Universidad de las Ciencias Informáticas

Facultad 3

Propuesta de Arquitectura para la Herramienta de Minería de Datos de PATDSI.

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

Autores:

Thais Casares Gonzalez

José Miguel Cazorla Santana

Tutor:

Ing. Yunior Miguel Almaguer Bajuelo

Cotutor:

Ing. Yasmany Molina Díaz

Junio 2009

Declaración de Autoría.

Declaro que soy el único autor de este trabajo y autorizo a la Dirección de la Universidad de las Ciencias Informáticas (UCI) a hacer uso del mismo en su beneficio. Para que así conste firmo la presente a los ____ días del mes de _____ del año _____.

_____ _____.

Thais Casares Gonzalez José Miguel Cazorla Santana

_____

Ing. Yunior Miguel Almaguer Bajuelo

"El conocimiento es experiencia, todo lo demás es información"

Albert Einstein

Dedicatoria

A mimí y a pipí por ser las personas que me han impulsado y apoyado siempre en la vida, gracias por quererme tanto.

A mi flaqui linda por ser la mejor hermana del mundo, te quiero mucho mucho.

A mis abuelos Ana, Frank y Deisy y a toda mi familia.

A mis amigas Mh, Mary y Ali por ser tan especiales y saber siempre sobrellevarme, incluso en esos malos días.

A mis queridos amigos del Team: Enrique, Nahuel y Tico.

A mi nene lindo, gracias por aguantarme estos cinco años y estar a mi lado siempre, te amo mucho.

A todas aquellas personas que de una forma u otra contribuyeron a hacer mejores estos cinco años de mi vida.

Thais

A mi mamá y a mi papá por la educación que me han dado, por la confianza que siempre han depositado en mí y por ser mis guías e inspiración.

A mis hermanos, Bety por ser lo más grande que tengo y Víctor por ser tan especial en mi vida y sentirme tan orgulloso de él.

A mis abuelos, Blanquita y Tino que los quiero mucho y Eva y Ñoño que aunque no están yo los llevo muy dentro.

A todos mis primos y tíos, que son muchos pero siempre están presentes.

A Thais, mi amor, por estos años lindos que hemos pasado juntos y los que nos faltan.

A mis amigos y todos los que han compartido conmigo estos 5 años inolvidables, especialmente a MH, Mary, Ali, Enrique y Topiz que tantos buenos momentos hemos pasado.

José M.

Agradecimientos

A nuestros padres, hermanos, familiares y amigos.

A nuestro tutor por su apoyo y ayuda.

A todos los que de una forma u otra contribuyeron al desarrollo de este trabajo.

Resumen

Este trabajo es una propuesta arquitectónica para la Herramienta de Minería de Datos de PATDSI. Este sistema será desarrollado en el Centro de Tecnología de Almacenamiento y Análisis de Datos. Para ello se estudiarán y seleccionarán las mejores tecnologías, herramientas y lenguajes de programación, utilizados para el desarrollo de aplicaciones bajo la política de software libre. Al mismo tiempo se tomarán ideas de las aplicaciones de Minería de Datos más potentes hoy en día en el mundo con el objetivo de entender las funcionalidades principales que brindan este tipo de herramientas.

Como resultado final se obtendrá la propuesta arquitectónica para la herramienta antes mencionada y la evaluación de la misma.

Índice

Índice de Figuras

Índice de Tablas

Introducción

Algo peor que no tener información disponible es tener mucha y no saber qué hacer con ella. Este problema surge a partir de la década de los 70's cuando los sistemas basados en servidores y las bases de datos relacionales se convirtieron en la moda de los departamentos de sistemas de información de las organizaciones. A partir de ese entonces cada año en el mundo se multiplica la cantidad de datos almacenados. Un gran porcentaje de los datos generados representan "hechos" que diariamente se están registrando, tales como: transacciones financieras, operaciones de compra y venta, préstamos o devoluciones y movimientos de almacén. Vivimos en una época en que la información es la clave para obtener una ventaja competitiva en el mundo de los negocios.

El siglo XXI se enfrenta a la creciente implantación de la sociedad del conocimiento. La nueva era en que se encuentra el mundo no solo está cambiando la sociedad en sí misma, sino que los recientes modelos de negocios requieren la reformulación de nuevos conceptos. Inteligencia, vigilancia tecnológica, estudio de mercado, activos intangibles, son algunos de los términos más utilizados en cualquier ambiente o negociación. Este nuevo momento que el mundo enfrenta hoy, requiere también de nuevas tendencias apoyadas precisamente en el conocimiento. Los innumerables avances en las tecnologías han contribuido a que el conocimiento sea considerado como un nuevo recurso, generador de importantes ventajas competitivas.

El proceso de toma de decisiones en toda empresa, independientemente de la envergadura no es tarea fácil, porque cualquier cambio es riesgoso, si bien debemos aceptarlos cuando se establecen, también debemos comprender que esos cambios pueden ser muy beneficiosos. Para ello hoy en día se manejan grandes volúmenes de datos que interpretados con inteligencia pueden ser utilizados para detectar áreas de oportunidad o crear nuevas estrategias para las organizaciones que han estado colectándolos durante años. Convertir los datos en información limpia, útil para el análisis y el apoyo en la toma de decisiones, es una tarea compleja, de aquí la creciente necesidad de una nueva generación de técnicas computacionales y sistemas informáticos para apoyar la extracción de conocimiento útil.

Estas técnicas y sistemas son el centro del emergente y dinámico campo de investigación denominado: Descubrimiento de conocimiento en Bases de Datos (KDD, en ingles Knowledge Discovery in DataBases) y Minería de Datos (DM, en inglés Data Mining).

Minería de Datos es un área de la computación que surgió hace mas de 20 años con el objetivo de lidiar con grandes volúmenes de datos, y a partir de 1996 fue definida como parte del proceso de KDD, mucho más amplio y complejo.

Aunque desde un punto de vista académico el término Data Mining es una etapa dentro del proceso mayor KDD, en el entorno comercial, ambos términos se usan de manera indistinta. Lo que en verdad hace el Data Mining es reunir las ventajas de varias áreas como la Estadística, la Inteligencia Artificial, la Computación Gráfica, las Bases de Datos y el Procesamiento Masivo, principalmente usando como materia prima las bases de datos.

La Universidad de las Ciencias Informáticas (UCI), centro de altos estudios en ciencias informáticas, se adentra cada vez más en la producción de software empresarial con el reto de convertir a la industria de software cubana en un renglón fundamental de la economía del país. Dentro de la infraestructura productiva de la UCI se encuentran los centros de producción los que se especializan en distintos productos y servicios informáticos, uno de ellos es el Centro de Tecnologías Almacenamiento y Análisis de Datos (CENTALAD).

CENTALAD tiene por misión proveer soluciones integrales y consultorías relacionadas con tecnologías de bases de datos y análisis de información, además de desarrollar nuevas tecnologías de bases de datos y de procesamiento y representación de la información.

Este centro con el objetivo de hacerse de herramientas propias, que le permitan cumplir con la misión que se ha trazado, ha creado el proyecto Paquete de Herramientas para la Ayuda en la Toma de Decisiones (PATDSI), el cual ya cuenta con herramientas tales como un generador de reportes y un sistema de gestión estadística, pero se hace necesario que el mismo cuente con una herramienta que permita procesar automáticamente grandes cantidades de datos crudos e identificar los patrones más significativos y relevantes y presentar los mismos como conocimiento apropiado para satisfacer las metas del usuario.

Se necesita, por lo tanto, de una arquitectura de software que permita el desarrollo de esta herramienta.

Por todo lo antes mencionado se define el **problema** de esta investigación con la siguiente interrogante: ¿Cómo diseñar una arquitectura que garantice la estructura, comportamiento y colaboración entre los componentes de la Herramienta de Minería de Datos de PATDSI?

Por consiguiente el **objeto de estudio** de este trabajo es proceso de desarrollo de software de la herramienta de minería de datos de PATDSI y el **campo de acción** es diseño de la arquitectura de software.

Constituye **Objetivo General** del trabajo diseñar una arquitectura robusta, flexible y escalable para la herramienta de minería de datos de PATDSI.

La **hipótesis** planteada por este trabajo es: Si se diseña una arquitectura de software robusta, flexible y escalable haciendo uso correcto de estilos y patrones arquitectónicos, se contribuirá a la estructura, comportamiento y colaboración entre los componentes de la Herramienta de Minería de Datos de PATDSI.

En beneficio de obtener un trabajo de mayor calidad se trazaron **tareas de investigación**, las cuales se muestran a continuación:

- Realizar un estudio detallado sobre la arquitectura de software y conceptos relacionados con la misma.
- Realizar un estudio sobre las herramientas de minería de datos más potentes en la actualidad.
- Desarrollar el Documento Descripción de la Arquitectura.
- Estudiar pruebas existentes para evaluar las arquitecturas de software.
- Evaluar la arquitectura propuesta.

Durante la realización de este trabajo se hizo uso de diferentes **métodos de investigación científica** a los cuales se hace alusión a continuación:

Métodos teóricos:

- **Analítico-sintético:**

Este método se utilizó con el objetivo de procesar la información y arribar a las conclusiones de la investigación, así como precisar las características del modelo arquitectónico propuesto.

- **Histórico-Lógico:**
  Se utilizó para estudiar la evolución histórica y desarrollo del objeto de estudio de la investigación, así como para determinar las tendencias actuales de desarrollo de los modelos y enfoques arquitectónicos.
- **Inducción-deducción:**
  Es utilizado para estudiar los distintos modelos y estilos arquitectónicos existentes y deducir cual es el más apropiado, o sea proponer estilos de arquitecturas específicas.
- **Método Sistémico:**
  Este método se utilizó para definir los componentes de la arquitectura y las relaciones entre ellos.

El presente documento se estructura en 3 capítulos que incluye lo relacionado con el trabajo investigativo realizado:

El Capitulo 1: Fundamentación Teórica, Se abordan brevemente los conceptos fundamentales de Inteligencia de negocio, así como un estudio del estado del arte de la arquitectura de software, estilos y patrones arquitectónicos más utilizados en la actualidad.

El Capitulo 2: Diseño arquitectónico: En este capítulo se hace una propuesta de solución al problema planteado en el diseño teórico de la investigación. La solución propuesta se centra en el Documento Descripción de la Arquitectura, haciendo énfasis en los puntos principales del mismo.

El Capitulo 3: Evaluación de la solución: En este capítulo se hace un estudio de los principales métodos existentes para evaluar la arquitectura de software, así como la evaluación, utilizando uno de los métodos estudiados, de la solución propuesta.

Capítulo 1

Fundamentación Teórica

1.1. Introducción

En este capítulo se realiza una fundamentación teórica de los aspectos a tratar en el resto del documento. Se abordan conceptos relacionados con la Minería de Datos y las herramientas más potentes existentes hoy día para realizarla; así como conceptos de arquitectura de software y las tendencias de la misma, breve descripción de los patrones y estilos arquitectónicos más utilizados en la actualidad, las vistas arquitectónicas, el lenguaje de modelado y la metodología de desarrollo a utilizar, así como las principales responsabilidades de un arquitecto de software.

