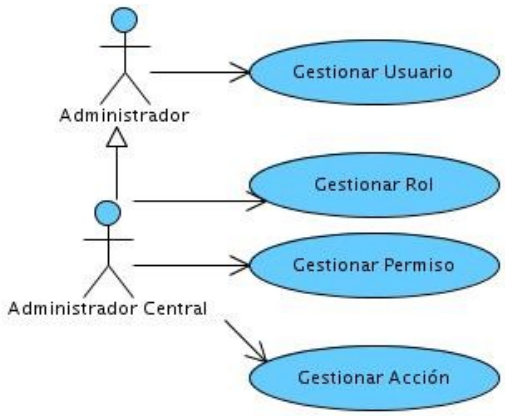


Fig. 5: Diagrama de CUS arquitectónicamente significativos del paquete Usuarios

2.5.2.4 Paquete Configuración de la Plataforma

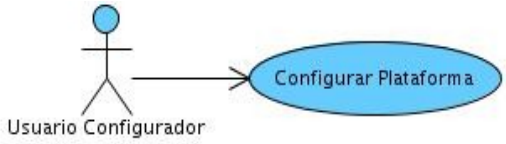


Fig. 6: Diagrama de CUS arquitectónicamente significativos del paquete Configuración de la Plataforma

2.5.2.5 Paquete Comunicaciones

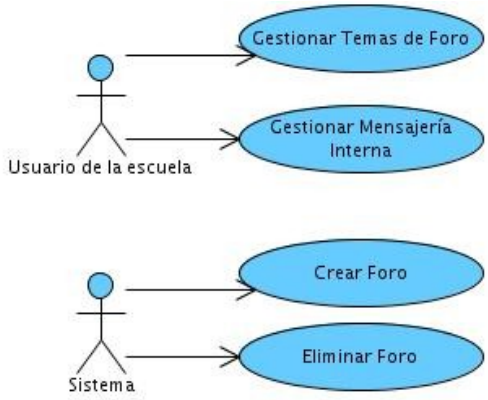


Fig. 7: Diagrama de CUS arquitectónicamente significativos del paquete Comunicaciones

2.5.2.6 Paquete Réplica

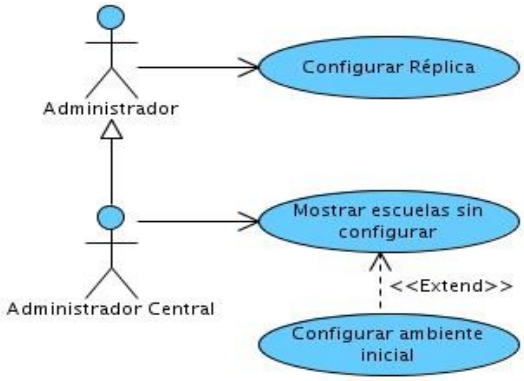


Fig. 8: Diagrama de CUS arquitectónicamente significativos del paquete Réplica

2.5.2.7 Paquete Gestor de Páginas

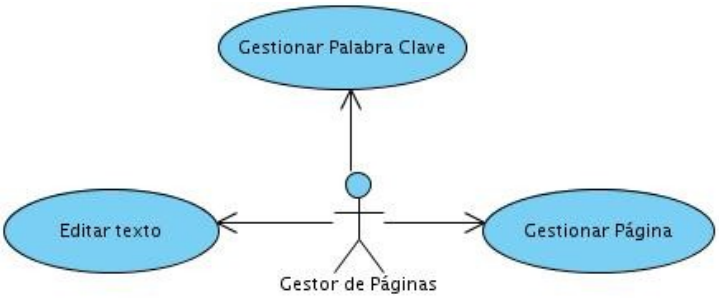


Fig. 9: Diagrama de CUS arquitectónicamente significativos del paquete Gestor de Páginas

2.5.2.8 Paquete Recursos

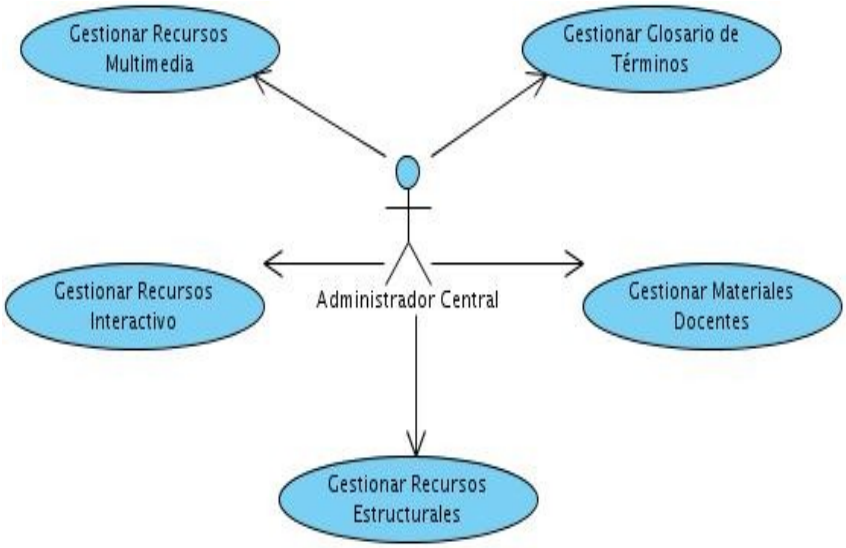


Fig. 10: Diagrama de CUS arquitectónicamente significativos del paquete Recursos

2.5.2.9 Paquete Cuestionario

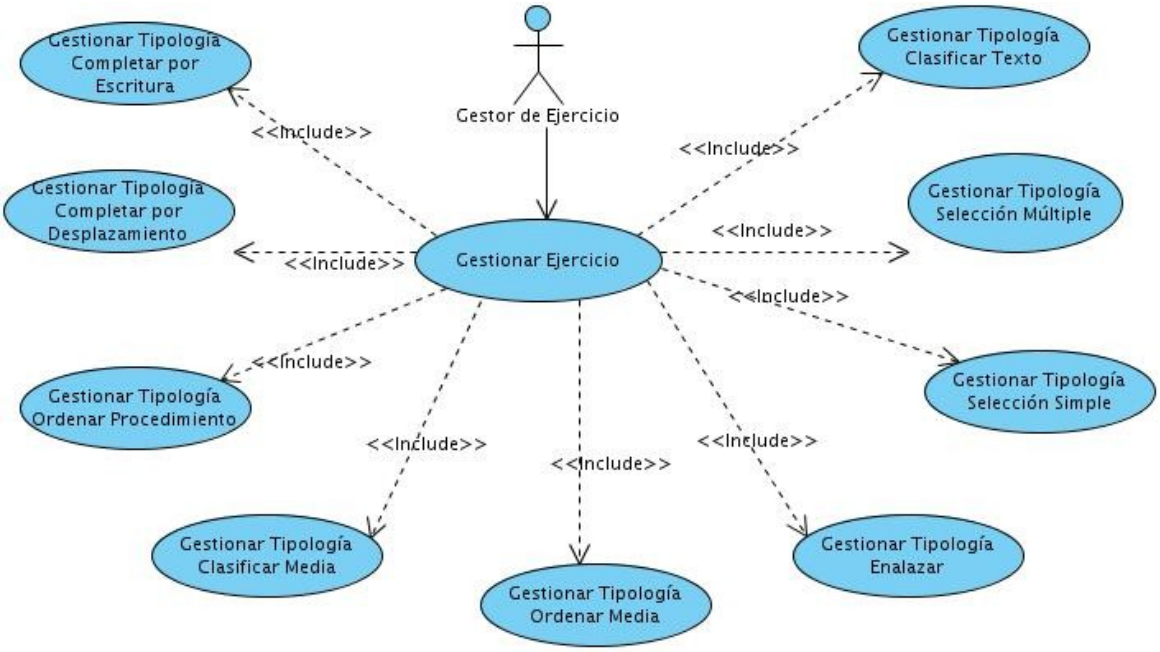


Fig. 11: Diagrama de CUS arquitectónicamente significativos del paquete Cuestionario

2.5.2.10 Paquete Materias

Fig. 12: Diagrama de CUS arquitectónicamente significativos del paquete Materias

2.6 Vista Lógica

La Vista Lógica muestra cómo la funcionalidad es diseñada dentro del sistema y define la estructura y el comportamiento del mismo. Para facilitar el desarrollo del proyecto, este fue dividido en paquetes de

diseño según las aplicaciones symfony que lo componen. En correspondencia con lo anterior, el diseño es agrupado en cuatro paquetes generales, más uno de apoyo (plugins) formado por los subsistemas de diseño comunes en los paquetes generales y otras funcionalidades extendidas.

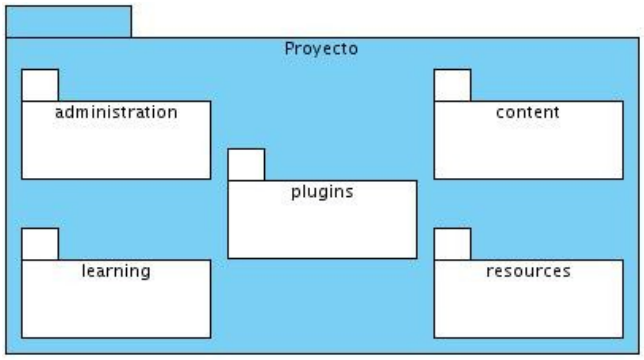


Fig. 13: Vista Lógica principal

Los paquetes generales están divididos en subsistemas de diseño, estructurados en su mayoría como se muestra a continuación:

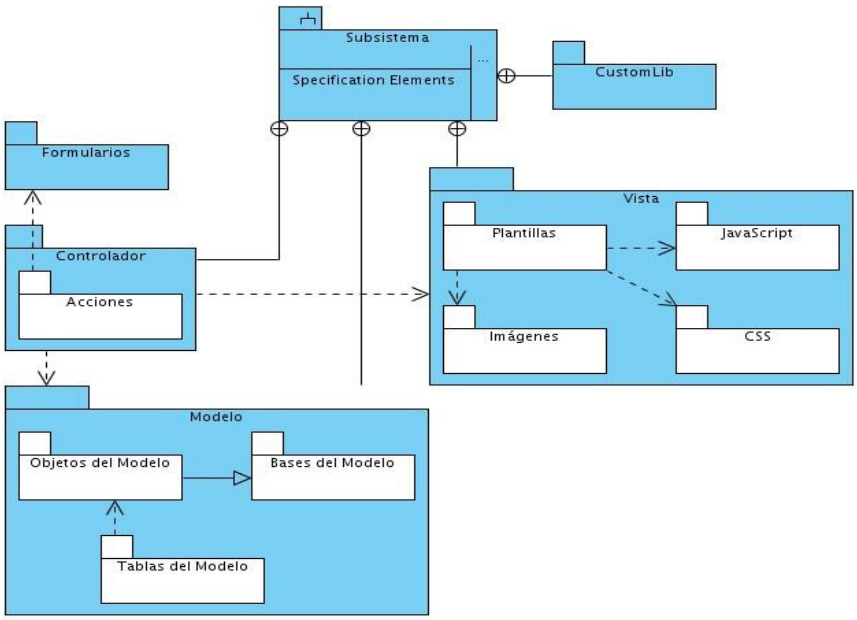


Fig. 14: Estructura de los subsistemas de diseño

- **Acciones:** Contiene las clases de las acciones, estas son la esencia de la aplicación. Además de contener toda la lógica del sistema, utilizan el modelo y definen variables para la vista.
- **Plantillas:** Páginas que muestran información al usuario.
- **Javascript:** Conjunto de scripts necesarios para las plantillas.
- **CSS:** Archivos con hojas de estilos necesarios en las plantillas.
- **Imágenes:** Imágenes necesarias en las plantillas.
- **Bases del modelo:** Clases que representan una tupla de una tabla del modelo, estas poseen atributos y relaciones del registro correspondiente en la tabla y no pueden se pueden personalizar porque se sobreescriben cada vez que el modelo es generado.
- **Objetos del modelo:** Contiene las clases especializadas de la base, estas permiten manipular los atributos de la tupla correspondiente en la tabla del modelo.