1.2. Data Mining (Minería de Datos)

Las empresas suelen generar grandes cantidades de información sobre sus procesos productivos, desempeño operacional, mercados y clientes. Pero el éxito de los negocios depende por lo general de la habilidad para ver nuevas tendencias o cambios en las mismas.

La minería de datos consiste en la extracción no trivial de información que reside de manera implícita en los datos. Las aplicaciones de Data Mining pueden identificar tendencias y comportamientos, no sólo para extraer información, sino también para descubrir las relaciones en bases de datos que pueden identificar comportamientos que no son muy evidentes.

Una definición tradicional de Minería de Datos es la siguiente: Un proceso no trivial de identificación válida, novedosa, potencialmente útil y entendible de patrones comprensibles que se encuentran ocultos en los datos (1). Desde el punto de vista empresarial, se define como: La integración de un conjunto de áreas que tienen como propósito la identificación de un conocimiento obtenido a partir de las bases de datos que aporten un sesgo hacia la toma de decisión (2).

Los análisis prospectivos automatizados ofrecidos por la automatización del Data Mining van más allá de los eventos pasados provistos por las herramientas usuales de sistemas de soporte de decisión.

Las herramientas de Data Mining pueden responder a preguntas de negocios que tradicionalmente consumen demasiado tiempo para poder ser resueltas.

1.3. Herramientas existentes de Minería de Datos

Para la realización de este trabajo se tuvieron en cuenta algunas de las herramientas existentes para la Minería de Datos con el objetivo de estudiar y profundizar en las funcionalidades y potencialidades de las mismas.

Las herramientas seleccionadas son: SPSS Clementine, Oracle Data Miner, Orange y WEKA.

1.3.1. WEKA

WEKA (Waikato Environment for Knowledge Analysis - Entorno para Análisis del Conocimiento de la Universidad de Waikato) fue desarrollada en la Universidad de Waikato (Nueva Zelanda) bajo licencia GPL y en java, lo cual ha impulsado que sea una de las herramientas más utilizadas en el área en los últimos años.

Una de las propiedades más interesantes de este software, es su facilidad para añadir extensiones, modificar métodos:

- Es muy portable porque está completamente implementado en Java y puede correr en casi cualquier plataforma.
- Contiene una extensa colección de técnicas para pre-procesamiento de datos y modelado.

- Es fácil de utilizar por un principiante gracias a su interfaz gráfica de usuario.

Además de lo anteriormente dicho WEKA soporta varias de las tareas estándar de minería de datos especialmente pre-procesamiento de datos, clustering, clasificación, regresión, visualización, y selección.

Concretamente WEKA permite cargar los datos para analizar desde una base de datos, un fichero .csv o ficheros .arff (el formato propio de WEKA), además de proporcionar acceso a bases de datos vía SQL gracias a la conexión JDBC (Java Database Connectivity) y puede procesar el resultado devuelto por una consulta hecha a la base de datos.

No puede realizar minería de datos multi-relacional, pero existen aplicaciones que pueden convertir una colección de tablas relacionadas de una base de datos en una única tabla que ya puede ser procesada con WEKA. (3)

1.3.2. SPSS Clementine

Clementine es una de las soluciones líderes en minería de datos que permite acceder a datos de varias fuentes para producir, evaluar, y desplegar modelos analíticos rápida y fácilmente.

La arquitectura escalable y abierta de este producto permite obtener el máximo provecho de la infraestructura actual, haciendo de la minería de datos un proceso efectivo en todas las empresas.

Clementine permite que se realicen varios procedimientos en una base de datos central, incluyendo el acceso a algoritmos propios del manejador de la base de datos. Esto puede ser de gran ayuda para maximizar la base de datos para incrementar el desempeño y la velocidad de la misma.

Permite realizar agrupaciones de observaciones o de variables (cluster analysis) mediante tres algoritmos distintos, crear variables sintéticas a partir de variables colineales por medio del Análisis Factorial. Permite al usuario dar un formato especial a las salidas de los datos para su uso posterior, además de realizar revisiones lógicas de la información contenida en un fichero.sav. y obtener reportes de los valores considerados extraños. Es similar al uso de sintaxis o scripts para realizar revisiones de los ficheros.

Esta herramienta es fácil de utilizar debido a la interfaz visual que implementa, la cual hace que no sea necesario contar con habilidades de programación para utilizarla.

Clementine trabaja sobre sistemas operativos Windows y además es una herramienta que consume grandes recurso hardware.

1.3.3. Oracle Data Mining

Al estar integrado en la base de datos, Oracle Data Mining simplifica el proceso de extracción de conclusiones basadas en grandes cantidades de datos, ya que se elimina la necesidad de movimientos de datos para el proceso de análisis.

Oracle Data Mining contiene gran cantidad de algoritmos de minería de datos y de análisis de datos tales como predicción, clustering, clasificación, regresión, asociación, detección de anomalías, etc. Además provee medios para la creación, administración y desarrollo operacional de los modelos de minería de datos dentro del ambiente de la base de datos, lo que resulta en una mejora de la productividad, automatización e integración.

Oracle Data Mining acepta tanto tablas transaccionales como no transaccionales (resúmenes, registros únicos), además hace todas las transformaciones necesarias automáticamente de forma interna, liberando así de este trabajo a los usuarios o desarrolladores.

1.3.4. Orange

Orange es una herramienta de código abierto para minería de datos que contiene una colección de algoritmos programados en C++ interconectados con Python. Dicha herramienta cuenta con un entorno gráfico bastante cómodo y además cuenta con distribuciones para varios sistemas operativos, tales como Windows, Linux y Macintosh, lo que la hace multiplataforma.

Orange para el formato entrada/salida usa ficheros separados por tabulación e incluye otros como C4.5. Esta herramienta contiene técnicas de clasificación, evaluación automatizada, asociación y clustering.

1.4. Arquitectura de Software

La Arquitectura de Software (AS) tiene sus antecedentes en 1968, cuando Edsger Dijkstra, de la Universidad Tecnológica de Eindhoven en Holanda y Premio Turing 1972, propuso que se establezca una estructuración correcta de los sistemas de software antes de lanzarse a programar, escribiendo código de cualquier manera (4).

La historia de la AS no ha sido tan continua como la de la Ingeniería de Software, pero con el paso de los años y el aumento de la complejidad de las aplicaciones la AS fue tomando auge, pero no fue hasta 1992 que se define a la AS como disciplina de software en la publicación “Foundations for the study of software architecture” escrita por Dewayne Perry y Alexander Wolf donde aparece por primera vez la expresión “arquitectura de software” en el sentido en que hoy lo conocemos.

En la actualidad existen cientos de definiciones de AS y ejemplo de ello es la colección que se encuentra en el Software Engineering Institute (SEI), pero la realidad es que ninguna de ellas ha sido respaldada por la totalidad de los arquitectos, aunque todas tienen que ver con una de las tres ideas siguientes:

1. La Arquitectura de Software como un proceso dentro del ciclo de vida de un sistema.
2. La Arquitectura de Software como la Topología del Sistema. Es decir la forma de articular los diferentes estilos y componentes dentro de una solución.
3. La Arquitectura de Software como una disciplina profesional y académica.

Mary Shaw y David Garlan, sugieren que la arquitectura del software es un nivel del diseño que trata cuestiones“…más allá de las estructuras de los algoritmos y de datos del cómputo; diseñar y especificar la estructura del sistema total emerge como nueva clase de problema. Las ediciones estructurales incluyen la organización gruesa y la estructura global del control; los protocolos para la comunicación, la sincronización, y el acceso a los datos; asignación de la funcionalidad para diseñar elementos; distribución física; composición de los elementos del diseño; escalamiento y funcionamiento; y selección entre alternativas del diseño” (5).

Rational Unified Process, 1999, propone que: “Una arquitectura es el sistema de decisiones significativas sobre la organización de un sistema de software, la selección de los elementos estructurales y de sus

interfaces por los cuales el sistema es compuesto, junto con su comportamiento según lo especificado en las colaboraciones entre esos elementos, la composición de estos elementos estructurales y del comportamiento en subsistemas progresivamente más grandes, y el estilo arquitectónico que dirigen esta organización, los elementos y sus interfaces, sus colaboraciones y su composición" (6).

Phillipe Kruchten explica que: "La arquitectura de software tiene que ver con el diseño y la implementación de estructuras de software de alto nivel. Es el resultado de ensamblar un cierto número de elementos arquitectónicos de forma adecuada para satisfacer la mayor funcionalidad y requerimientos de desempeño de un sistema, así como los requerimientos no funcionales, como la confiabilidad, portabilidad y disponibilidad" (7)

Estas definiciones a pesar de ser diferentes coinciden que la AS es el diseño de más alto nivel de abstracción de la estructura de un sistema, programa o aplicación y tiene el encargo de definir los módulos y las responsabilidades que tendrán cada uno de esos módulos así como la interacción entre ellos.

La definición oficial de AS es la que aparece en el documento Std 1471-2000 de la IEEE y plantea que: "La arquitectura de software es la organización fundamental de un sistema formada por sus componentes, las relaciones entre ellos y el contexto en el que se implantarán y los principios que sustentan su diseño y evolución".

El objetivo primario de la AS es el de aportar elementos que ayuden en la toma de decisiones significativas y, al mismo tiempo, proporcionar conceptos y un lenguaje común que permitan la comunicación entre todos los que tienen parte en el proceso del desarrollo del software.
Estas decisiones significativas se toman sobre:

- La organización del sistema de software.
- La selección de elementos estructurales y sus interfaces a través de los cuales se constituye el sistema.
- Su comportamiento, según resultados de las colaboraciones entre esos elementos.
- El estilo arquitectónico que guía esta organización: los elementos estáticos y dinámicos; sus interfaces, sus colaboraciones y su composición

Es importante señalar que la AS no es:

- Una disciplina madura.
- Diseño de software.
- Vinculada a la metodología RUP o ninguna otra metodología.
- Relacionada con modelado Orientado a Objetos.

Además, se necesita una arquitectura para:

- Comprender el sistema.
- Organizar el desarrollo.
- Fomentar la reutilización.
- Hacer evolucionar el sistema.

1.5. Modalidades y tendencias de la Arquitectura de software

En la década de 1990 se establece definitivamente la AS como un dominio, todavía hoy separado de manera confusa de ese marco global que es la ingeniería y de esa práctica puntual que es el diseño.

En la actualidad existen seis tendencias de la Arquitectura de Software, que se han estructurado de esta forma debido a las manera en que los arquitectos trabajan y hacia qué criterios se perfilan más, evidentemente todos tienen su forma de hacer los productos, por lo que existen varios modelos que dividen los estudios de la arquitectura del software a nivel mundial, estos son:

- **Arquitectura como etapa de ingeniería y diseño orientada a objetos:**

Este modelo fue planteado por James Rumbaugh, Ivar Jacobson, Grady Booch, Craig Larman y otros, y está ligado estrechamente al mundo de UML y Rational.

En esta postura, se plantea básicamente que la arquitectura se restringe a las fases iniciales y preliminares del proceso y concierne a los niveles más elevados de abstracción, pero no está ligada al requerimiento que viene antes o a la composición del diseño que viene después.

En esta escuela, si bien se reconoce el valor primordial de la abstracción y del ocultamiento de información, estos conceptos tienen que ver más con el encapsulamiento en clases y objetos que con la visión de conjunto arquitectónica.

La definición de arquitectura que se promueve en esta corriente tiene que ver con aspectos formales a la hora del desarrollo. Una definición típica y demostrativa sería la de Grady Booch; para él, la AS es "la estructura lógica y física de un sistema, forjada por todas las decisiones estratégicas y tácticas que se aplican durante el desarrollo" (8)

- **Arquitectura estructural, basada en un modelo estático de estilos, ADLs y vistas:**

Los representantes de esta corriente son todos académicos, mayormente de la Universidad Carnegie Mellon en Pittsburgh: Mary Shaw, Paul Clements, David Garlan, Robert Allen, Gregory Abowd, John Ockerbloom. Constituye la corriente fundacional y clásica de la disciplina, además de ser también la visión de la AS dominante en la academia, y aunque es la que ha hecho el esfuerzo más importante por el reconocimiento de la AS como disciplina, sus categorías y herramientas son todavía mal conocidas en la práctica industrial.

En toda la corriente, el diseño arquitectónico no sólo es el de más alto nivel de abstracción; rara vez se encontrarán referencias a lenguajes de programación o piezas de código, y en general nadie habla de clases o de objetos. Mientras algunos participantes excluyen el modelo de datos de las incumbencias de la AS (Shaw, Garlan, etc), otros insisten en su relevancia (Medvidovic, Taylor).

- **Estructuralismo arquitectónico radical.**

Se trata de un desprendimiento de la corriente anterior, mayoritariamente europeo, que asume una actitud más confrontativa con el mundo UML.

En el seno de este movimiento hay al menos dos tendencias, una que excluye de plano la relevancia del modelado orientado a objetos para la AS y otra que trata de buscar soluciones a los problemas que tiene el UML para la representación de la arquitectura, porque este lenguaje no se considera un ADL, proponiendo nuevos estereotipos y otros elementos necesarios para ello. (9)

- **Arquitectura basada en patrones.**

Si bien reconoce la importancia de un modelo emanado históricamente del diseño orientado a objetos, esta corriente surgida hacia 1996 no se encuentra tan rígidamente vinculada a UML en el modelado, ni a CMM en la metodología.

Esta modalidad hace énfasis en los patrones de diseño planteados en algunas literaturas muy conocidas como los de GOF entre otros. Además aquí se plantea que el diseño consiste en definir algunos patrones que ya existen para el sistema y estilos arquitectónicos relacionados con estos patrones y por supuesto que sean compatibles, para así conformar la solución más adecuada para el software

- **Arquitectura procesual.**

Es otra de las modalidades que se definen de la arquitectura de software, esta fue realizada en el SEI y con la participación de personalidades como Rick Kazman, Len Bass, Paul Clements, entre otros. Ellos tratan de precisar cuáles son los métodos y técnicas que se relacionan con la práctica arquitectónica, desde el punto de vista de la justificación económica de la arquitectura de software, de qué forma se seleccionan los atributos de la calidad, como se evalúa que una arquitectura de software propuesta satisface o no los requerimientos funcionales y no funcionales, como se evalúa el costo tanto material como intelectual que puede adoptar determinado estilo arquitectónico y no otro para la realización del software.

1.6. Patrones

Los patrones surgen en la década de los 70 cuando Christopher Alexander desarrolló en diversos estudios de temas de análisis del sentido de los planos, las formas, la edificación y la construcción, para procurar un modelo constructivo y humano de arquitectura. Estos estudios quedaron plasmados en sus libros A Pattern Language y The Timeless Way of Building, en los que proponía el aprendizaje y uso de una serie de patrones para la construcción de edificios de una mayor calidad.

Pero no fue hasta el década de los 90 que los patrones pasaron al campo del software cuando Erich Gamma, Richard Helm, Ralph Johnson y John Vlissides, más conocidos como the Gang of Four (la Banda de los Cuatro, GoF) publican el libro "Design Patterns: Elements of Reusable Object-Oriented Software".

Los patrones de software facilitan la reutilización, capturando las estructuras estáticas y dinámicas de colaboración de soluciones exitosas a problemas que surgen al construir aplicaciones.

Podemos asumir que un patrón no es más que una solución probada, que se puede aplicar con éxito a un determinado tipo de problema, a la que se le dado un nombre y aparece repetidamente en un marco específico. Los patrones son el resultado de la experiencia acumulada en el desarrollo de software.

Los patrones están formados por cuatro elementos esenciales:

- **El nombre:**
  Describe un problema con una o dos palabras, sus soluciones y la consecuencia, por las que será conocido por todos los desarrolladores de software, permitiendo que todos usen el mismo lenguaje.
- **El problema:**
  Describe cuándo se puede aplicar el patrón. Explica el problema y su contexto.
- **La solución:**
  Describe los elementos que forman el diseño, sus relaciones, responsabilidades, y cooperaciones
- **Las consecuencias:**
  Este elemento describe los resultados e inconvenientes de aplicar el patrón.

Clasificación de los Patrones

- **Patrones de Diseño:**
  Son aquéllos que expresan esquemas para definir estructuras de diseño (o sus relaciones) con las que construir sistemas software.
- **Patrones de Arquitectura:**
  Son aquellos que expresan un esquema organizativo y estructural para sistemas de software.
- **Patrones de Análisis:**

Son un conjunto de clases y asociaciones que se aplican a un modelo de análisis. Los patrones de análisis provienen de varios dominios y cada uno de ellos se describe en forma textual y con una simple notación pre-UML.

- **Patrones de Proceso o de Organización:**
  Son aquellos que muestran las estructuras de organización o los procesos de administración; pueden estar conformados por roles y áreas de trabajo.
- **Patrones de Idioma:**
  Son específicos de un lenguaje de implementación que describen como implementar algunos aspectos de un problema mediante el uso del lenguaje de programación.

1.6.1. Patrones de Arquitectura

Los patrones de arquitectura son muchos y muy variados, resolviendo cada uno de ellos problemas frecuentes de organización y estructura de sistemas, pero tienen estrecha relación con cada uno de los estilos arquitectónicos, por lo que se recomienda el uso de los patrones que estén predeterminados en cada uno de los estilos que se seleccionen para la arquitectura.

Entre los libros de patrones de arquitectura más reconocidos y estudiados mundialmente se encuentra Pattern Oriented Software Architecure: A System of Patterns (POSA) de Frank Buschmann, que con el tiempo quedó como primer volumen de una saga en la que colaboraron otros autores, donde los patrones “expresan un esquema de organización estructural para los sistemas de software. Proporcionan un conjunto de subsistemas predefinidos, especifica sus responsabilidades e incluye reglas y lineamientos para organizar la relación entre ellos” (10). Otro de estos libros es Pattern of Enterprise Application Architecture (PEAA), en estos dos libros se hacen una categorización de los patrones de arquitectura, la cual se muestra a continuación:

En POSA se divide a los patrones en las siguientes categorías (ver Anexo 1 Distribución de los patrones de POSA):

- Estructura (From Mud to Structure)
- Sistemas Distribuidos (Distributed Systems)
- Sistemas Interactivos (Interactive Systems)

- Sistemas Adaptables (Adaptable Systems)

En PEAA se definen las siguientes categorías de patrones (ver Anexo 2 Distribución de los patrones de PEEA y Anexo 3 Clasificación de patrones de arquitectura de PEEA.):

- Patrones de Dominio Lógico (Domain Logic Patterns)
- Patrones de Mapeo Objeto / Relacional (Object / Relational Mapping Patterns)
- Patrones de Presentación Web (Web Presentation Patterns)
- Patrones de Distribución (Distribution Patterns)
- Patrones de Concurrencia fuera de Línea (Offline Concurrency Patterns)
- Patrones de Estado de Sesión (Session State Patterns)
- Patrones de Base (Base Patterns)

1.7. Vistas

El concepto de vista señala la posibilidad de que la arquitectura de software puede ser examinada desde diferentes perspectivas, de modo que cada una de estas arquitecturas parciales constituye una vista y la visión total del sistema se obtiene por la combinación de todas ellas.

Los primeros en sugerir un enfoque multi-vista fueron Perry y Wolf, que proponían que la arquitectura habría de tener una vista de flujos de datos, una de control y otra de uso de recursos. Sin embargo, la propuesta más utilizada es el Modelo 4+1 vistas de Philippe Kruchten, que debe su auge a su adaptación a UML y su difusión dentro del campo orientado a objeto. La siguiente figura muestra el modo en que el modelo fue modificado para obtener correspondencia más directa con UML.

Vista Lógica	$\longrightarrow$	Vista Lógica
Vista de Procesos	$\longrightarrow$	Vista de Procesos
Vista de Desarrollo	$\longrightarrow$	Vista de Implementación
Vista Física	$\longrightarrow$	Vista de Despliegue
Vista de Escenarios	$\longrightarrow$	Vista de Caso de Uso

Figura 1 Evolución de vistas del modelo 4+1 de Kruchten

- **Vista de casos de uso:**
  Esta vista representa la selección y descripción de los casos de usos arquitectónicamente significativos brindando una noción general del sistema.
- **Vista lógica:**
  Esta vista representa los elementos de diseño más importantes para la arquitectura del sistema. Este describe las clases más importantes, su organización en paquetes y subsistemas, y estos a su vez en capas.
- **Vista de proceso:**
  Esta vista suministra una base para la comprensión de la organización de los procesos de un sistema, ilustrados en el mapeo de las clases y subsistemas en procesos e hilos. Solo suele usarse cuando el sistema presenta procesos concurrentes o hilos.
- **Vista de despliegue:**
  Esta vista suministra una base para la comprensión de la distribución física de un sistema a través de nodos. Suele utilizarse cuando el sistema está distribuido.
- **Vista de implementación:**
  Esta vista describe la descomposición del software en capas y subsistemas de implementación. También provee una vista de la trazabilidad de los elementos de diseño de la vista lógica ahora para la implementación. Además contiene una enumeración de los subsistemas y las dependencias entre los subsistemas.

1.8. Estilos Arquitectónicos

El estilo arquitectónico constituye uno de los conceptos más importantes dentro de la AS, describe y proporciona las propiedades básicas de una arquitectura y es encargado de imponer los límites de su evolución.

En algunas bibliografía se puede encontrar que se utilice el termino de estilo arquitectónico para referirse a los patrones o viceversa, pero por lo generar se estudian de manera separada. Los dos términos se refieren a la estructura y organización del sistema y la relación entre los elementos del mismo, por lo que la diferencia se aprecia en el nivel de abstracción en que se aplican. Los estilos arquitectónicos están a un

nivel de abstracción más alto, mientras los patrones se encuentran dentro de ellos a un nivel más cercano al diseño.

Según Pressman: “Cada estilo describe una categoría del sistema que contiene: un conjunto de componentes por ejemplo, una base de datos, módulos computacionales que realizan una función requerida por el sistema; un conjunto de conectores que posibilitan la comunicación, la coordinación y la cooperación entre los componentes; restricciones que definen como se pueden integrar los componentes que forman el sistema; y modelos semánticos que permiten al diseñador entender las propiedades globales de un sistema para analizar las propiedades conocidas de sus partes constituyentes” (11).