- **Tablas del modelo:** Clases que definen funciones que generalmente retornan colecciones o listas de objetos de la clase anterior.
- **Formularios:** Clases de formularios para la entrada de datos a la aplicación.
- **Custom Lib:** Contiene las clases o librerías que brindan funcionalidades auxiliares a la aplicación.

2.6.1 Paquetes generales del proyecto

A continuación se describe la estructura de cada paquete general del proyecto, con los subsistemas del diseño que lo componen. Como la estructura de estos subsistemas es similar se realiza el diagrama de clases del diseño asociado a uno de ellos en el paquete *administration.*

2.6.2 Paquete "administration"

Está dividido en tres subsistemas de diseño, el primero para gestionar las escuelas, y los restantes encargados de la gestión y configuración de los sistemas educativos con sus respectivos sistemas curriculares:

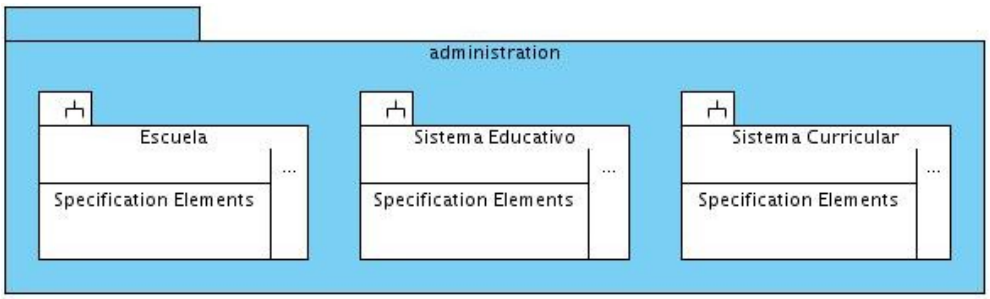


Fig. 15: Subsistemas de diseño del paquete administration

- **Escuela**: Se encarga de la gestión de una institución educativa y todos sus datos.
- **Sistema educativo**: Tiene el objetivo de crear y/o personalizar los sistemas educativos y todos los elementos que lo componen.

- **Sistema curricular**: Responsable de la gestión de los sistemas curriculares.

2.6.2.1 Diagrama de clases del diseño

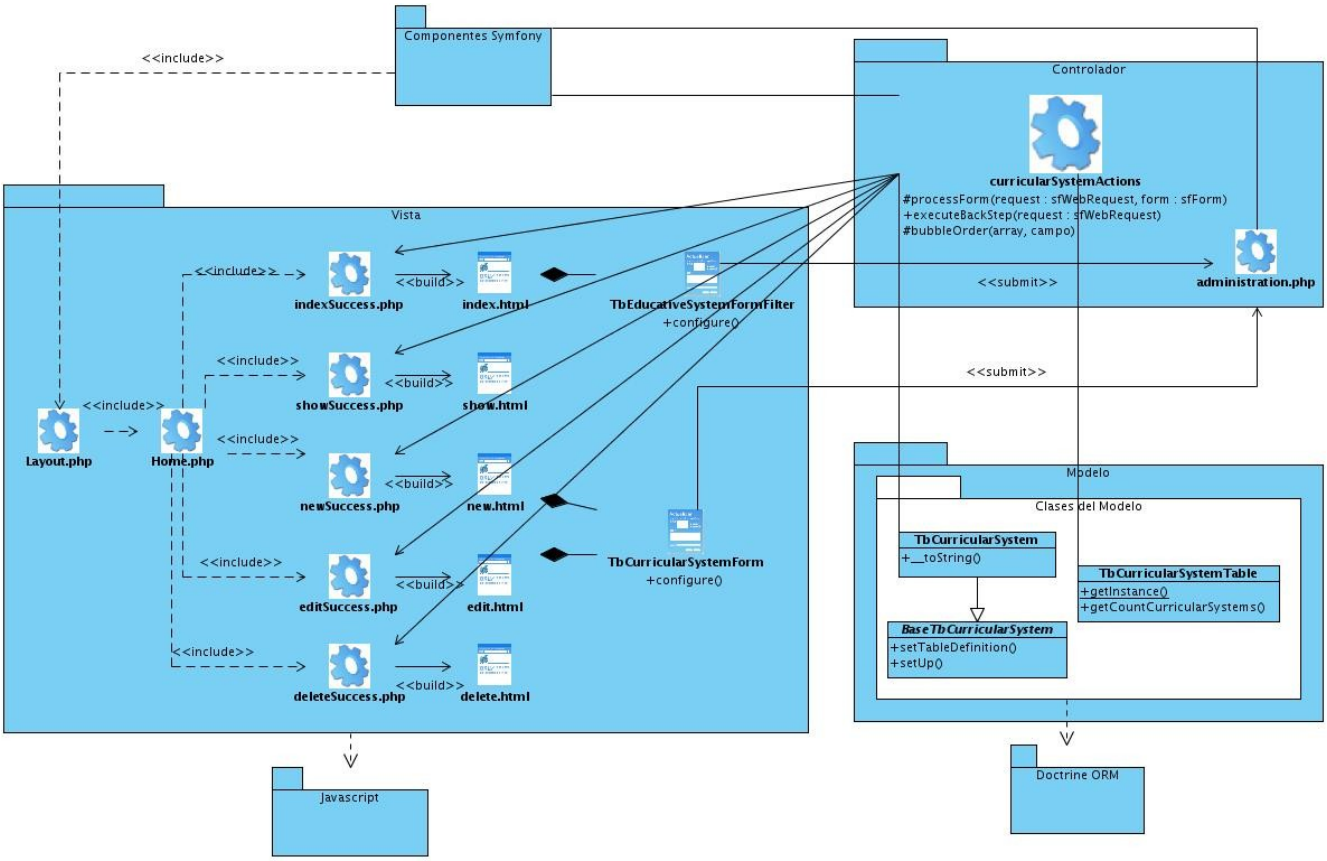


Fig. 16: Diagrama de clases del diseño del subistema Sistema Curricular

2.6.3 Paquete "learning"

Se divide en dos subsistemas de diseño, ambos encargados de las comunicaciones, el primero para la gestión del foro y el otro para la mensajería:

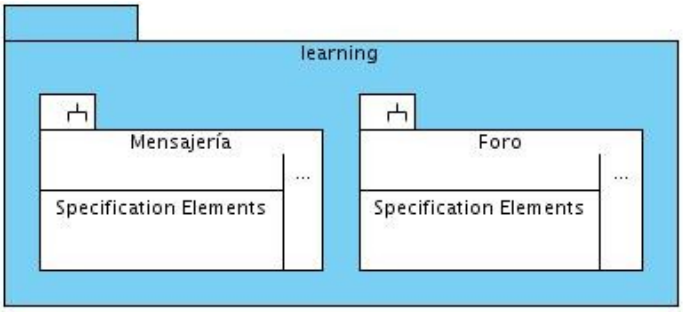


Fig. 17: Subsistemas de diseño del paquete learning

- **Mensajería**: Su objetivo es controlar la comunicación en la plataforma, o sea, todo lo relacionado con la comunicación entre los usuarios y profesores docentes.

- **Foro**: Responsable de crear un espacio de debate y discusión sobre las diferentes temáticas creadas por los docentes, todo esto mediado por un moderador. Este espacio permitirá intercambiar ideas, llevar a cabo una participación activa de los estudiantes y docentes, estimulando en la mayoría de los casos, un aprendizaje activo.

2.6.4 Paquete "content"

Tiene como propósito fundamental agrupar una serie de funcionalidades correspondientes a cada Materia y su respectivo Macro Índice, de manera que se pueda garantizar su correcta administración, permitiendo de esta forma la creación de contenidos que luego serán mostrados y utilizados por los docentes y estudiantes. Está dividido en dos subsistemas de diseños:

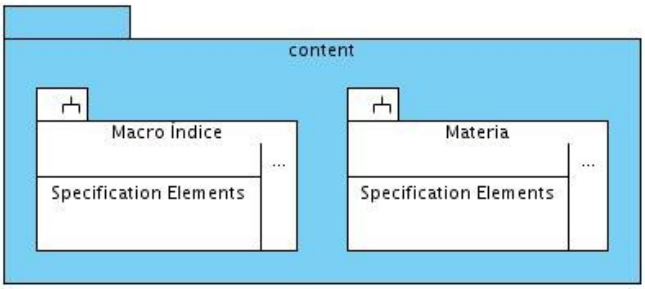


Fig. 18: Subsistemas de diseño del paquete content

- **Macro Índice:** Gestiona la estructura de la materia a través de la asociación de capítulos, temas y subtemas. La estructura es similar a la de un libro donde se incluyen además las páginas y los recursos que se asocian al mismo.
- **Materia**: Tiene como objetivo la gestión de las materias que constituyen la base para la creación de los contenidos correspondientes a las mismas.

2.6.5 Paquete "resources"

Tiene como objetivo principal gestionar todos los tipos de recursos interactivos, así como asociarlos a una materia existente para su posterior visualización. Se compone por tres subsistemas de diseño:

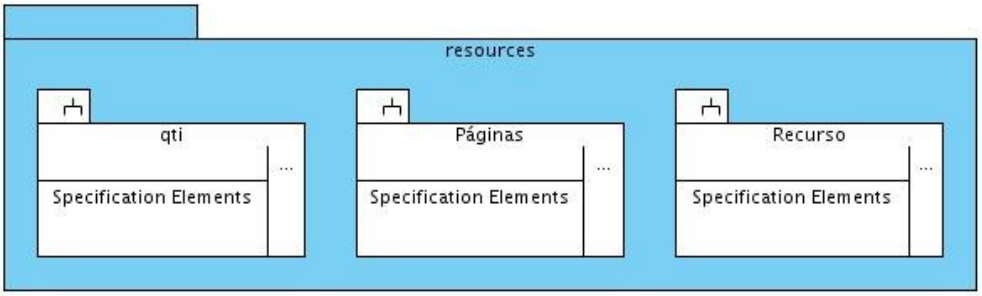


Fig. 19: Subsistemas de diseño del paquete resources

- **Qti**: Su objetivo es gestionar los cuestionarios interactivos de manera que puedan ser utilizados por los docentes y estudiantes; actualmente se definen nueve tipologías de ejercicios diferentes.
- **Páginas**: Se gestionan las páginas, las cuales representan los pilares de los contenidos, pues dentro de ellas se redacta el texto y se asocian los diferentes recursos interactivos.

- **Recurso**: Su objetivo es gestionar los recursos de manera detallada, estos se organizan en cuatro tipos: Multimedia, Estructurales, Interactivos y Glosario de Término.

2.6.6 Paquete "plugins"

Atendiendo a las características de Symfony definidas en el acápite "2.4.3" se crea este paquete, el cual se divide en trece subsistemas de diseño, tres encargados de agrupar funcionalidades comunes de los paquetes *administration* y *learning,* referente a la configuración de los sistemas educativos; cuatro para gestionar los usuarios, roles, permisos y las acciones que se podrán realizar en la aplicación. Uno responsable de configurar el sistema de réplica, y otro encargado de las restantes configuraciones de la plataforma.

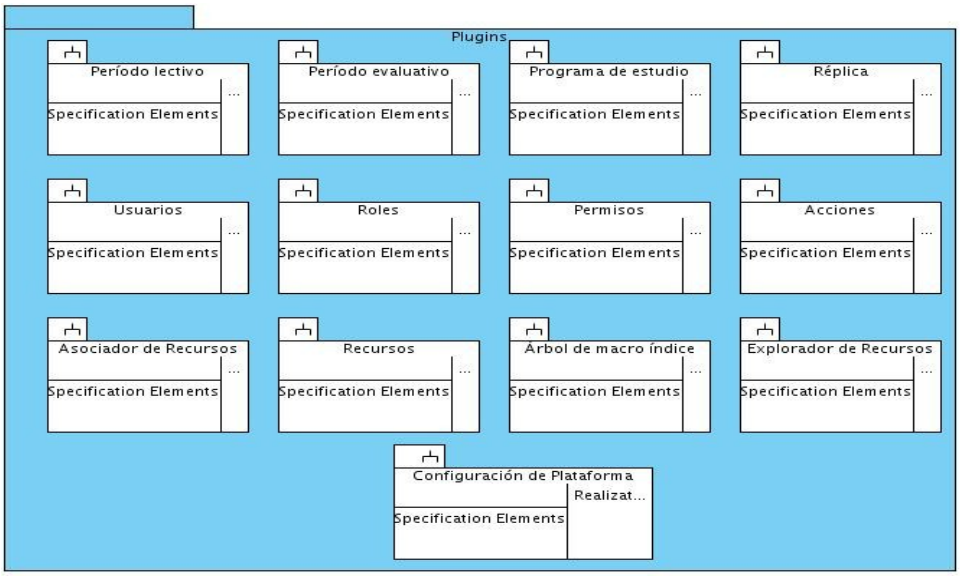


Fig. 20: Subsistemas de diseño del paquete plugins

- **Período lectivo:** Permite la gestión y consulta de los períodos lectivos de las escuelas.
- **Período evaluativo**: Gestión de los períodos evaluativos.
- **Programa de estudio**: Responsable de la gestión y consulta de los programas de estudio de las escuelas.

- **Usuarios**: Gestión de los usuarios de la plataforma.
- **Roles**: Gestión de los roles asignados a los usuarios que pertenecen a la institución.
- **Permisos**: Gestión de los permisos asignados a los roles.
- **Acciones**: Encargado de administrar las acciones que se pueden realizar en la plataforma y la asignación a los permisos.
- **Configuración de plataforma**: Encargado de los principales procesos de configuración de la plataforma como, configurar servidor de medias, archivos válidos y tamaño máximo de los ficheros.
- **Réplica**: Gestión de la sincronización y réplica de los datos entre servidores.
- **Recursos**: Encargado de la visualización de los recursos en la plataforma.
- **Árbol de macro índice**: Encargado de la visualización del macro índice en forma de árbol con las opciones contextuales correspondientes a cada elemento del macro índice.
- **Explorador de recursos:** Componente que permite realizar búsquedas sobre los distintos tipos de recursos existentes, así como la selección de distintas formas de estos para su posterior uso.
- **Asociador de recursos:** Componente que permite asociar recursos existentes a otras estructuras del macro índice. Depende del explorador de recursos para su ejecución.

2.7 Vista de Despliegue

La vista de despliegue permite capturar los elementos de configuración que se necesitan para el procesamiento y las conexiones que se necesitan entre esos elementos en tiempo de ejecución. Visualizar la distribución de los componentes de software en los nodos físicos.

En la siguiente imagen muestra el diagrama de despliegue correspondiente a la plataforma ZERA:

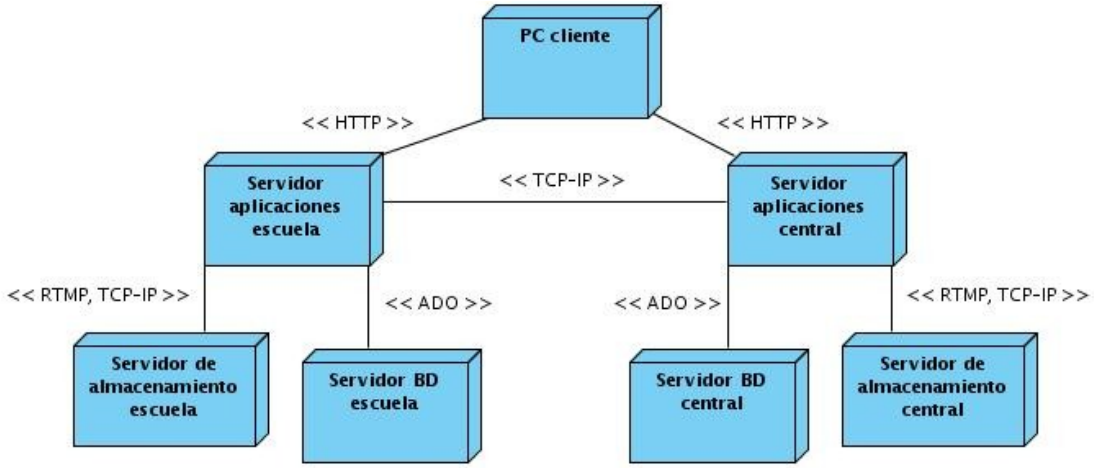


Fig. 21: Diagrama de despliegue de ZERA

Nodos:

- **PC cliente**: Ordenador cliente que se conecta a través de un navegador web al servidor donde reside la plataforma.
- **Servidor de aplicaciones central**: Servidor central donde se encuentra instalada la plataforma con todas las funcionalidades.
- **Servidor BD central**: Servidor donde se encuentra instalado la base de datos correspondiente al servidor de aplicaciones de central.
- **Servidor de almacenamiento central**: Servidor para el almacenamiento de recursos como medias y datos, el cual tendrá instalado un servidor streaming o de medias accesibles por el servidor de aplicaciones central.
- **Servidor de aplicaciones escuela**: Servidor donde se encuentra instalada la versión de la aplicación disponible para una escuela, es decir, solo tendrá visibilidad para los clientes de esa escuela.

- **Servidor BD escuela**: Servidor con similar uso del servidor de BD central pero solo tendrá alcance para el servidor de aplicaciones de la escuela.
- **Servidor de almacenamiento escuela**: Servidor con similar uso del servidor de almacenamiento central pero solo tendrá alcance para el servidor de aplicaciones de la escuela.

Conectores:

- **HTTP**: la comunicación entre el ordenador cliente y el servidor correspondiente se realiza a través de un navegador web, que usa el protocolo HTTP.
- **TCP/IP**: este protocolo se utiliza para la comunicación entre los servidores de aplicaciones para la replicación, así como entre un servidor de aplicaciones y el de almacenamiento para el acceso a las medias y datos.
- **RTMP**: protocolo que usa la plataforma para la visualización de las medias en tiempo de ejecución de la misma.

2.8 Conclusiones

En este capítulo se realizó una descripción de la arquitectura de software de la plataforma de gestión de aprendizaje ZERA, haciendo referencia a los requerimientos no funcionales del sistema y la óptica arquitectónica de las 4+1 vistas que propone RUP. Quedaron definidas las vistas de Casos de Uso, la Lógica y la de Despliegue. Se concluye que estas vistas arquitectónicas son de gran importancia para el equipo de desarrollo ya que le permite un mejor entendimiento de la organización del sistema.

Capítulo 3: Evaluación de la Arquitectura

3.1 Introducción

El establecimiento de una correcta arquitectura de software tributa de manera positiva en la calidad del sistema propuesto, por lo que es muy importante estar en capacidad de tomar decisiones acertadas sobre ella, así como realizar las pruebas necesarias para asegurar su correcta definición. En el presente capítulo se aborda lo concerniente a la evaluación de la arquitectura, su necesidad e importancia; así como los diferentes métodos, técnicas y atributos de calidad que se emplean para realizarla.

3.2 Evaluación de la Arquitectura

El primer paso para la evaluación de una arquitectura es conocer qué es lo que se quiere evaluar. De esta forma, es posible establecer la base para la evaluación, puesto que la intención es saber qué se puede evaluar y qué no [24]. Resulta interesante el estudio de la evaluación de una arquitectura: si las decisiones que se toman sobre la misma determinan los atributos de calidad del sistema, es entonces posible evaluar las decisiones de tipo arquitectónico con respecto al impacto sobre estos atributos.

La evaluación de una arquitectura de software es una tarea no trivial, puesto que se pretende medir propiedades del sistema en base a especificaciones abstractas, como por ejemplo los diseños arquitectónicos [25]. Por ello, la intención es más bien la evaluación del potencial de la arquitectura diseñada para alcanzar los atributos de calidad requeridos, así como el cumplimiento de los requerimientos no funcionales que solicita el cliente.

3.2.1 ¿Por qué evaluar la arquitectura?

Cuanto más temprano se encuentre un problema en el desarrollo del software, mejor. El costo de arreglar un error durante las fases de requerimientos o diseño, es mucho menor al costo de arreglar ese mismo error en la fase de verificación. Dado que la arquitectura es un producto temprano de la fase de diseño,

esta tiene un profundo efecto en el sistema y en el proyecto. Una mala arquitectura puede llevar a un proyecto al fracaso. Todos los requerimientos de calidad pueden quedar insatisfechos.

3.2.2 Atributos de calidad

La calidad de software se define como el grado en el cual el software posee una combinación deseada de atributos. Estos atributos son requerimientos adicionales del sistema, que hacen referencia a características que este debe satisfacer, diferentes a los requerimientos funcionales [26]. Estas características o atributos se conocen con el nombre de atributos de calidad, los cuales se definen como las propiedades de un servicio que presta el sistema a sus usuarios [27].

Algunos autores establecieron una clasificación de los atributos de calidad en dos categorías [4]:

- **Observables vía ejecución**: aquellos atributos que se determinan del comportamiento del sistema en tiempo de ejecución. La descripción de algunos de estos atributos se presenta en la Tabla 5.
- **No observables vía ejecución**: aquellos atributos que se establecen durante el desarrollo del sistema. La descripción de algunos de estos atributos se presenta en la Tabla 6.

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

Tabla 5: Descripción de atributos de calidad observables vía ejecución.

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

Tabla 6: Descripción de atributos de calidad no observables vía ejecución.