Los estilos arquitectónicos no son más que arquitecturas comunes, marcos de referencias arquitectónicas o formas comunes que pueden ser aplicadas. Definen los posibles patrones que van a tener las aplicaciones y permiten evaluar arquitecturas alternativas con ventajas y desventajas conocidas ante diferentes conjuntos de requerimientos no funcionales. Los estilos arquitectónicos comprenden los componentes (elementos), conectores, configuraciones y restricciones de las aplicaciones, así como sus relaciones y comportamiento, representando un alto nivel de abstracción.

Algunos de los principales estilos arquitectónicos que se usan en la actualidad están divididos por Clases de Estilos las que agrupan según características comunes:

- **Estilos de Flujo de Datos:**
  Esta familia de estilos enfatiza la reutilización y la modificabilidad. Es ideal para sistemas que realizan transformaciones de datos dividiéndolos en pasos sucesivos. En ella nos encontramos con dos tipos de componentes, el primero son los que realizan las transformaciones a los datos cuando pasan por ellos y el otro es el que se encarga de enviar el flujo de los datos desde la salida de un componente transformador a la entrada de otro, es decir, realiza el rol de comunicador.
  - Tubería y filtros.
- **Estilos Centrados en Datos:**

Esta familia de estilos enfatiza la integrabilidad de los datos. Su uso es apropiado en sistemas que se centran en el acceso y actualización de datos en estructuras de almacenamiento que son compartidos por un número indefinido de componentes consumidores.

- Arquitecturas de Pizarra o Repositorio.

- **Estilos de Llamada y Retorno:**

  Esta familia de estilos enfatiza la modificabilidad y la escalabilidad. Son los estilos más generalizados en sistemas en gran escala. Miembros de la familia son las arquitecturas de programa principal y subrutina, los sistemas basados en llamadas a procedimientos remotos, los sistemas orientados a objeto y los sistemas jerárquicos en capas (12).

  - Modelo – Vista – Controlador (MVC)
  - Arquitectura en Capas
  - Arquitectura Orientada a Objetos
  - Arquitectura basada en Componentes

- **Estilos de Código Móvil:**

  Esta familia se caracteriza por su enorme portabilidad. Las arquitecturas que siguen este estilo tienen una parte del sistema que pertenece al propio entorno nativo de la máquina que lo incluye, la otra parte no. Ejemplos de la misma son los intérpretes, los sistemas basados en reglas y los procesadores de lenguaje de comando.

  - Arquitectura de Máquinas Virtuales.

- **Estilos Peer to Peer:**

  Esta familia, también llamada de componentes independientes, enfatiza la modificabilidad por medio de la separación de las diversas partes que intervienen en la computación. Consiste por lo general en procesos independientes o entidades que se comunican a través de mensajes. Cada entidad puede enviar mensajes a otras entidades, pero no controlarlas directamente. Los mensajes pueden ser enviados a componentes nominados o propalados mediante broadcast (12).

- Arquitectura basada en Eventos.
- Arquitecturas Orientas a Servicios (SOA).
- Arquitecturas basadas en Recursos.

- **Estilos Heterogéneos:**

  Es la familia de estilos más fuertemente referida en los últimos tiempos, se incluyen en este grupo formas compuestas o indóciles a la clasificación en las categorías habituales. Podrían agregarse formas que aparecen esporádicamente en los estudios de nuevos estilos, como los sistemas de control de procesos industriales, sistemas de transición de estados, arquitecturas específicas de dominios o estilos derivados de otros estilos.

  - Sistemas de control de procesos.
  - Arquitecturas Basadas en Atributos.

Los estilos más importantes a analizar para su posible utilización son:

1.8.1. Modelo – Vista – Controlador

Reconocido como estilo arquitectónico por Taylor y Medvidovic (13). Este estilo se utiliza principalmente cuando es necesario modularizar la interfaz de usuario, las reglas de negocio y el control de eventos.

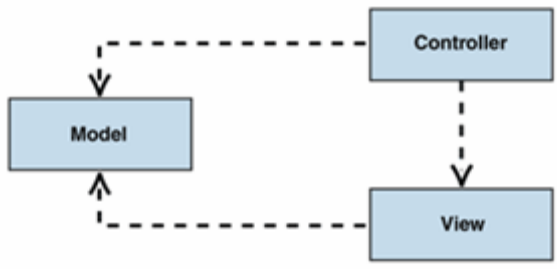


Figura 2 Estilo Modelo-Vista-Controlador

- **Modelo:**

  Administra el comportamiento y los datos del dominio de aplicación, responde a requerimientos de información sobre su estado (usualmente formulados desde la vista y responde a instrucciones de

cambiar el estado habitualmente desde el controlador). 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.

Ventajas:

- Permite el soporte de vistas múltiples.
- Permite la adaptación a los cambios, ya que al agregar nuevos tipos de vista no afecta el modelo.
- Evita poner el código indebido en la capa impropia. Facilita despliegue en caso de modificaciones en el modelo de datos.

Desventajas:

- Puede aumentar un poco la complejidad de la solución. Como está guiado por eventos puede ser algo más difícil de depurar.
- Si el modelo experimenta cambios frecuentes, por ejemplo, podría desbordar las vistas con una lluvia de requerimientos de actualización.

1.8.2. Arquitectura basada en Objetos

En la bibliografía referente a estilos arquitectónicos este estilo se puede encontrar con nombres como Arquitecturas Basadas en Objetos, Abstracción de Datos y Organización Orientada a Objetos y perteneciente a la familia arquitectónica Llamada y Retorno. Los componentes de este estilo son los objetos, o más bien instancias de los tipos de dato abstractos.

Según David Garlan y Mary Shaw, los objetos representan una clase de componentes que ellos llaman managers, debido a que son responsables de preservar la integridad de su propia representación.

En la Arquitectura Basada en Objeto sus componentes se basan en los principios de la programación orientada a objetos: encapsulamiento, herencia y polimorfismo. Son así mismo las unidades de modelado,

diseño e implementación, y los objetos y sus interacciones son el centro de las incumbencias en el diseño de la arquitectura y en la estructura de la aplicación. Además las clases interfaces están separadas de sus implementaciones. En general la distribución de objetos es transparente.

Ventajas:

- Se puede modificar la implementación de un objeto sin afectar a sus clientes.
- Es posible descomponer problemas en colecciones de agentes en interacción.
- Un objeto es ante todo una entidad reutilizable en el entorno de desarrollo.

Desventajas:

- Para poder interactuar con otro objeto a través de una invocación de procedimiento, se debe conocer su identidad.
- Dependencia en cascada.
- Si se cambia un objeto se deben modificar todos los objetos o métodos que lo invocan.

1.8.3. Arquitectura en Capas

Las arquitecturas en capas constituyen uno de los estilos más conocidos y utilizados en la actualidad. Este estilo proporciona una organización jerárquica de tal manera que cada capa brinda servicios a la capa inmediatamente superior y se sirve de las prestaciones que le brinda la inmediatamente inferior. Cada capa suele ser una entidad compleja, formada por un conjunto de paquetes o subsistemas.

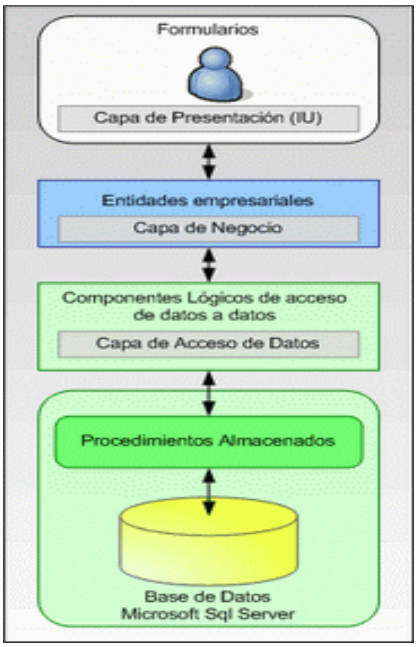


Figura 3 Estilo Capas

Ventajas:

- Soporta un diseño basado en niveles de abstracción crecientes, lo cual permite la partición de un problema complejo en una secuencia de pasos incrementales.
- Proporciona amplia reutilización, ya que se pueden utilizar diferentes implementaciones o versiones de una misma capa en la medida que soporten las mismas interfaces de cara a las capas adyacentes.
- Mejora soporte del sistema al admitir muy naturalmente optimizaciones y refinamientos.

Desventajas:

- Puede ser difícil definir que componentes ubicar en cada una de las capas.
- Los cambios en las capas inferiores tienden a filtrarse hacia las superiores.

1.8.4. Arquitectura Orientada a Servicios

La Arquitectura Orientada a Servicios (en inglés Service-oriented architecture o SOA), es un concepto de arquitectura de software que define la utilización de servicios para dar soporte a los requerimientos de software del usuario.

El concepto SOA hace referencia a un enfoque de arquitectura cuyo objetivo es la creación de sistemas a partir de servicios autónomos. Con SOA, la integración pasa a ser una reflexión previa más que una idea posterior. Probablemente, la solución final esté formada por servicios desarrollados en distintos lenguajes de programación y se aloje en plataformas diferentes con numerosos modelos de seguridad y procesos empresariales (14).

Según IBM este estilo es: “Una arquitectura de aplicación en la cual todas las funciones se definen como servicios independientes con interfaces invocables bien definidas, que pueden ser llamadas en secuencias definidas para formar procesos de negocios”

SOA permite la creación de sistemas altamente escalables que reflejan el negocio de la organización, a su vez brinda una forma estándar de exposición e invocación de servicios, lo cual facilita la interacción entre diferentes sistemas propios o de terceros.

1.9. Lenguajes de descripción arquitectónica (ADL)

Desde el surgimiento de la disciplina arquitectura de software a principios de los 90 y en lo que va del siglo XXI se han materializado diversas propuestas para describir y razonar en términos de arquitectura de software; muchas de ellas han asumido la forma de lenguajes de descripción arquitectónicos (ADL). Los ADL se utilizan para satisfacer requerimientos descriptivos de alto nivel de abstracción permitiendo descomponer un sistema en componentes y conectores, especificando de qué manera estos elementos se combinan para formar configuraciones y definiendo familias de arquitecturas o estilos.

Schneider define un ADL como una notación que permite una descripción y análisis preciso de las propiedades observables de una arquitectura de software, dando soporte a distintos estilos arquitectónicos a diferentes niveles de abstracción. (15)

La delimitación categórica de los ADL es problemática. Ocasionalmente, otras notaciones se utilizan como si fueran ADL para la descripción de arquitecturas, un ejemplo lo constituye el UML, lo ha provocado que los ADL no ocupen el lugar que deberían.

Según Shaw y Garlan todo ADL debe tener las siguientes propiedades (5):

- **Composición:**
  Permiten dividir un sistema complejo en partes más pequeñas, de manera jerárquica, o construir un sistema a partir de los elementos que lo constituyen.
- **Abstracción:**
  Permiten identificar los distintos elementos en una estructura de alto nivel, así como su papel en la misma.
- **Reutilización:**
  Permiten la reutilización tanto de los componentes y conectores como de la propia arquitectura.
- **Configuración:**
  Permiten separar con claridad la descripción de los elementos individuales y de las estructuras -elementos compuestos- en las que éstos participen.
- **Heterogeneidad:**
  El ADL ha de ser independiente del lenguaje en que se implemente cada uno de los componentes que manipula; y debe diseñarse de modo que pueda combinar patrones arquitectónicos diferentes en un único sistema complejo.
- **Flexibilidad:**
  Permiten la definición de nuevas formas de interacción entre componentes.
- **Análisis:**
  Permiten diversas formas de análisis de la arquitectura, de modo que se puedan determinar sus propiedades con independencia de una implementación concreta, así como verificarlas después de cualquier modificación.