3.3 Técnicas de Evaluación

Existen diferentes técnicas que permiten realizar la evaluación de la arquitectura, estas son ubicadas en dos grupos, las técnicas cualitativas y las cuantitativas. Las técnicas contenidas en cada grupo se muestran en la figura que aparece a continuación:

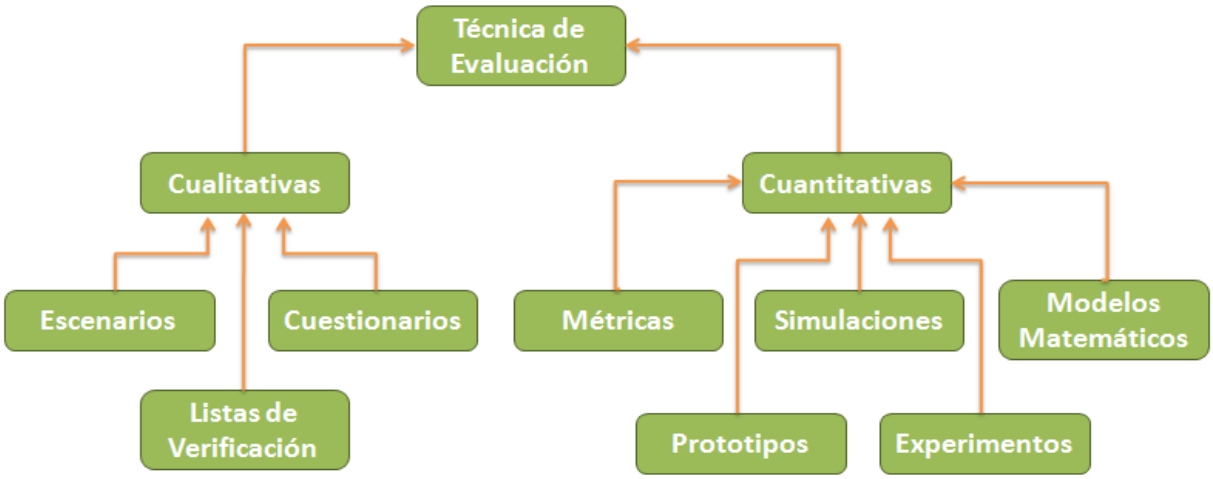


Fig. 22: Clasificación de las técnicas de evaluación

Para su estudio se realizó una selección de cada grupo, con el objetivo de analizar características generales de ambos enfoques. Para ello se analizaron las técnicas basadas en escenarios por la parte cualitativa, y las basadas en modelos matemáticos, simulaciones y experiencias por las que cuantifican sus resultados.

3.3.1 Técnicas 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 [26]. Por ejemplo, un usuario hará la descripción en términos de la ejecución de una tarea; un encargado de mantenimiento hará referencia a cambios que deban realizarse sobre el sistema; un desarrollador se enfocará en el uso de la arquitectura para efectos de su construcción o predicción de su desempeño.

Un escenario consta de tres partes: el estímulo, el contexto y la respuesta. El estímulo es la parte del escenario que explica o describe lo que el involucrado en el desarrollo hace para iniciar la interacción con el sistema. Puede incluir ejecución de tareas, cambios en el sistema, ejecución de pruebas, configuración, etc. El contexto describe qué sucede en el sistema al momento del estímulo. La respuesta describe, a través de la arquitectura, cómo debería responder el sistema ante el estímulo [26]. Este último elemento es el que permite establecer cuál es el atributo de calidad asociado.

Entre las principales ventajas de su uso están:

- Son simples de crear y entender .
- Son poco costosos y no requieren mucho entrenamiento.
- Son efectivos .

Esta técnica de evaluación es una de las más útiles en la evaluación del comportamiento de una arquitectura, debido a su simplicidad y nivel de abstracción, logra ser comprendida por todos los involucrados en un proceso de evaluación, desde los desarrolladores hasta los clientes y usuarios finales; permitiendo ampliar el espectro de opiniones valorativas sobre la arquitectura de software propuesta.

Actualmente las técnicas basadas en escenarios cuentan con dos instrumentos de evaluación relevantes: el Árbol de Utilidades y los Perfiles, los cuales se detallan a continuación.

3.3.1.1 Árbol de Utilidades

Un Árbol de Utilidades 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 [26]. La intención del uso de este instrumento es la identificación de los atributos de calidad más importantes para un proyecto particular. No existe un conjunto preestablecido de atributos, sino que son definidos por los involucrados en el desarrollo del sistema al momento de la construcción del árbol.

El Árbol de Utilidades contiene como nodo raíz la utilidad general del sistema. Los atributos de calidad asociados al mismo conforman el segundo nivel del árbol, los cuales se refinan hasta la obtención de un escenario lo suficientemente concreto para ser analizado y otorgarle prioridad a cada atributo considerado. Cada atributo de calidad perteneciente al árbol contiene una serie de escenarios relacionados, y una escala de importancia y dificultad asociada, que será útil para efectos de la evaluación de la arquitectura.

3.3.1.2 Perfiles

Un perfil es un conjunto de escenarios, generalmente con alguna importancia relativa asociada a cada uno de ellos. El uso de perfiles permite hacer especificaciones más precisas del requerimiento para un atributo de calidad [25]. Los perfiles tienen asociados dos formas de especificación:

- **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 [25].
- **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 [25].

3.3.2 Técnicas de evaluación basadas en simulación

La evaluación basada en simulación utiliza una implementación de alto nivel de la arquitectura de software. El enfoque básico consiste en la implementación de componentes de la arquitectura y la implementación del contexto del sistema donde se supone va a ejecutarse [25]. La finalidad es evaluar el comportamiento de la arquitectura bajo diversas circunstancias. Una vez disponibles estas implementaciones, pueden usarse los perfiles para evaluar los atributos de calidad.

En términos de los instrumentos asociados a las técnicas de evaluación basadas en simulación, se encuentran los lenguajes de descripción arquitectónica y los modelos de colas.

Esta técnica presenta un grupo de dificultades, como son, la exactitud de los resultados de la evaluación depende, a su vez, de la exactitud del perfil utilizado para evaluar el atributo de calidad y de la precisión con la que el contexto del sistema simula las condiciones del mundo real. Estos elementos tributan a que la evaluación no tenga un nivel alto de precisión, pues sus resultados depende de factores subjetivos, que pueden ser logrados o no. Otro factor importante es que la aplicación de esta técnica durante un proceso de desarrollo de software provoca un aumento considerable de esfuerzo y tiempo, por el alto nivel de especificación que se necesita en cada uno de sus pasos y los conocimientos avanzados con que debe contar el equipo de evaluación para su aplicación.

3.3.3 Técnicas de evaluación basadas en modelos matemáticos

La evaluación basada en modelos matemáticos permite una evaluación estática de los modelos de diseño arquitectónico, y se presentan como alternativa a la simulación, dado que evalúan el mismo tipo de atributos. Ambos enfoques pueden ser combinados, utilizando los resultados de uno como entrada para el otro [25].

Entre los instrumentos que se cuentan para las técnicas de evaluación de arquitecturas de software basada en modelos matemáticos, se encuentran las Cadenas de Markov y los Diagramas de bloques.

Esta técnica de evaluación, aparte de que carece para su implementación de modelos matemáticos apropiados para los atributos de calidad relevantes para la arquitectura de software, su principal desventaja radica en los esfuerzos sustanciales que requiere la técnica para su implementación, aparejado a un alto nivel de conocimientos necesarios para la adaptación de los modelos.

3.3.4 Técnicas de evaluación basadas en experiencias

En muchas ocasiones los arquitectos e ingenieros de software otorgan valiosas ideas que resultan de utilidad para la evasión de decisiones erradas de diseño. Todas estas experiencias se basan en evidencia

anecdótica; es decir, basada en factores subjetivos como la intuición y la experiencia. Sin embargo, la mayoría de ellas puede ser justificada por una línea lógica de razonamiento, y pueden ser la base de otros enfoques de evaluación [25].

Existen dos tipos de evaluación basada en experiencia: la evaluación informal, que es realizada por los arquitectos de software durante el proceso de diseño, y la realizada por equipos externos de evaluación de arquitecturas.

Estos tipos de técnicas de evaluación, no están basadas en conceptos teóricos, sino que se apoyan en factores intrínsecos de las personas, que pueden ser efectivos o no. Su principal dificultad radica en que sus resultados son variables y no confiables. Una aplicación de esta técnica puede dar excelentes resultados en un contexto, mientras que puede ser fatal su aplicación en otros. Es importante para su uso, la existencia de un personal que haya acumulados un conjunto de conocimientos y experiencias, que brinden un razonamiento lógico a las decisiones arquitectónicas tomadas. De manera general, puede servir como criterio durante el proceso de evaluación.

3.3.5 Conclusiones parciales sobre las técnicas de evaluación

De manera general, las técnicas de evaluación de arquitectura permiten una valoración de la arquitectura de software en las primeras fases del diseño arquitectónico, y pueden servir como medidor para valorar lo acertado de la arquitectura de software. Luego de un análisis, se observa que las técnicas de evaluación cuantitativas requieren un esfuerzo mayor en su implementación que las técnicas cualitativas. Estas últimas están caracterizadas por su simplicidad y el logro de su entendimiento por parte de todos los involucrados que participan en una evaluación arquitectónica.

Las técnicas, generalmente, no son utilizadas para realizar una evaluación arquitectónica de manera directa. Su principal utilidad es apoyar la evaluación realizada por un método [28]. Por su sencillez y fácil implementación, la más difundida es la técnica de evaluación basada en escenario, en la cual se apoyan los métodos que se analizan a continuación.

3.4 Métodos de evaluación

Los métodos de arquitectura surgen de la necesidad de evaluar el potencial de una arquitectura de software para soportar los requisitos de software. Muchos han sido los métodos propuestos para el análisis de las arquitecturas, en su mayoría adecuaciones a contextos específicos basados en métodos ya diseñados [29]. A continuación se detallan un conjunto de métodos de evaluación seleccionados por su importancia y representatividad con respecto a los existentes en la actualidad. Para caracterizar a cada uno se utilizan los siguientes aspectos: objetivo general, pasos a realizar para la evaluación y comentarios generales del método.

3.4.1 Método de Análisis de Arquitecturas de Software (SAAM)

El Método de Análisis de Arquitecturas de Software (Software Architecture Analysis Method, SAAM) es el primero que fue ampliamente promulgado y documentado. Este tiene como objetivo evaluar un grupo de atributos de calidad que debe lograr la arquitectura de software, a través del uso de escenarios. En la práctica ha demostrado ser útil para la rápida evaluación de atributos de calidad como: modificabilidad, portabilidad y extensibilidad. En resumen se puede afirmar, que al evaluar una arquitectura, SAAM indica los puntos de fortalezas y debilidades, junto con los puntos de la arquitectura que no cumple con los requisitos de modificabilidad [26].

Este método consta de seis pasos principales :

1. Desarrollar escenarios.
2. Describir la arquitectura.
3. Clasificar y Priorizar Escenarios.
4. Evaluar los escenarios indirectos.
5. Evaluar la interacción de escenarios
6. Crear una evaluación global

Con la aplicación de este método, si el objetivo de la evaluación es una sola arquitectura, se obtienen los lugares en los que la misma puede fallar, en términos de los requerimientos de modificabilidad.

Dentro de las principales deficiencias que posee se observa que no ofrece un claro indicador de la calidad arquitectónica en los atributos que analiza. Además el proceso de generación de escenarios se basa en una visión de los involucrados externos. Es muy difícil ver, por ejemplo, a un actor en la definición de un escenario indirecto.urando

3.4.2 Método de Análisis de Acuerdos de Arquitectura (ATAM)

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 [30], explicado anteriormente. Este método no sólo analiza cómo la arquitectura satisface los atributos de calidad como: modificabilidad, portabilidad, extensibilidad e integrabilidad, sino que además proporciona información sobre el cumplimiento de cada atributo de calidad teniendo en cuenta las interdependencias que se establecen entre ellos.

El método ATAM consta para su implementación de cuatro fases: presentación, investigación y análisis, pruebas y presentación de informes [30]. Cada fase es una colección de pasos. La fase de presentación implica el intercambio de información a través de presentaciones. La de investigación y análisis se refiere a la evaluación de los principales atributos de calidad teniendo en cuenta los requisitos que debe cumplir el sistema y los enfoques arquitectónicos. En la fase de prueba se comparan los resultados de la fase anterior con las necesidades de las partes interesadas. Por último, en la fase de presentación de informe, se resumen los resultados de la evaluación.

A continuación se presentan los pasos para la evaluación a través de ATAM [30]:

- **Fase 1: Presentación**
    1. Presentación del ATAM .
    2. Presentación de las metas del negocio .
    3. Presentación de la arquitectura .
- **Fase 2: Investigación y análisis**

4. Identificación de los enfoques arquitectónicos .
5. Generación del Árbol de Utilidades.
6. Análisis de los enfoques arquitectónicos .

- **Fase 3: Pruebas**
  7. Lluvia de ideas y establecimiento de prioridad de escenarios .
  8. Análisis de los enfoques arquitectónicos .

- **Fase 4: Reporte**
  9. Presentación de los resultados .

El método ATAM es el más abarcador de todos los métodos de evaluación de arquitectura. Brinda tratamiento a todos los atributos de calidad que impactan de manera directa sobre el diseño arquitectónico y permite analizar todos los sistemas de software. Estructura el proceso de evaluación de forma muy organizada, estableciendo una secuencia de actividades lógicas que abarcan involucrados, requisitos de entrada y resultado de cada una. Al igual que el resto de los métodos no utiliza un modelo de calidad o estándar en la selección, refinamiento y medición de los atributos de calidad. Sin embargo, mediante el instrumento del árbol de utilidad propone un nivel de refinamiento de los atributos que abarca, hasta la definición de métricas con criterios de máximos y mínimos, que permite un nivel de medición aceptable.

El ATAM tiene como fortaleza además, el manejo de la identificación de los riesgos de la arquitectura diseñada, mediante la identificación de puntos de sensibilidad, de desventaja y de riesgo.

3.4.3 Método de Análisis del Nivel de Modificación de la Arquitectura (ALMA)

El Método de Análisis del Nivel de Modificación de la Arquitectura (Architecture Level Modifiability Analysis, ALMA) tiene como objetivo la realización de un análisis correcto basado en la hipótesis de la evaluación de la arquitectura de software a través de la modificabilidad de un conjunto de indicadores: la predicción de los costes de mantenimiento, la evaluación de los riesgos, entre otros. En caso de evaluar y comparar diferentes sistemas, el análisis que realiza el método ALMA apoya el logro de la calidad

arquitectónica y la selección de la alternativa correcta. Es por esto que este método utiliza como técnica de evaluación, los escenarios de cambio, proporcionados por las partes interesadas [31].

El análisis de la modificabilidad que realiza, inicia con la definición de un conjunto de escenarios que pudieran ocurrir durante la evolución del sistema. Estos escenarios se utilizan para verificar la forma en que la actual arquitectura soporta o puede adaptarse a los cambios futuros.

Este método de evaluación consta de cinco pasos. Estos no siempre se realizan de forma secuencial y las iteraciones sobre ellos también son posibles:

1. Definir el objetivo de la evaluación .
2. Describir la arquitectura de software .
3. Selección de los escenarios de cambios .
4. Evaluar el escenario de cambio .
5. Interpretar los resultados .

Entre las consideraciones más importantes que aporta este método está la utilización de las vistas arquitectónicas propuestas por Kruchten [32] y el uso de la notación UML para la representación de la arquitectura de software. Esto facilita el entendimiento de los participantes en las sesiones evaluativas que propone el método. Sin embargo, como principal desventaja es que solo brinda atención al tratamiento del atributo de calidad modificabilidad y desecha atributos relevantes para la calidad del producto final como el rendimiento, la portabilidad y la confiabilidad del sistema.

3.4.4 Conclusiones parciales de los Métodos de Evaluación

Después de analizar cada uno de los métodos de evaluación, de manera general, se puede afirmar que los mismos, logran una coherencia en el orden de los pasos que proponen para realizar la evaluación de la arquitectura, y realizan un análisis preciso de los riesgos que representa el diseño arquitectónico seleccionado, para el desarrollo del sistema. Sin embargo, para la evaluación de la arquitectura propuesta en el capítulo anterior el método más conveniente a utilizar es el ATAM, ya que este método es considerado el más completo, porque 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 utiliza la técnica de evaluación basada en escenarios poco costosa para el equipo de desarrollo.

3.5 Evaluación de la arquitectura de software propuesta

Como se indicó anteriormente, para la evaluación de la arquitectura se utilizará el método de ATAM. Este no se aplicará en su totalidad debido a la compresión de los procesos descritos. En tal caso, solo se realizará la evaluación de la arquitectura basándose en los atributos de calidad más significativos del sistema.

Se determinaron los atributos de calidad que engloban la utilidad del sistema, es decir, los que poseen alta prioridad. La Fig. 23 muestra el árbol de utilidades formado por estos atributos y refinados a escenarios para percibir su comportamiento según el diseño arquitectónico.

Seguidamente se dio paso a analizar los escenarios obtenidos en el Árbol de Utilidades, los cuales se encuentran en el Anexo 2.

La evaluación de la arquitectura demostró que la utilización del framework Symfony, sus características y algunas herramientas integrables a este, como el plugin sfDoctrineGuardPlugin y el ORM Doctrine, dan cumplimiento a atributos de calidad como son la seguridad, portabilidad y confidencialidad. Dentro de las principales características del framework se destacan la capacidad de ser usado y configurado en múltiples plataformas, su capa de abstracción de datos dando soporte a varios gestores de bases de datos, su diseño modular permitiendo un bajo acoplamiento entre los componentes del sistema, logrando la modificabilidad y la modularidad.

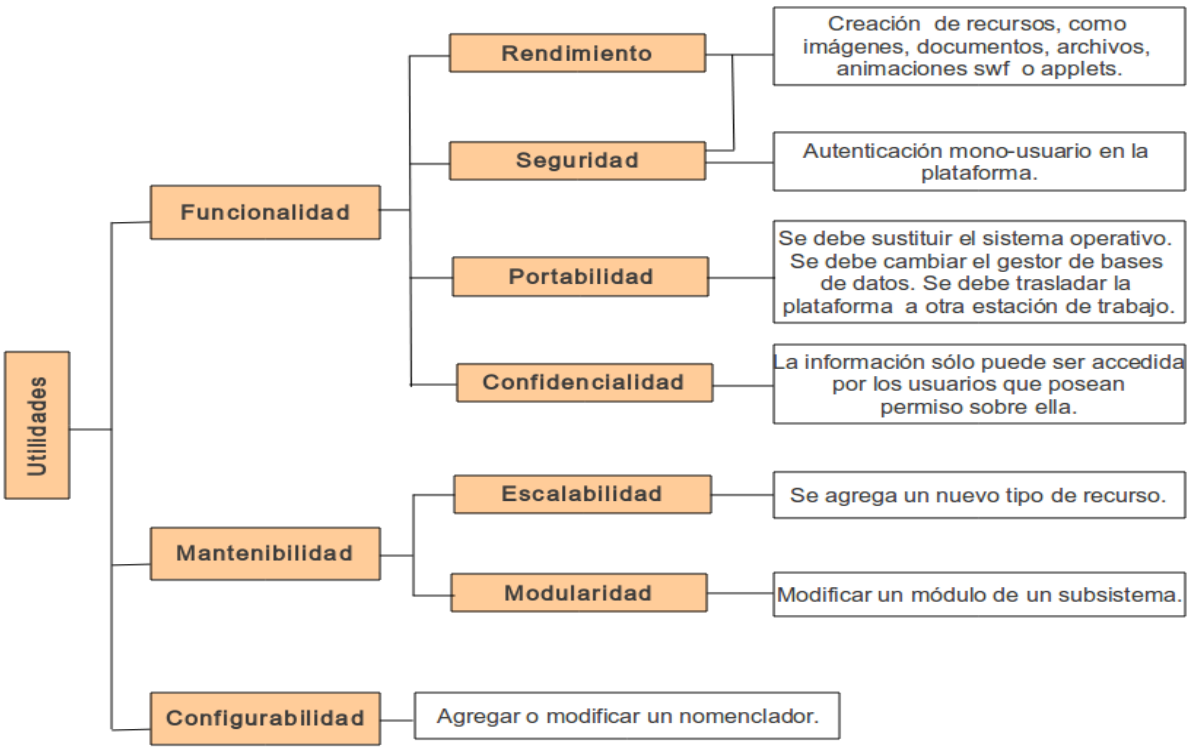


Fig. 23: Árbol de utilidades

3.6 Conclusiones

Se llega a la conclusión de que la evaluación de la arquitectura de software permite encontrar puntos sensibles, vulnerabilidades y fortalezas en el desarrollo de la aplicación. En este caso se obtuvieron cinco no riesgos, cinco puntos de sensibilidad, un punto de tradeoff y un riesgo por lo que se plantea que la arquitectura propuesta cumple con los requerimientos no funcionales del sistema.

En este capítulo se realizó un breve estudio de las tendencias actuales de la evaluación de la arquitectura de software, así como sus principales técnicas y métodos. De estas se seleccionaron las más propicias para evaluar la arquitectura de la plataforma de gestión del aprendizaje ZERA. Se analizaron brevemente

los atributos de calidad de la arquitectura propuesta, teniendo en cuenta el procedimiento seleccionado, refinando estos a escenarios y arrojando como resultado una serie de no riesgos y puntos sensibles de la arquitectura.

Conclusiones Generales

Al terminar la definición de la arquitectura de software para la plataforma de gestión del aprendizaje ZERA, se realiza un análisis de los resultados y el cumplimiento de los objetivos propuestos.

Se realizó el estudio del marco teórico acerca de las principales conceptos y estilos relacionados con la arquitectura de software, soluciones similares, así como el análisis de las herramientas y metodologías para llevar a cabo la implementación de esta. A partir de este análisis se desarrolló la propuesta y se arribó a las siguientes conclusiones:

Se adoptó como concepto de arquitectura de software la definición brindada por la IEEE, por ser esta amplia, abarcadora y proviene de una de las organizaciones con mayor prestigio y confianza a nivel mundial. El estudio de estilos y patrones permitió seleccionar como patrones arquitectónicos el Modelo Vista Controlador (MVC) y el Controlador Frontal, garantizando principalmente la organización, la escalabilidad y seguridad de la aplicación. Finalmente, con el análisis de las herramientas, tecnologías y metodologías se determinó Symfony como marco de trabajo, lo que propició la determinación de mejores definiciones arquitectónicas. La selección de RUP como metodología de desarrollo facilitó la descripción de los principales artefactos.

En el desarrollo de la arquitectura se estructuraron los subsistemas de diseño y se definieron las responsabilidades de estos. Se describieron las principales características utilizadas del Framework seleccionado, definiendo los componentes que conforman la linea base de la arquitectura, las librerías y se estandarizaron las funcionalidades comunes entre módulos. Se priorizaron los casos de usos arquitectónicamente significativos y se definieron los principales requerimientos no funcionales del sistema. Lo que proporcionó el desarrolló la linea base de la arquitectura.

Finalmente se realizó la evaluación de la arquitectura aplicando la técnica basada en escenarios con árbol de utilidades, el método ATAM. Esta evaluación permitió identificar los posibles puntos de riesgos y

sensibilidad. Con todo lo anterior se concluye que la arquitectura desarrollada soluciona las principales limitaciones planteadas en el estudio de la situación problémica.