Entre los principales ADL que se usan en la actualidad se encuentran:

- **Acme:**
  Se define como una herramienta capaz de soportar el mapeo de especificaciones arquitectónicas entre diferentes ADL, o en otras palabras, como un lenguaje de intercambio de arquitectura. No es entonces un ADL en sentido estricto, aunque la literatura de referencia acostumbra tratarlo como tal. De hecho, posee numerosas prestaciones que también son propias de los ADL (16) y los desarrollos actuales profundizan su capacidad intrínseca como ADL puro.

- **Armani:**

  Es un lenguaje puramente declarativo que describe la estructura del sistema y las restricciones a respetar, pero no hace referencia alguna a la generación del sistema o a la verificación de sus propiedades no funcionales. Se basa en siete entidades para describir las instancias del diseño: componentes, conectores, puertos, roles, sistemas, representaciones y propiedades (16).

- **Aesop:**

  El nombre oficial es Aesop Software Architecture Design Environment Generator y se centra en la especificación propiamente dicha de arquitecturas. Basado en componentes, con un sistema de tipos extensible donde los puntos de interfaz se modelan como puertos de entrada y salida. La evolución la permite mediante un subtipado que preserva el comportamiento. Respecto a las propiedades no funcionales, solo permite que se asocien textos arbitrarios a los componentes.

- **Jacal:**

  Es un lenguaje de descripción de arquitecturas de software de propósito general. Reúne ventajas de otros lenguajes existentes y tiene la virtud de permitir la ejecución (animación) de las arquitecturas descritas. Mediante este procedimiento se puede comprobar o refutar propiedades deseables de los diseños y recopilar métricas dinámicas. Además cuenta con una representación gráfica que permita a simple vista transmitir la arquitectura del sistema, sin necesidad de recurrir a información adicional.

- **Darwin:**

  Es un lenguaje de descripción arquitectónica desarrollado por Jeff Magee y Jeff Kramer. El mismo describe un tipo de componente mediante una interfaz consistente en una colección de servicios que son ya sea provistos (declarados por ese componente) o requeridos (o sea, que se espera ocurran en el entorno) (16). Darwin no proporciona una base adecuada para el análisis de la conducta de una arquitectura, debido a que el modelo no dispone de ningún medio para describir las propiedades de un componente o de sus servicios más que como comentario (17).

1.10. Marcos de Trabajo (Frameworks)

Dentro del desarrollo de software un marco de trabajo o framework, por su término en inglés, no es más que una estructura de soporte definida por la cual un proyecto de software puede ser organizado y

desarrollado. Define una arquitectura genérica, compuesta por una serie de componentes personalizables y sus interfaces, así como las reglas y mecanismos de integración entre ellos.

Los framework están basados en la idea de permitir la producción fácil de sistemas de software pertenecientes a un dominio específico utilizando una estructura genérica.

Los objetivos principales que persigue un framework son:

- El desarrollo rápido de aplicaciones.
- La reutilización de componentes software.
- Promover buenas prácticas de desarrollo como el uso de patrones.

1.11. Lenguaje Unificado de Modelado (UML)

Lenguaje Unificado de Modelado (UML, por sus siglas en inglés, Unified Modeling Language) es el lenguaje de modelado de sistemas de software más conocido y utilizado en la actualidad. Es un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema software orientado a objetos.

Un modelo UML está compuesto por tres clases de bloques de construcción (ver Anexo 4 Componentes UML):

- **Elementos:**
  Son abstracciones de cosas reales o ficticias (estructurales, comportamiento, agrupación, anotación).
- **Relaciones:**
  Relaciones entre los elementos (dependencia, asociación, generalización).
- **Diagramas:**
  Son los objetos con sus relaciones (clases, objetos, casos de uso, secuencia, colaboración, estados, actividades, componentes, despliegue).

Un sistema de software es representado por este lenguaje a través de cinco vistas o modelos principales separados, pero relacionados entre sí, denominadas vista de caso de uso, lógica, de implementación, de

procesos y de despliegue. Cada uno de estas vistas utiliza diversos diagramas para describir el sistema, los cuales se dividen en estáticos y dinámicos.

UML no es considerado un lenguaje de descripción arquitectónica ya que carece del concepto de estilo y su forma de expresar ciertas características, sobre todo las dinámicas no son suficientes para los arquitectos. A pesar de esto, UML es el lenguaje más usado en el modelado de sistemas de software, el cual describe a la AS mediante las cinco vistas mencionadas anteriormente y cuenta con algunos conceptos utilizados por esta como interfaz, componentes, etc.

1.12. Metodologías del Desarrollo de Software

Cuando se va a comenzar con el desarrollo de un software algo fundamental es tener seleccionada la Metodología de Desarrollo que se va a utilizar. Lo más recomendable es realizar un estudio de las metodologías más utilizadas dentro del proceso de desarrollo de software, para así tener un conocimiento base y realizar una buena selección.

1.12.1. Extreme Programing (XP)

Es una de las metodologías de desarrollo de software más exitosas y utilizadas en la actualidad para el desarrollo de proyectos de corto plazo y corto equipo. La metodología consiste en una programación rápida o extrema, cuya particularidad es tener como parte del equipo, al usuario final, pues es uno de los requisitos para llegar al éxito del proyecto.

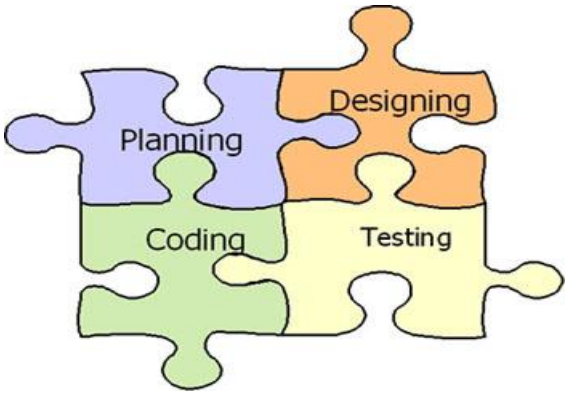


Figura 4 Metodología Extreme Programing

Esta metodología se basa en:

- **Pruebas Unitarias:** se basa en las pruebas realizadas a los principales procesos, de tal manera que adelantándonos en algo hacia el futuro, podamos hacer pruebas de las fallas que pudieran ocurrir. Es como si nos adelantáramos a obtener los posibles errores.

- **Refabricación:** se basa en la reutilización de código, para lo cual se crean patrones o modelos estándares, siendo más flexible al cambio.

- **Programación en pares:** una particularidad de esta metodología es que propone la programación en pares, la cual consiste en que dos desarrolladores participen en un proyecto en una misma estación de trabajo. Cada miembro lleva a cabo la acción que el otro no está haciendo en ese momento.

Lo fundamental en este tipo de metodología es:

- La comunicación, entre los usuarios y los desarrolladores.
- La simplicidad, al desarrollar y codificar los módulos del sistema.
- La retroalimentación, concreta y frecuente del equipo de desarrollo, el cliente y los usuarios finales.

1.12.2. Microsoft Solution Framework (MSF)

El MSF es una metodología flexible e interrelacionada con una serie de conceptos, modelos y prácticas de uso, que controlan la planificación, el desarrollo y la gestión de proyectos. MSF se centra en los modelos de proceso y de equipo dejando en un segundo plano las elecciones tecnológicas.

Figura 5 Metodología MSF

MSF tiene las siguientes características:

- **Adaptable:** es parecido a un compás, usado en cualquier parte como un mapa, del cual su uso es limitado a un específico lugar.
- **Escalable:** puede organizar equipos tan pequeños entre 3 o 4 personas, así como también, proyectos que requieren 50 personas a más.
- **Flexible:** es utilizada en el ambiente de desarrollo de cualquier cliente.
- **Tecnología Agnóstica:** porque puede ser usada para desarrollar soluciones basadas sobre cualquier tecnología.

MSF se compone de varios modelos encargados de planificar las diferentes partes implicadas en el desarrollo de un proyecto: Modelo de Arquitectura del Proyecto, Modelo de Equipo, Modelo de Proceso, Modelo de Gestión del Riesgo, Modelo de Diseño de Proceso y por último el Modelo de Aplicación.

- **Modelo de Arquitectura del Proyecto:** Diseñado para acortar la planificación del ciclo de vida. Este modelo define las pautas para construir proyectos empresariales a través del lanzamiento de versiones.
- **Modelo de Equipo:** Este modelo ha sido diseñado para mejorar el rendimiento del equipo de desarrollo. Proporciona una estructura flexible para organizar los equipos de un proyecto. Puede ser escalado dependiendo del tamaño del proyecto y del equipo de personas disponibles.
- **Modelo de Proceso:** Diseñado para mejorar el control del proyecto, minimizando el riesgo, y aumentar la calidad acortando el tiempo de entrega. Proporciona una estructura de pautas a seguir en el ciclo de vida del proyecto, describiendo las fases, las actividades, la liberación de versiones y explicando su relación con el Modelo de equipo.
- **Modelo de Gestión del Riesgo:** Diseñado para ayudar al equipo a identificar las prioridades, tomar las decisiones estratégicas correctas y controlar las emergencias que puedan surgir. Este modelo proporciona un entorno estructurado para la toma de decisiones y acciones valorando los riesgos que puedan provocar.
- **Modelo de Diseño del Proceso:** Diseñado para distinguir entre los objetivos empresariales y las necesidades del usuario. Proporciona un modelo centrado en el usuario para obtener un diseño eficiente y flexible a través de un enfoque iterativo. Las fases de diseño conceptual, lógico y físico

proveen tres perspectivas diferentes para los tres tipos de roles: los usuarios, el equipo y los desarrolladores.

- **Modelo de Aplicación:** Diseñado para mejorar el desarrollo, el mantenimiento y el soporte, proporciona un modelo de tres niveles para diseñar y desarrollar aplicaciones software. Los servicios utilizados en este modelo son escalables, y pueden ser usados en un solo ordenador o incluso en varios servidores.

1.12.3. Rational Unified Process (RUP)

La metodología RUP, llamada así por sus siglas en inglés Rational Unified Process, es un proceso de desarrollo de software y junto con el Lenguaje Unificado de Modelado UML, constituye la metodología estándar más utilizada para el análisis, implementación y documentación de sistemas orientados a objetos.

El RUP no es un sistema con pasos firmemente establecidos, sino un conjunto de metodologías adaptables al contexto y necesidades de cada organización.

RUP se caracteriza por ser centrado en la arquitectura, ya que la organización del sistema depende de los casos de usos claves y debe tener en cuenta la comprensibilidad, la facilidad de adaptación al cambio y la reutilización. Los casos de uso claves son aquellos que más le interesan al cliente. Es un proceso iterativo e incremental, el trabajo se divide en partes llamadas iteraciones, en cada una de ellas se recorren todos los flujos que RUP propone (Modelamiento del negocio, Requerimiento, Análisis y Diseño, Implementación, etc.). Y por último RUP es dirigido por casos de uso, porque son estos los que especifican las funcionalidades con las que va a contar el sistema. Los casos de uso no sólo son una herramienta para especificar los requisitos del sistema, también guían su diseño, implementación y pruebas, es decir, guían todo el desarrollo software.

RUP divide en 4 fases el desarrollo del software:

- **Inicio:** El Objetivo en esta etapa es determinar la visión del proyecto.
- **Elaboración:** En esta etapa el objetivo es determinar la arquitectura óptima.
- **Construcción:** En esta etapa el objetivo es llegar a obtener la capacidad operacional inicial.

- **Transición:** El objetivo es llegar a obtener el release del proyecto.

El ciclo de vida que se desarrolla por cada iteración, es llevada bajo dos disciplinas:

Disciplinas de Desarrollo

- **Modelamiento del Negocios:** Entendiendo las necesidades del negocio. Identifica quienes participan en el negocio y las actividades que requieren automatización.
- **Requerimientos:** Trasladando las necesidades del negocio a un sistema automatizado. Se definen las funcionalidades requeridas y las restricciones.
- **Análisis y Diseño:** Trasladando los requerimientos dentro de la arquitectura de software.
- **Implementación:** Creando software que se ajuste a la arquitectura y que tenga el comportamiento deseado.
- **Pruebas:** Asegurándose que el comportamiento requerido es el correcto y que todo lo solicitado está presente. Busca defectos lo largo del ciclo de vida del software.
- **Despliegue:** Produce release del producto y realiza actividades (empaque, instalación, asistencia a usuarios, etc.) para entregar el software a los usuarios finales.

Disciplina de Soporte

- **Gestión de configuración y cambio:** Guardando todas las versiones del proyecto. El control ayuda a evitar confusiones costosas.
- **Gestión de proyecto:** Proporcionar directrices prácticas para la planificación, selección de personal, ejecución y supervisión de los proyectos. Proporcionar una infraestructura para gestionar los riesgos.
- **Ambiente:** La disciplina de entorno proporciona el entorno de soporte para un proyecto. Proporciona a la empresa de desarrollo de software un entorno de desarrollo de software (los procesos y las herramientas) que den soporte al equipo de desarrollo.

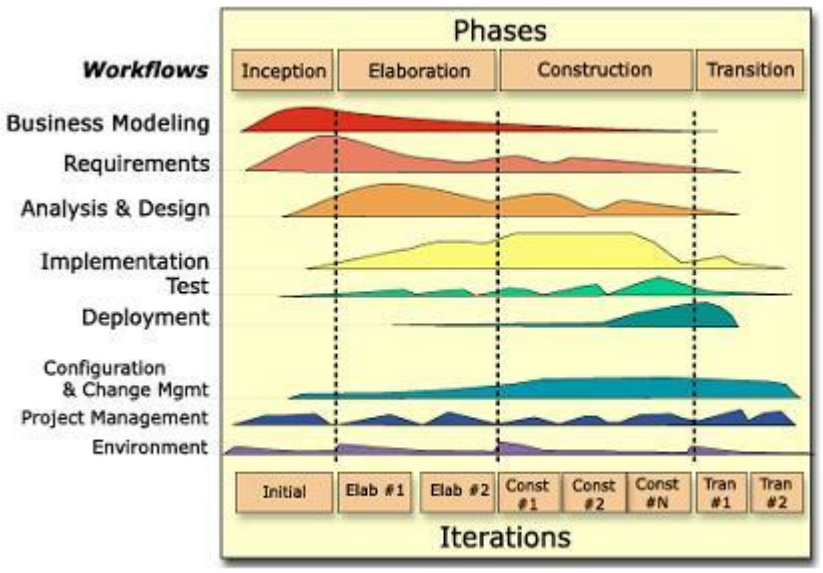


Figura 6 Fases e Iteraciones de la Metodología RUP

Los elementos del RUP son:

- **Actividades**, Son los procesos que se llegan a determinar en cada iteración.
- **Trabajadores**, Vienen hacer las personas o entes involucrados en cada proceso.
- **Artefactos**, Un artefacto puede ser un documento, un modelo, o un elemento de modelo.

Una particularidad de esta metodología es que, en cada ciclo de iteración, se hace exigente el uso de artefactos, siendo por este motivo, una de las metodologías más importantes para alcanzar un grado de certificación en el desarrollo del software.

1.13. Arquitecto de software

El arquitecto de software es considerado hoy en día como un integrante fundamental en los proyectos de desarrollo de software. La IEEE define al arquitecto de software de esta manera: “es una persona, equipo u organización responsable por la arquitectura del sistema” (18).

El arquitecto de software es el que toma las principales decisiones acerca de cómo será construido el software por los programadores, por lo que debe dominar la mayor cantidad de tecnologías de software y

prácticas de diseño para lograr garantizar el rehúso, robustez, portabilidad, flexibilidad, escalabilidad y mantenibilidad de las aplicaciones. Las decisiones que el arquitecto toma deberán ser plasmadas en una notación formal estandarizada como lo es UML, sobre todo si se utilizan las nuevas tecnologías, en especial con los lenguajes orientados a objetos.

El arquitecto de software debe contar con una serie de características y capacidades, las que RUP plantea de esta forma:

El arquitecto de software debe disponer de formación, madurez, visión y una amplia experiencia que permita recuperar cuestiones rápidamente y realizar valoraciones educadas y críticas en ausencia de la información completa. Más concretamente, el arquitecto de software, o los miembros del equipo de arquitectura, deben combinar estas habilidades (19):

- Experiencia en el dominio del problema, a través de una comprensión precisa de los requisitos, y el dominio de ingeniería de software. Si hay un equipo, estas cualidades pueden abarcar los diferentes miembros del equipo, pero como mínimo un arquitecto de software debe proporcionar la visión global para el proyecto. (19)
- Liderazgo para dirigir el esfuerzo técnico entre los diferentes equipos, y para tomar decisiones críticas bajo presión y adherirse a estas decisiones. Para ser efectivos, el arquitecto de software y el gestor de proyectos deben trabajar conjuntamente, con el arquitecto de software dirigiendo las cuestiones técnicas y el gestor de proyectos dirigiendo las cuestiones administrativas. El arquitecto de software debe tener autoridad para tomar decisiones técnicas. (19)
- Comunicación para ganar confianza, persuadir, motivar y orientar. El arquitecto de software no puede dirigir por decreto, sólo con el consentimiento del resto del proyecto. Para ser efectivos, el arquitecto de software debe ganar el respeto del equipo de proyecto, el gestor del proyecto, el cliente y la comunidad de usuarios, así como del equipo de gestión. (19)
- Orientación a objetivos y proactividad centrándose exclusivamente en los resultados. El arquitecto de software es la fuerza técnica dirigente detrás del proyecto, no un visionario o un soñador. La carrera de un arquitecto de software con éxito es una larga serie de decisiones sub-óptimas efectuadas en momentos de incertidumbre y bajo presión. Sólo quienes puedan centrarse en lo que se debe hacer tendrán éxito en este entorno del proyecto. (19)

1.14. Conclusiones

A partir de lo estudiado en este capítulo se arribó a las siguientes conclusiones:

- Se estudiaron las herramientas de minería de datos más potentes y se entendió el funcionamiento de las mismas, así como las principales funcionalidades que brindan.
- Del estudio de los diferentes conceptos relacionados con la arquitectura, tales como Vistas, Frameworks, Estilos, etc., se logró un mejor entendimiento de los mismos, así como definir los que se van a utilizar en la arquitectura que se propone en el documento (se exponen en el capítulo 2).
- Se definió UML como lenguaje de modelado, RUP como metodología a utilizar y los principales artefactos y responsabilidades que debe generar y poseer un arquitecto de software según esta metodología.

Capítulo 2

Diseño arquitectónico

2.1. Introducción

En este capítulo se hace una propuesta de solución al problema planteado en el diseño teórico de la investigación basada en los puntos fundamentales del Documento Descripción de la Arquitectura que es el artefacto fundamental que define y construye el arquitecto a lo largo del ciclo de vida del software.

2.2. Ambiente de desarrollo

2.2.1. Metodología a utilizar

Lo más importante antes de elegir la metodología a utilizar en el proceso de desarrollo es determinar el alcance que tendrá el software, para después poder seleccionar la metodología que más se ajuste.

El proyecto a desarrollar es amplio y tiene un alcance en correspondencia con él, el equipo de desarrollo está compuesto por gran cantidad de personas además se necesita de una documentación detallada, lo que fue solicitado por el cliente, por lo que se decidió utilizar como metodología para el proceso de desarrollo de software Rational Unified Process (RUP), el cual ha sido brevemente explicado en el epígrafe 1.13.3 del capítulo anterior.

2.2.2. Herramientas de modelado

Como herramienta de modelado se seleccionó el Visual Paradigm por las facilidades de uso que brinda, las cuales se exponen a continuación.

Visual Paradigm

Es una herramienta CASE que utiliza UML como lenguaje de modelado. Es muy completa y fácil de usar, con soporte multiplataforma y que proporciona excelentes facilidades de interoperabilidad con otras aplicaciones.

Visual Paradigm es una herramienta profesional que soporta el ciclo de vida completo del desarrollo de software: análisis y diseño orientados a objetos, construcción, pruebas y despliegue. Tiene la capacidad de crear el esquema de clases a partir de una base de datos y crear la definición de base de datos a partir del esquema de clases. Permite invertir código fuente de programas, archivos ejecutables y binarios en modelos UML al instante, creando de manera simple toda la documentación.

Visual Paradigm para UML soporta una serie de lenguajes para la ingeniería inversa tales como: Java, C++, PHP, Ada y Python.

Está disponible en varias ediciones, cada una destinada a unas necesidades: Enterprise, Professional, Community, Standard, Modeler y Personal.

2.2.3. Entornos de desarrollo integrados (IDE)

Es un programa compuesto por una serie de herramientas para un programador, es decir, consiste en un editor de código, un compilador, un depurador y un constructor de interfaz gráfica GUI.

Puede dedicarse en exclusiva a un solo lenguaje de programación, o bien puede utilizarse para varios.

Zend Studio for Eclipse 6.0:

Zend Studio para Eclipse combina la tecnología probada de Zend y las herramientas de desarrollo de Eclipse para PHP como proyección de crear el IDE más poderoso del mundo para el desarrollo de aplicaciones Web (20).

Soporta varios lenguajes de programación, aunque fue construido especialmente para PHP.

En concreto, Zend Studio para Eclipse aporta:

- Las ventajas de Eclipse.
- Una comunidad de millones de usuarios y miles de desarrolladores.
- Cientos de plugins.
- Soporte multi-lenguaje en una única herramienta (Eclipse).
- Coloreado de sintaxis PHP, autocompletado de código e inspección de métodos y atributos.
- Soporte básico de depuración de scripts PHP.

Eclipse:

Eclipse es principalmente una plataforma de programación de código abierto y multiplataforma, usada para crear entornos integrados de desarrollo.