Recomendaciones

Después de culminar el presente trabajo se plantean las siguientes recomendaciones:

- Refinar constantemente la arquitectura propuesta durante el ciclo de desarrollo del software.
- Realizar la propuesta de arquitectura para los restantes subsistemas de la plataforma.
- Profundizar en el estudio de los servicios web y las Arquitecturas Orientadas a Servicios con el objetivo de integrar la plataforma con otros sistemas externos.
- Realizar una evaluación más detallada de la arquitectura para identificar nuevos riesgos y debilidades, contribuyendo al fortalecimiento de la misma.

Referencias Bibliográficas

[1] A. M. Cañellas Cabrera, «Impacto de las TIC en la educación: un acercamiento desde el punto de vista de las funciones de la educación», *Quaderns digitals: Revista de Nuevas Tecnologías y Sociedad*, nº. 43, 2006.

[2] L. F. Ortiz F., «Campus Virtual: la educación más allá del LMS», *Revista de Universidad y Sociedad del Conocimiento, RUSC*, vol. 4, nº. 1, 2007.

[3] E. Chaviano Gómez y Y. A. Carrascoso Puebla, «Propuesta de Arquitectura Orientada a Servicios para el Módulo de Inventario del ERP Cubano», Universidad de las Ciencias Informáticas, 2008.

[4] L. Bass, P. Clements, y R. Kazman, *Software Architecture in Practice*, 2nd ed. Addison-Wesley, 2004.

[5] C. Billy Reynoso, «Introducción a la Arquitectura de Software». 2004.

[6] C. Billy Reynoso y N. Kiccillof, «Estilos y Patrones en la Estrategia de Arquitectura de Microsoft». 2004.

[7] C. Larman, L. M. H. Rodríguez, y H. C. Anaya, *UML y patrones: Introducción al análisis y diseño orientado a objetos*. Prentice Hall, 1999.

[8] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerland, y M. Stal, *Pattern – Oriented Software Architecture. A System ofPatterns.* Jhon Wiley & Sons Ltd, 1996.

[9] O. Salvador Gómez, *Evaluando Arquitecturas de Software. Parte 2.* México: Brainworx S.A, 2007.

[10] E. Gamma, R. Helm, R. Johnson, y J. Vlissides, *Design patterns: elements of reusable object-oriented software*, vol. 206. Addison-wesley Reading, MA, 1995.

[11] G. Díaz-Antón y M. Pérez, «Hacia una ontología sobre LMS», *Proceeding VII Jornadas Internacionales de las Ciencias Computacionales*, 2005.

[12] J. Adell y J. Gumbau, «Selección de un entorno virtual de enseñanza/aprendizaje de código fuente abierto para la Universitat Jaume I», *Disponible en: http://cent. uji. es/doc/eveauji_es. pdf [2006, Junio 20]*, 2004.

[13] M. N. Florentín Núñez, «Método Pedagógico en la EducaciónSuperior del Paraguay Utilizando elParadigma del Aprendizaje Mixto», Tesis de Maestría, Universidad Nacional de Pilar, 2008.

[14] J. A. Vela Dávila, «Ambientes de Desarrollo de Software Basado en Componentes».

[15] I. Jacobson, G. Booch, y J. Rumbaugh, *El Proceso Unificado de Desarrollo de Software*. Madrid: Addison-Wesley, 2000.

[16] C. Larman, *Applying UML and patterns: an introduction to object-oriented analysis and design and iterative development*. Prentice Hall PTR Upper Saddle River, NJ, USA, 2004.

[17] J. Eguíluz Pérez, *Introduccion al XHTML*, vol. 24. 2007.

[18] J. Eguíluz Pérez, *Introducción a CSS*, vol. 2. Librosweb, 2008.

[19] J. Eguíluz Pérez, *Introducción a JavaScript*. Librosweb, 2009.

[20] J. E. Villate, *Introducción al XML*. 2001.

[21] J. Eguíluz Pérez, *Introducción a AJAX*. Librosweb, 2008.

[22] P. R. Hinostroza, «Características de PHP», 2007.

[23] F. Potencier y F. Zaninotto, *Symfony, la guía definitiva*. Librosweb, 2008.

[24] E. Camacho, F. Cardeso, y G. Nuñez, «Arquitecturas de Software», *Guías de Estudio*, 2004.

[25] J. Bosch, *Design and use of software architectures: adopting and evolving a product-line approach.* 2000.

[26] P. Clements, R. Kazman, y M. Klein, *Evaluating software architectures: methods and case studies*, vol. 11. Addison-Wesley, 2002.

[27] M. Barbacci, M. H. Klein, T. A. Longstaff, C. B. Weinstock, y C.-M. U. P. P. S. E. INST, *Quality attributes*, vol. 6. Citeseer, 1995.

[28] M. H. Klein, R. Kazman, L. Bass, J. Carriere, M. Barbacci, y H. Lipson, «Attribute-based architecture styles», in *Software architecture: TC2 first Working IFIP Conference on Software Architecture (WICSA1): 22-24 February 1999, San Antonio, Texas, USA*, 1999, pág. 225.

[29] T. G. Lane, J. S. Herman, U. Signature, y O. File, «Studying software architecture through design spaces and rules», *The Computer Science Department, Carnegie Mellon University*, 2006.

[30] R. Kazman, M. Klein, y P. Clements, *ATAM: Method for architecture evaluation*. Citeseer, 2000.

[31] P. O. Bengtsson, «Architecture-level modifiability analysis (ALMA)», *Sweden : Department of Software Engineering and Computer Science. Blekinge Institute of Technology.*, 2008.

[32] P. Kruchten, *The Rational Unified Process: An Introduction.* Pearson Education, 2000.

Bibliografía Consultada

P. M. Moreno Clari, «Análisis del uso universitario de plataformas de gestión del aprendizaje. Estudio de caso en la Universitat de València», http://purl.org/dc/dcmitype/Text, Universitat de València, 2009.

R. T. Fielding, «Architectural Styles and the Design of Network-based Software Architectures», 2000.

L. G. Aretio, M. R. Corbella, y D. D. Figaredo, *De la educación a distancia a la educación virtual*. Editorial Ariel, 2007.

J. Bosch, *Design and use of software architectures: adopting and evolving a product-line approach.* 2000.

O. Salvador Gómez, *Evaluando Arquitecturas de Software. Parte 1. Panorama General.* México: Brainworx S.A, 2007.

S. G. Sánchez, «Revisión de plataformas de entorno de aprendizaje», *IX Encuentro Internacional Virtual Educa, Zaragoza*, 2008.

R. Kazman, L. Bass, M. Webb, y G. Abowd, «SAAM: A method for analyzing the properties of software architectures», in *Proceedings of the 16th international conference on Software engineering*, 1994, pág. 81–90.

L. Bass, P. Clements, y R. Kazman, *Software Architecture in Practice*, 2nd ed. Addison-Wesley, 2004.

M. Shaw y D. Garlan, *Software Architecture: Perspectives on an Emerging Discipline*. 1996.

F. Brooks, *The Mythical Man-Month*. Addison-Wesley, 1975.

J. R. Gómez, «TIC y educación», *Las TIC en educación*, 2004. [Online]. Available: http://boj.pntic.mec.es/jgomez46/ticedu.htm. [Accessed: 13-Dic-2010].

Glosario de Términos

Administrador: Usuario que generaliza las acciones que realizan el administrador central y el administrador local.

Administrador central: Usuario que se encarga de la administración global del sitio y tiene permisos sobre los principales componentes de la plataforma.

Administrador local: Es el usuario que realiza las tareas de administración en la plataforma instalada en la escuela.

Cookies: Archivo de texto que se graba en el ordenador del visitante del cual se sirven los servidores web para guardar información acerca del cliente de un sitio. Sirve para identificar a visitantes recurrentes.

CRUD: En el área de la informática es el acrónimo de Crear, Obtener, Actualizar y Borrar los datos de una base de datos.

Escala de calificación: La escala de calificación mide la capacidad de una persona para producir algo o realizar una operación, es definida como un rango.

Hiperentorno: Un hiperentorno de aprendizaje se define como una modalidad informática que se sustenta en la tecnología hipermedia y en el que están presentes un conjunto de elementos representativos de diversas tipologías de software educativo. Estos tienen en común la utilización de un ambiente informático lo más didáctico posible, integrando a estudiantes y profesores con las facilidades que ofrecen los sistemas informáticos, esto proporciona condiciones de interacción local o través de redes, así como posibilidades de acceso a recursos formativos, locales o distribuidos.

Macro índice: Índice de contenido global que estructuran los contenidos educativos de las materias. Contienen capítulos, temas y subtemas, así como recursos, sugerencias de evaluación y recomendaciones de uso.

Materia: Conjunto de contenidos específicos comunes a un área específica del conocimiento, agrupados bajo una denominación genérica. En un plan de estudios una materia puede ser objeto del tratamiento de uno o varios cursos o asignaturas.

ORM (Mapeo objeto-relacional): es una técnica para convertir datos del lenguaje de programación orientado a objetos al utilizado en una base de datos relacional, esto facilita el uso de las características propias de la orientación a objetos.

Periodo evaluativo: Es el período de tiempo en el cual los estudiantes tendrán evaluaciones.

Período lectivo: Es el periodo de tiempo por el cual el estudiante va a tener acceso a la materia. Se asocia a uno o varios programas de estudio.

Programa de Estudio: Son creados a partir del sistema curricular. Es la adecuación del macro índice al plan de estudios del sistema educativo y la institución que adquiere el software.

Sistema Curricular: Contiene todos los contenidos educativos que imparte una escuela. Puede estar asociado a uno o varios sistemas educativos.

Sistema Educativo: Es un concepto inserto en el contexto de las prácticas pedagógicas, de la organización política, social y económica de un país. Su estructura y organización interna está fuertemente vinculada a otros conceptos como: sistema curricular, período lectivo, programa de estudio, escala de calificación y período evaluativo.

Software Educativo: Se refiere a los programas educativos o didácticos, conocidos también como programas por ordenador, creados con la finalidad específica de ser utilizados para facilitar los procesos de enseñanza y aprendizaje .

Tipología de ejercicio: Es el formato en el que se puede presentar un determinado ejercicio. Para la plataforma de gestión de aprendizaje ZERA pueden ser: Selección simple, Selección múltiple, Seleccionar

texto, Completar por escritura, Completar por desplazamiento, Ordenar Procedimiento, Ordenar Medias, Enlazar, Clasificar Texto, Clasificar Media.