Eclipse fue desarrollado originalmente por IBM como el sucesor de su familia de herramientas para VisualAge. Eclipse es ahora desarrollado por la Fundación Eclipse, una organización independiente sin ánimo de lucro que fomenta una comunidad de código abierto.

Eclipse emplea módulos (en inglés plug-in) para proporcionar toda su funcionalidad, a diferencia de otros entornos monolíticos donde las funcionalidades están todas incluidas, las necesite el usuario o no. Esta característica le permite extenderse usando otros lenguajes de programación como son C/C++ y Python.

La versión actual de Eclipse dispone de las siguientes características:

- Editor de texto
- Resaltado de sintaxis
- Compilación en tiempo real
- Pruebas unitarias con JUnit
- Control de versiones con CVS
- Integración con Ant
- Asistentes (wizards): para creación de proyectos, clases, tests, etc.
- Refactorización

Eclipse fue liberado originalmente bajo la Common Public License, pero después fue re-licenciado bajo la Eclipse Public License. La Free Software Foundation ha dicho que ambas licencias son licencias de software libre. (21).

- Pydev

Pydev es un módulo que permite utilizar Eclipse para el desarrollo con Python y Jython, convirtiendo a eclipse en un IDE para Python de primera clase.

Las principales características con que cuenta son:

- resaltado de sintaxis
- explorador de clases
- resaltado de errores
- completamiento de código.

2.2.4. Lenguajes de programación

Los lenguajes de programación son herramientas que nos permiten crear programas y software. Entre ellos tenemos Python, C, PHP, Java, etc.

Python:

Python es un lenguaje de scripting independiente de plataforma y orientado a objetos, preparado para realizar cualquier tipo de programa, desde aplicaciones Windows a servidores de red o incluso, páginas web. Es un lenguaje interpretado, lo que significa que no se necesita compilar el código fuente para poder ejecutarlo, lo que ofrece ventajas como la rapidez de desarrollo.

- **¿Por qué Python?**

Es un lenguaje multiplataforma, aunque originalmente se desarrolló para Unix, cualquier sistema es compatible con el lenguaje siempre y cuando exista un intérprete programado para él. Ejemplo: Windows, Mac, etc.

Este lenguaje contiene una gran cantidad de librerías, así como tipos de datos y funciones incorporadas que ayudan a realizar muchas tareas habituales sin necesidad de tener que programarlas desde cero.

La sencillez y velocidad con la que se crean los programas. Un programa en Python puede tener de 3 a 5 líneas de código menos que su equivalente en Java o C.

Además, Python es gratuito, incluso para propósitos empresariales.

PHP[1]:

PHP es un lenguaje interpretado de alto nivel embebido en páginas HTML y ejecutado en el servidor. Es un lenguaje de propósito general, aunque fue concebido, en un principio, para la creación de páginas web dinámicas.

- **¿Por qué PHP?**

PHP funciona tanto en sistemas Unix o Linux con servidor web Apache como en sistemas Windows con Microsoft Internet Information Server, de forma que el código generado por cualquiera de estas plataformas no debe ser modificado al pasar a la otra.

PHP tiene soporte para la programación orientada a objetos, es decir, es posible crear clases para la construcción de objetos, con sus constructores, etc. Además soporta herencia, aunque no múltiple.

Tiene como principal característica su amplio soporte para una gran cantidad de bases de datos. Tiene acceso a un gran número de gestores de bases de datos: Adabas D, dBase, Empress, Ingress, InterBase, FrontBase, DB2, Informix, mSQL, MySQL, ODBC, Oracle, PostgreSQL, Sybase, etc.

Es un lenguaje basado en herramientas con licencia de software libre, es decir, no hay que pagar licencias, ni está limitada su distribución y, es posible ampliarlo con nuevas funcionalidades.

JavaScript:

[1] acrónimo recursivo que significa **P**HP **H**ypertext **P**re-processor

JavaScript es el lenguaje que permite interactuar con el navegador de manera dinámica y eficaz, proporcionando a las páginas web dinamismo y vida.

Es un lenguaje de programación interpretado, es decir, que no requiere compilación, en otras palabras, los programas escritos con JavaScript se pueden probar directamente en cualquier navegador sin necesidad de procesos intermedios, con una sintaxis semejante a la del lenguaje Java y al lenguaje C.

A pesar de su nombre, JavaScript no guarda ninguna relación directa con el lenguaje de programación Java. Legalmente, JavaScript es una marca registrada de la empresa Sun Microsystems.

Los programas JavaScript se puede incluir en cualquier documento HTML, o todo aquel que termine traduciéndose en HTML en el navegador del cliente, ya sea PHP, ASP, JSP, SVG, y se encargan de realizar acciones en el cliente, como pueden ser pedir datos, confirmaciones, mostrar mensajes, crear animaciones, comprobar campos.

CSS:

Las CSS (Cascading Style Sheets- Hojas de Estilo en Cascada) son un lenguaje formal utilizado para definir la presentación de un documento estructurado escrito en HTML o XML. Es un mecanismo simple que describe cómo se va a mostrar un documento en la pantalla, o cómo se va a imprimir, o incluso cómo va a ser pronunciada la información presente en ese documento a través de un dispositivo de lectura.

CSS permite separar la estructura de un documento de su presentación y a su vez a los desarrolladores Web controlar el estilo y el formato de múltiples páginas Web al mismo tiempo.

2.2.5. Sistema Gestor de Bases de Datos

Es un conjunto de programas que permiten crear y mantener una Base de datos, asegurando su integridad, confidencialidad y seguridad.

PostgreSQL:

PostgreSQL es un producto open source y disponible sin costo. Postgres, desarrollado originalmente en el Departamento de Ciencias de Computación de UC Berkeley, fue pionero en muchos de los conceptos de objetos y relaciones que ahora están apareciendo en algunas bases de datos comerciales.

Como muchos otros proyectos open source, el desarrollo de PostgreSQL no es manejado por una sola compañía sino que es dirigido por una comunidad de desarrolladores y organizaciones comerciales las cuales trabajan en su desarrollo. Dicha comunidad es denominada el PGDG (PostgreSQL Global Development Group).

PostgreSQL puede ser ampliado por los propios usuarios en varias maneras, por ejemplo agregando nuevos:

- Tipos de datos
- Funciones
- Operadores
- Lenguajes de procedimiento

Y gracias a su licencia libre, PostgreSQL puede ser usado, modificado y distribuido por cualquier persona gratuitamente para cualquier propósito, ya sea privativo, académico o comercial.

2.2.6. Servidor de Aplicaciones.

Apache:

El servidor Web Apache es considerado la plataforma de servidores Web de código abierto más potente del mundo para plataformas GNU/Linux, Macintosh, Windows, entre otros sistemas operativos. Es desarrollado dentro del proyecto HTTP Server (httpd) de la Apache Software Foundation.

La arquitectura del servidor Apache es muy modular. El servidor consta con diversos módulos que aportan mucha de la funcionalidad que podría considerarse básica para un servidor web. Algunos de estos módulos son:

- mod_ssl: Comunicaciones Seguras vía TLS.
- mod_rewrite: reescritura de direcciones (se utiliza generalmente para transformar paginas dinámicas, como php en páginas estáticas HTML para así engañar a los navegantes o a los motores de búsqueda en cuanto a cómo fueron desarrolladas estas páginas).
- mod_proxy_ajp - Conector para enlazar con el servidor Jakarta Tomcat de páginas dinámicas en Java (servlets y JSP).

El servidor base puede ser extendido mediante la inclusión de módulos externos entre los que tenemos:

- mod_perl - Páginas dinámicas en Perl.
- mod_php - Páginas dinámicas en PHP.
- mod_python - Páginas dinámicas en Python.
- mod_ruby - Páginas dinámicas en Ruby.
- mod_mono - Páginas dinámicas en Mono
- mod_security - Filtrado a nivel de aplicación, para seguridad.

2.2.7. Framework o Componentes

Symfony:

Es un frameworks desarrollado completamente con PHP5 que ha logrado un gran éxito entre los desarrolladores. La primera versión de Symfony fue publicada en Octubre de 2005 por Fabien Potencier fundador del proyecto y presidente de Sensio, una empresa francesa de desarrollo de aplicaciones web conocida por sus innovaciones en este campo.

Symfony separa la lógica de negocio, la lógica de servidor y la presentación de la aplicación web. Proporciona varias herramientas y clases encaminadas a reducir el tiempo de desarrollo de una aplicación web compleja. Además, automatiza las tareas más comunes, permitiendo al desarrollador dedicarse por completo a los aspectos específicos de cada aplicación.

Principales Características de Symfony:

- Symfony viene con tareas de testing por línea de comandos y genera una clase vacía para ello al crear un controlador.

- Es compatible con la mayoría de gestores de bases de datos, como MySQL, PostgreSQL, Oracle y SQL Server de Microsoft, y además es independiente de los mismos (gestores de base de datos).
- Se puede ejecutar tanto en plataformas *nix (Unix, Linux, etc.) como en plataformas Windows
- Código fácil de leer que incluye comentarios de phpDocumentor y que permite un mantenimiento muy sencillo.
- Fácil de extender, lo que permite su integración con librerías desarrolladas por terceros.
- Sigue la mayoría de mejores prácticas y patrones de diseño para la web.

➢ **Propel:**

Propel es un ORM[2] para PHP utilizado por Symfony, que facilita la labor de desarrollo de aplicaciones web, gracias a la capa que transforma el tratamiento de la BD mediante objetos con la que se puede recuperar, insertar y modificar datos.

Este componente externo ha sido desarrollado por el equipo de Propel, y está completamente integrado en Symfony, por lo que se pueden considerar una parte más del framework.

Con Propel no es necesario preocuparse por las conexiones de la BD ni de escribir código SQL. Tan solo es necesario definir la base de datos en formato XML u obtener la definición desde una base de datos ya existente.

ExtJS:

ExtJS empezó siendo un conjunto de librerías y extensiones de JavaScript para el desarrollo de aplicaciones web interactivas usando tecnologías como AJAX, DHTML y DOM. Con el tiempo se convirtió en un Framework independiente y a principios de 2007 se creó una compañía para comercializar y dar soporte a este Framework.

ExtJS permite la implementación de interfaces visuales muy potentes, ofreciendo componentes para la implementación de tablas, árboles, formularios, contenedores, etc

Extjs soporta:

[2] Mapeo de objetos a bases de datos

- Internet Explorer 6+
- FireFox 1.5+
- Safari 3+
- Opera 9+

Algunas de las cualidades de Extjs son:

- Alto rendimiento
- Interfaces personalizables
- Muy buena arquitectura que permite extender los componentes gracias a su buen modelado.
- API intuitiva
- Licencia comercial y open source

Características:

- Ext.Element: Representa un elemento del árbol DOM. Muchas de las funciones de manipulación de los elementos tienen un parámetro opcional que permite realizar el cambio mediante un efecto de animación. El parámetro de animación puede ser un dato boleano o un objeto que incluye las opciones de la animación.
- Ext.BorderLayout: Esta clase representa un diseño común para ser usado en aplicaciones de escritorio.
- Ext.DomHelper: Utilidades para trabajar plantillas o DOM. Suporta el uso de DOM o fragmentos de HTML de forma transparente.
- Ext.TabPanel: Un ligero contenedor de tabs.
- Ext.UpdateManager: Proporciona soporte para actualización AJAX de los objetos Element.