Anexos

Anexo 1 Descripción del los plugins utilizados en la plataforma

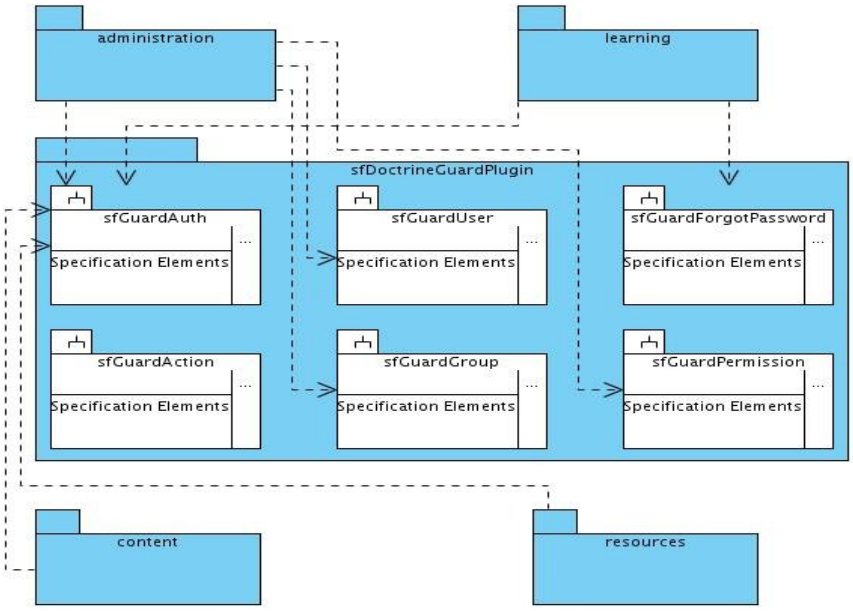
Nompre plugin	sfDoctrineGuardPlugin
Propósito	Encargado de los procesos fundamentales asociados a la seguridad de la aplicación. Permite incluir autenticación, autorización y otras opciones de gestión de usuarios más avanzadas que las que brinda symfony por defecto.
Módulos	SfGuardAuth, sfGuardUser, sfGuardGroup, sfGuardPermission, sfGuardAction
Diagrama de dependencias entre el plugin y los subsistemas	
	
Forma de uso	Añadir los módulos en la opción “enabled_modules' del archivo de configuración “settings.yml” que se encuentra en la carpeta config/ de la aplicación symfony donde se quiere usar el plugin.

Tabla 7: Descripción del plugin sfDoctrineGuardPlugin

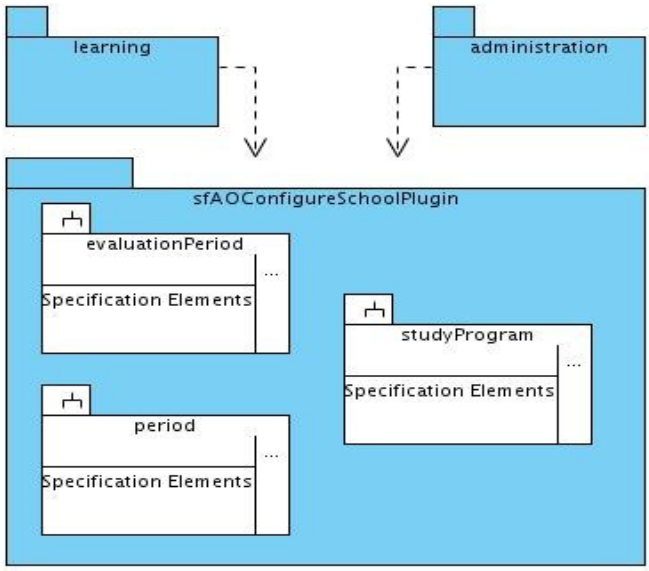
Nompre plugin	sfAOConfigureSchoolPlugin
Propósito	Agrupa funcionalidades comunes presentes en las aplicaciones *learning* y *administration*, como la gestión de períodos lectivos, períodos evaluativos y configuración de programas de estudios.
Módulos	EvaluationPeriod, period, studyProgram
Diagrama de dependencias entre el plugin y los subsistemas	
	
Forma de uso	Añadir los módulos en la opción “enabled_modules' del archivo de configuración “settings.yml” que se encuentra en la carpeta config/ de la aplicación symfony donde se quiere usar el plugin.

Tabla 8: Descripción del plugin sfAOConfigureSchoolPlugin

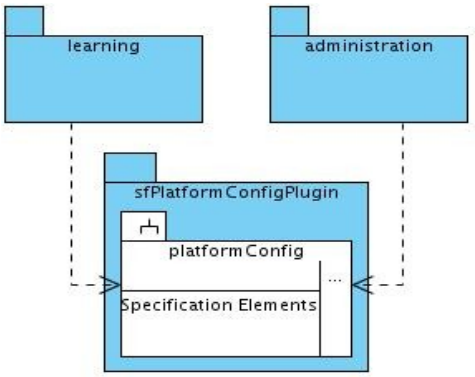
Nompre plugin	sfPlatformConfigPlugin
Propósito	Responsable de la configuración de la plataforma tanto en una institución educativa como en la aplicación central de Alfaomega.
Módulos	PlatformConfig
Diagrama de dependencias entre el plugin y los subsistemas	
	
Forma de uso	Añadir los módulos en la opción “enabled_modules' del archivo de configuración “settings.yml” que se encuentra en la carpeta config/ de la aplicación symfony donde se quiere usar el plugin.

Tabla 9: Descripción del plugin sfPlatformConfigPlugin

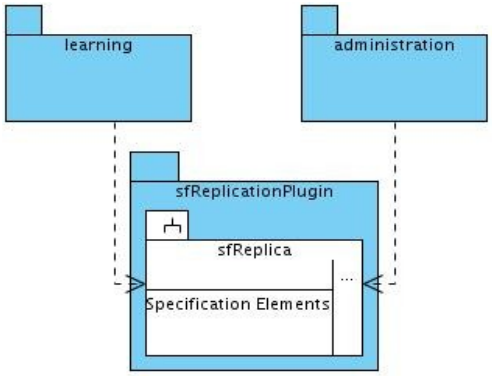
Nompre plugin	sfReplicationPlugin
Propósito	Encargado de gestionar la sincronización y réplica de los datos entre servidores. Se configura tanto en la escuela como en el servidor central de aplicación.
Módulos	sfReplica
Diagrama de dependencias entre el plugin y los subsistemas	
	
Forma de uso	Añadir los módulos en la opción “enabled_modules' del archivo de configuración “settings.yml” que se encuentra en la carpeta config/ de la aplicación symfony donde se quiere usar el plugin.

Tabla 10: Descripción del plugin sfReplicationPlugin

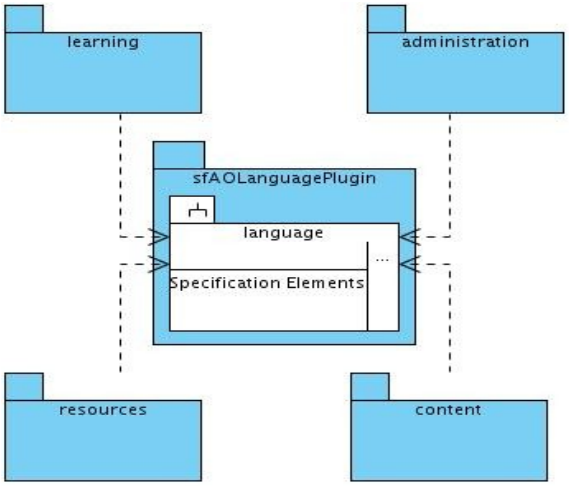
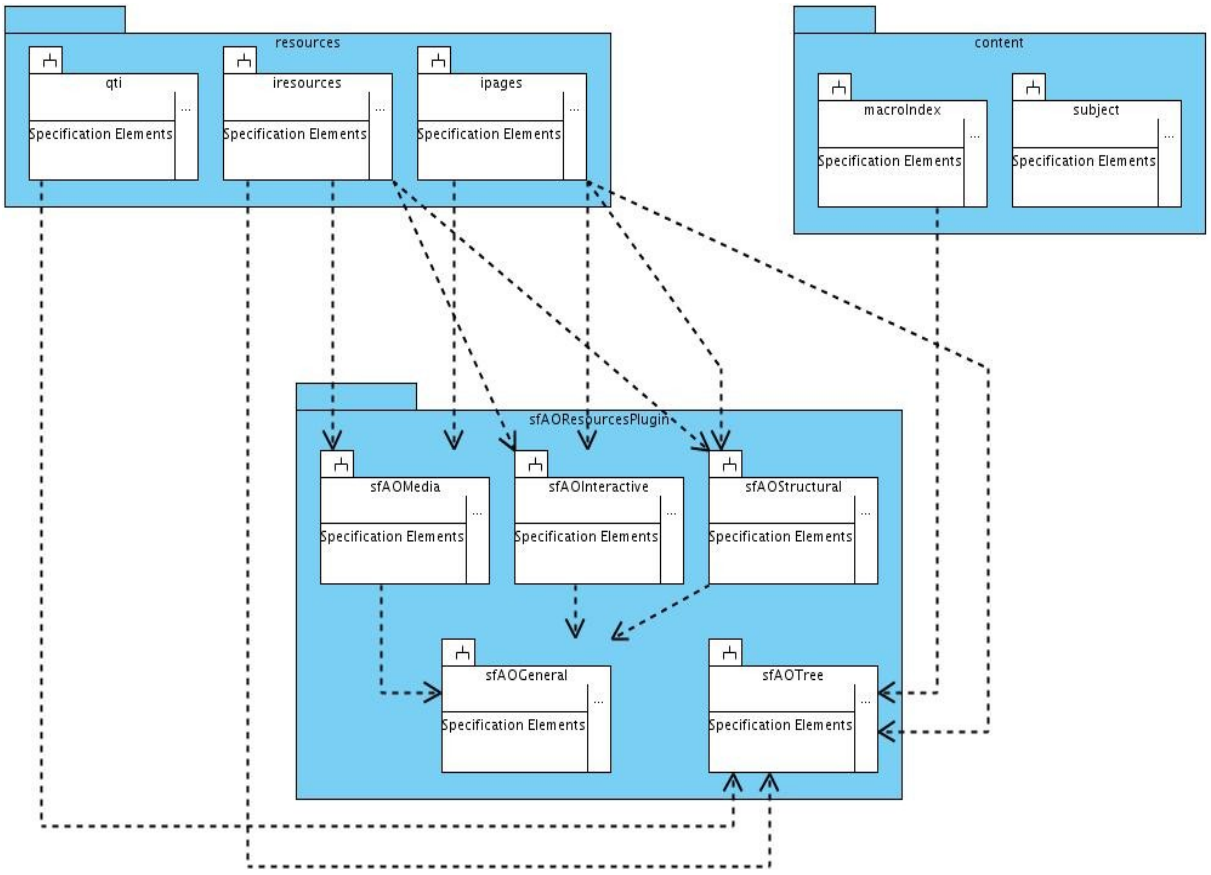
Nompre plugin	sfAOLanguagePlugin
Propósito	Permite la internacionalización de la plataforma.
Módulos	language
Diagrama de dependencias entre el plugin y los subsistemas	
	
Forma de uso	Añadir los módulos en la opción “enabled_modules' del archivo de configuración “settings.yml” que se encuentra en la carpeta config/ de la aplicación symfony donde se quiere usar el plugin. Incluir el componente en la plantilla para poder cambiar de lenguaje en la plataforma, mediante: **<?php include_component**('language', 'language') **?>**

Tabla 11: Descripción del plugin sfAOLanguagePlugin

Nompre plugin	sfJqueryReloadedPlugin
Propósito	Permite incluir el framework de javascript jQuery y varios de sus componentes para la interfaz de usuario.
Forma de uso	Habilitar el helper por defecto del plugin en el archivo de configuración “settings.yml” que se encuentra en la carpeta config/ de la aplicación que usará el plugin. Esto se realiza agregándolo en la opción “standard_helpers”. Para cargar algún componente adicional de jQuery en las plantillas se utiliza la función: **<?php jq_add_plugins_by_name**(**array**(“alerts”)) **?>** Si se quiere incluir el componente básico de interfaz de usuario de jQuery basta con: **<?php jq_add_my_ui() ?>.** En caso de surgir uno adicional que aún no forme parte de los componentes básicos, se puede agregar con: **<?php jq_add_my_ui(array**(“daterangepicker”)**) ?>**

Tabla 12: Descripción del plugin sfJqueryReloadedPlugin

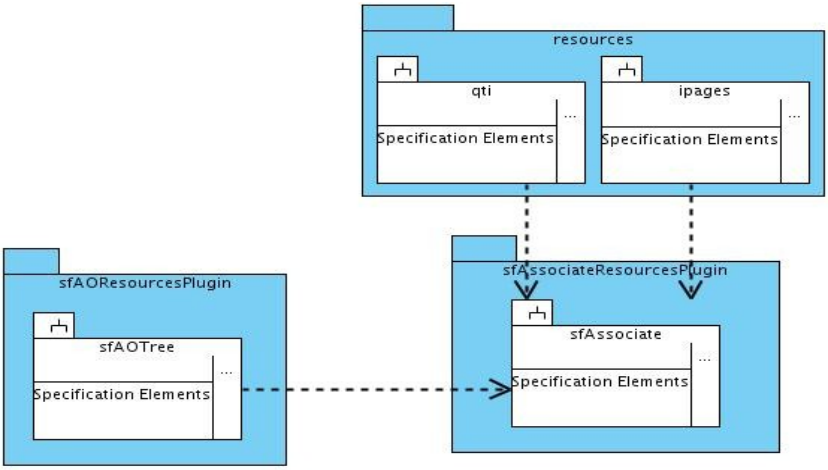
Nompre del plugin	sfAOResourcesPlugin
Propósito	Se encarga de la visualización de los recursos en los diferentes subsistemas de la plataforma, así como del árbol de macro Índice.
Módulos	sfAOMedia, sfAOGeneral, sfAOInteractive, sfAOEstructural, sfAOTree
Diagrama de dependencias entre el plugin y los subsistemas	



Forma de Uso	Añadir los módulos en la opción “enabled_modules' del archivo de configuración “settings.yml” que se encuentra en la carpeta config/ de la

	aplicación symfony donde se quiera usar. Los módulos sfAOMedia, sfAOInteractive y sfAOStructural implementan componentes los cuales son utilizados en cualquier vista de una aplicación symfony que los habilite. El módulo sfAOGeneral implementa funcionalidades comunes entre los anteriores. Este módulo se encarga de mostrar el árbol del macro Índice con un menú contextual que se lanza al dar click derecho sobre un elemento del árbol. Este menú permite la integración de funcionalidades entre los subsistemas content y resources. El árbol además define comportamientos los cuales varían en el contexto en que es ejecutado, para ello cuando se incluye recibe una serie de parámetros definidos, donde el desarrollador que lo necesite usar es el encargado de especificar los valores correctos para el comportamiento que se desee. También es desarrollado en forma de componente y se incluye igual que los módulos anteriores.

Tabla 13: Descripción del plugin sfAOResourcesPlugin

Nompre del plugin	sfAssociateResources
Propósito	Brinda la opción de buscar y asociar recursos existentes a otras estructuras del macro Índice.
Módulos	sfAssociate
Diagrama de dependencias entre el plugin y los subsistemas	
	
Forma de Uso	Añadir el módulo en la opción “enabled_modules' del archivo de configuración “settings.yml” que se encuentra en la carpeta config/ de la aplicación symfony donde se quiera usar. Implementa el explorador y asociador de recursos. Esta herramienta puede utilizarse incrustadas en una vista o con la ayuda de plugins de jQuery pueden ser lanzadas en una nueva ventana hija. Puede ser configurado para ser usado en diferentes ambientes, el explorador puede ser ejecutado para solamente buscar recursos y no asociar. Además, permite la opción de seleccionar varios recursos o uno solo, de explorar todos los recursos o los que desee el desarrollador. Para la

	ejecución del asociador siempre se necesita haber explorado y seleccionado al menos un recurso antes.

Anexo 2 Análisis de los Escenarios para la Evaluación de la Arquitectura.

Escenario	Autenticación mono-usuario en la plataforma.			
Atributo	Seguridad.			
Ambiente	Condiciones normales.			
Estímulo	Un usuario se autentica en la plataforma.			
Respuesta	Si los datos de autenticación están correctos el usuario se registra y caduca cualquier otra sesión del mismo usuario que estuviese activa en ese momento.			
Decisiones arquitectónicas	Riesgo	Sensibilidad	Tradeoff	No Riesgo
Implementación del filtro sfGuardBasicSecurityFilter en el plugin sfDoctrineGuardPlugin		S1		N1
Explicación	El uso del plugin sfDoctrineGuardPlugin gestiona de una forma más automatizada la autenticación de los usuarios en la plataforma. Además se implementó un filtro de seguridad para validar que los usuario sólo tuvieran una sesión activa en la plataforma, siendo este filtro crítico para el cumplimiento de este atributo.			

Tabla 14: Análisis del escenario: " Autenticación mono-usuario en la plataforma ".

Escenario	Creación de recursos, como imágenes, documentos, archivos, animaciones swf o applets.			
Atributo	Seguridad y Rendimiento.			
Ambiente	Condiciones normales.			
Estímulo	Un profesor necesita crear un recurso para asociarlo a los contenidos educativos.			
Respuesta	El recurso se encripta y se almacena en la base de datos.			
Decisiones arquitectónicas	Riesgo	Sensibilidad	Tradeoff	No Riesgo
Creación de librerías PHP para la codificación y decodificación de recursos.			T1	
Explicación	La codificación y decodificación de los recursos, además de garantizar la seguridad con el encriptado de estos trae consigo que se afecte el rendimiento de la aplicación ya que este proceso puede resultar un poco lento con altas concurrencias de usuarios. La relación de estos dos atributos de calidad en este escenario representa un punto de Tradeoff.			

Tabla 15: Análisis del escenario: "Creación de recursos, como imágenes, documentos, archivos, animaciones swf o applets".

Escenario	Se debe sustituir el sistema operativo. Se debe cambiar el gestor de bases de datos. Se debe trasladar la plataforma a otra estación de trabajo.			
Atributo	Portabilidad.			
Ambiente	El cliente solicita el cambio de servidores y requerimientos diferentes a los actuales variando el sistema operativo y el gestor de bases de datos.			
Estímulo	Se necesitan cambiar los servidores donde se encuentra instala la aplicación, los cuales poseen otro sistema operativo y gestor de bases de datos.			
Respuesta	La plataforma continúa 100% funcional.			
Decisiones arquitectónicas	Riesgo	Sensibilidad	Tradeoff	No Riesgo
Uso del framework Symfony				N2
Uso del ORM Doctrine				N3
Explicación	Una de las principales características que posee el framework de desarrollo utilizado (Symfony) es precisamente que es fácil de instalar y configurar en la mayoría de plataformas existentes, atributo que es heredado del lenguaje de programación con que está confeccionando, en este caso PHP. Con el uso del ORM Doctrine y la capa de abstracción de datos que posee Symfony nos permite cambiar en mitad de un proyecto de una base de datos MySQL a una Oracle sin ningún tipo de complicación. Esto es debido a que no se utiliza una sintaxis MySQL u Oracle para acceder a nuestro modelo, sino una sintaxis propia del ORM utilizado que es capaz de traducir a diferentes tipos de bases de datos.			

Tabla 16: Análisis del escenario: " Se debe sustituir el sistema operativo. Se debe cambiar el gestor de bases de datos. Se debe trasladar la plataforma a otra estación de trabajo ".

Escenario	La información sólo puede ser accedida por los usuarios que posean permiso sobre ella.			
Atributo	Confidencialidad.			
Ambiente	Uso de la plataforma en condiciones normales.			
Estímulo	Acceder o modificar información .			
Respuesta	Según los permisos del usuario este puede acceder o gestionar determinada información .			
Decisiones arquitectónicas	Riesgo	Sensibilidad	Tradeoff	No Riesgo
Uso del framework Symfony		S2		N4
Explicación	La confidencialidad se hace a través del framework Symfony el cual trabaja con las sesiones de usuarios. Para asegurar la aplicación en los puntos que se desee, Symfony tiene un sistema de credenciales que son requeridas para cada petición, los usuarios que no las posean no podrán acceder.			

Tabla 17: Análisis del escenario: " La información sólo puede ser accedida por los usuarios que posean permiso sobre ella ".

Escenario	Agregar o modificar un nomenclador.			
Atributo	Configurabilidad.			
Ambiente	Condiciones normales.			
Estímulo	Se necesita agregar un nuevo nomenclador o cambiar el nombre de uno existente por modificaciones en los requerimientos.			
Respuesta	El nomenclador se agrega o se modifica.			
Decisiones arquitectónicas	Riesgo	Sensibilidad	Tradeoff	No Riesgo
Implementación de la librería sfNomenclature		S3		
Concepción de un módulo para la gestión de nomencladores	R1			
Explicación	Los nomencladores se almacenan en tablas de la base de datos, los cuales son utilizados prácticamente en todos los subsistemas desarrollados. Para ello se crea la librería sfNomenclature la cual propone una capa de abstracción entre el nombre del nomenclador almacenado en la base de datos y el desarrollador que lo usa en la implementación de cualquier funcionalidad. Para la inserción o modificación de un nomenclador se necesita un módulo que agrupe estas funcionalidades, el cual representa un riesgo porque no está contemplado en la arquitectura.			

Tabla 18: Análisis del escenario: "Agregar o modificar un nomenclador".

Escenario	Se agrega un nuevo tipo de recurso.			
Atributo	Escalabilidad.			
Ambiente	Surge la necesidad de incorporar un tipo nuevo de recurso a la plataforma.			
Estímulo	Se necesita agregar un tipo nuevo de recurso.			
Respuesta	Se agrega la funcionalidad al módulo encargado de gestionar los recursos.			
Decisiones arquitectónicas	Riesgo	Sensibilidad	Tradeoff	No Riesgo
Uso del framework Symfony y el ORM Doctrine				N5
Utilizar PHP y Javascript para agregar la nueva funcionalidad		S4		
Explicación	Para agregar un nuevo tipo de recurso se necesita añadir las tablas al modelo, las cuales son mapeadas con el ORM sin afectar el diseño actual. Pero se necesita agregar a todas las funcionalidades existentes este nuevo tipo de recurso, utilizando los lenguajes de desarrollo seleccionados, lo cual significa una serie de cambios a lo largo del módulo y la plataforma.			

Tabla 19: Análisis del escenario: " Se agrega un nuevo tipo de recurso ".

Escenario	Modificar un módulo de un subsistema.			
Atributo	Modularidad.			
Ambiente	Necesidad de modificar un módulo de la plataforma.			
Estímulo	Modificar un módulo .			
Respuesta	Los otros módulos del subsistema y el resto de la plataforma no deben ser afectados.			
Decisiones arquitectónicas	Riesgo	Sensibilidad	Tradeoff	No Riesgo
Diseño de la aplicación en subsistemas		S5		
Explicación	Al dividir el desarrollo en subsistemas, se estableció que unos fueran independientes de otros, lo que constituye un punto de sensibilidad debido a que algunos de estos subsistemas puedan depender necesariamente de las funcionalidades de otro, aunque sea en pocas ocasiones. Si sucediera esto hay que tener bien identificado los puntos de contacto para mantener los subsistemas funcionales. El diseño modular permite una gran mantenibilidad del sistema ya que los subsistemas están bien desacoplados.			

Tabla 20: Análisis del escenario: " Modificar un módulo de un subsistema ".