Open-jACOB Draw2D:

Es una librería Javascript que utiliza AJAX para crear gráficos y diagramas. Su interfaz permite dibujarlos directamente desde el navegador, sin necesidad de instalar ningún software adicional.

Open-jACOB Draw2D es el componente de dibujo de Open-jACOB, un editor de Workflow al puro estilo de Visio con la finalidad de probar que realmente puede hacerse totalmente online y sin necesidad de instalar plugins.

pChart:

pChart es una biblioteca (librería) escrita en PHP. Las gráficas que permite crear son estáticas y, por tanto, no interactivas, ya que se generan en un archivo de imagen con formato PNG. Está liberada bajo una licencia GPL y los gráficos que genera son de alta calidad.

Principales características:

- Recibe datos desde consultas SQL, archivos CSV o de datos suministrados de forma manual;
- Los gráficos utilizan algoritmos como aliasing para que estos sean de alta definición;
- Permite crear muchos tipos de gráficas, entre las que se encuentran: pie (torta), barras, lineales, 3D, planos, etc.

TCPDF:

TCPDF es una clase/librería totalmente libre y gratuita escrita en PHP para generar documentos PDF. En un principio fue creada con el objetivo de solucionar carencias del FPDF pero finalmente se ha convertido en una versión mucho más potente, con gran cantidad de posibilidades.

Dos de las cualidades más apreciadas de esta clase, es su simplicidad a la hora de crear archivos PDF y la capacidad de interpretar código XHTML.

Como características principales tiene:

- Soporta formatos personalizados de páginas, márgenes y unidades de medida.
- No requiere ninguna librería externa
- Se distribuye en 2 versiones, para PHP4 y PHP5
- Soporta código HTML
- Soporta ISO y UTF-8

- Personalización de encabezado y pie de página
- Personalización de márgenes y tipo de página
- Soporta numeración de páginas y grupos de páginas
- Y algo muy importante, está completamente documentado.

2.3. Representación Arquitectónica

En este documento la arquitectura es representada a través de una serie de vistas que se presentan a continuación, las cuales fueron previamente explicadas en el epígrafe 1.7 del Capítulo anterior:

- Vista de Caso de Uso
- Vista Lógica
- Vista de implementación
- Vista de despliegue

Estas vistas serán modeladas utilizando RUP como metodología de desarrollo (epígrafe 1.13.3 Capitulo 1), y UML 2.0 (epígrafe 1.12 Capitulo 1). Además de estas 4 vistas se agregará una última que será la Vista de Datos.

2.4. Objetivos y Restricciones Arquitectónicas

Los objetivos y restricciones arquitectónicas vienen dados en gran medida por los requisitos no funcionales del sistema, que no son más que las propiedades o cualidades que el producto debe tener.

2.4.1. Requerimiento de software

- El sistema deberá correr sobre sistema operativo libre (GNU/Linux cualquier distribución)
- El sistema debe brindar servicio a través del servidor web Apache2
- El sistema debe permitir cargar los datos desde distintos orígenes y en distintos formatos.
- La PC cliente debe contar con el navegador Web estándar Mozilla Firefox v 1.5 o superior instalado.
- El servidor donde se encontrará la aplicación debe tener instalado los lenguajes de programación Python 2.6 y PHP 5.

2.4.2. Requerimiento de Hardware

- El sistema debe ser instalado en un servidor que cuente con los siguientes requisitos mínimos de hardware: Procesador Intel Pentium 4 de 1.7 GHz, 1 GB de RAM**,** 10 GB de espacio libre en disco duro.
- La base de datos debe ser instalada en un servidor que cuente con los siguientes requisitos mínimos de hardware: Procesador Intel Pentium 4 de 1.7 GHz, 512 GB de RAM, 20 GB de espacio en disco duro.
- Para utilizar el sistema los usuarios deberán conectarse desde una PC que cuente con los siguientes requisitos mínimos de hardware: Procesador Intel Pentium 4 de 1.7 GHz, 256 MB RAM.

2.4.3. Requerimiento de Usabilidad

- El sistema debe ser intuitivo y de alta usabilidad, para que los usuarios, aunque no sean personas expertas en la rama de la informática, necesiten poco tiempo de adiestramiento y puedan explotarlo al máximo y ser capaces de trabajar con él en un corto período de tiempo.

2.4.4. Requerimiento de Soporte

- Para garantizar el soporte a los clientes se les brindara la posibilidad de emitir sus quejas y sugerencias a los desarrolladores del sistema, ya sea por correo electrónico o por cualquier otra vía de comunicación por un periodo de tiempo límite.

2.4.5. Requerimientos de Seguridad

- Para gestionar la seguridad se hará uso del componente de seguridad (A.A.A.) desarrollado por el Centro de Compatibilización para Intereses de la Defensa.

2.4.6. Portabilidad

- El sistema solamente podrá ser instalado sobre sistemas operativos GNU/Linux.
- El sistema deberá permitir ser accedido desde cualquier sistema operativo, ya sea libre o propietario, siempre que se cuente con el navegador Mozilla Firefox versión 1.5 o superior instalado.

2.4.7. Restricciones en el diseño y la implementación

- El sistema debe ser implementado en lenguaje de programación Python, versión 2.6. y PHP, versión 5.
- El sistema debe utilizar como gestor de bases de datos PostgreSQL versión 8.2 o superior.

2.5. Tamaño y Rendimiento

En un sistema como este, con una arquitectura cliente-servidor, cuando se analiza su tamaño y rendimiento hay que tener muy en cuenta el servidor web y el sistema gestor de base de datos (SGBD) utilizado.

El SGBD que se utiliza es PostgreSQL, el cual ha ganado una muy buena reputación por su rendimiento y confiabilidad e integridad de datos gracias a un grupo de características como un sofisticado analizador/optimizador de consultas, integridad transaccional, la creación de puntos de recuperación, la replicación asincrónica, transacciones jerarquizadas, soporte multi-usuario, el uso de un modelo cliente/servidor conocido como "proceso por usuario", un sistema de registros para la tolerancia a fallos, integridad referencial y un Control de Concurrencia Multi-Versión (MVCC) que permite que varios usuarios puedan leer y escribir a la misma vez sobre una misma tabla sin bloquearse ni denegarse servicios.

El modelo de datos del sistema cuenta con apenas 24 tablas representadas en un esquema donde la mayor de las tablas no excede las 9 columnas y en total suman 78 columnas. Este modelo de datos es posible de manejar eficientemente mediante PostgreSQL, ya que este gestor soporta un tamaño ilimitado de la base de datos, las tablas pueden almacenar hasta 32 TB de datos, el tamaño máximo de de un registro es de 1.6 TB. Cada campo puede almacenar hasta 1 GB y pueden existir entre 250 a 1600 campos por tabla en dependencia de los tipos de datos usados y la cantidad de tuplas por tabla no tiene límites.

Por otra parte está el servidor web Apache, el cual soporta una gran carga de concurrencia y además es posible definir un número determinado de conexiones evitando que se exceda de la cantidad que es capaz de atender. Es un servidor altamente configurable con un diseño modular que permite seleccionar los

módulos que va a incluir ya que si fueran incluidas todas sus funcionalidades en una única versión de Apache lo haría muy pesado en cuanto a requerimientos de Memoria RAM y espacio en Disco Duro.

2.6. Estilos arquitectónicos

Como se vio en el capítulo anterior los estilos arquitectónicos son el nivel de abstracción más alto de la estructura del sistema, la elección del mismo está dada por el tipo de aplicación que se vaya a desarrollar y describe y proporciona las propiedades básicas de la arquitectura y es encargado de imponer los límites de su evolución. Los estilos presentes en el sistema son los siguientes:

- **Cliente-Servidor:** Al tratarse de una aplicación web se caracteriza por existir un nodo servidor web (Apache 2), un nodo servidor de bases de datos (PostgreSQL) y varios clientes que acceden al sistema a través de un navegador.

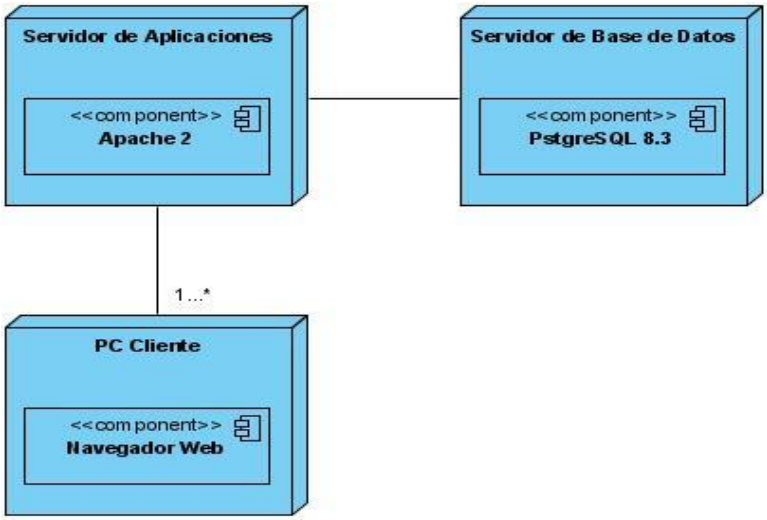


Figura 7 Estilo Cliente-Servidor

- **M-V-C**: Es el que provee Symfony para el desarrollo del sistema, el cual se basa en separar el sistema en la representación visual, el modelo del negocio y las acciones que controlan el flujo de datos, definiendo tres elementos: la vista, el modelo y el controlador.

- **Arquitectura en capas:** Organiza los componentes del sistema en capas con responsabilidades bien definida y la principal restricción que le impone al sistema es que los componentes de las capas brindan servicios a la capa inmediatamente superior y se sirve de las prestaciones que le brinda la inmediatamente inferior. Las capas en que se encuentra dividido el sistema son: presentación, lógica de negocio, acceso a datos y datos.

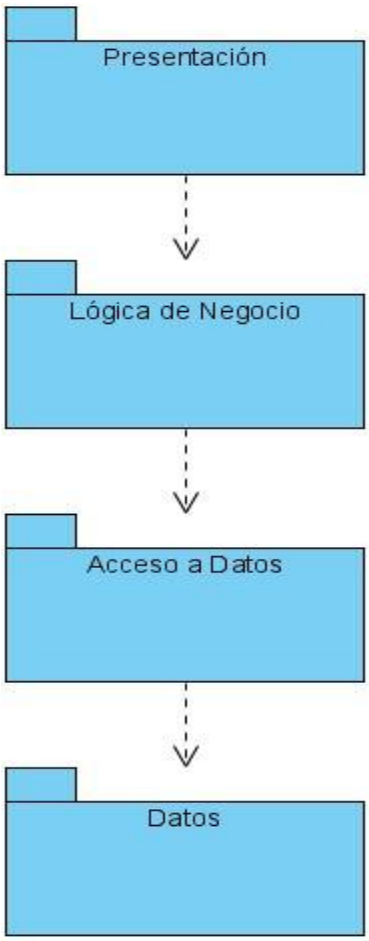


Figura 8 Estilo Arquitectura en Capas

Estos estilos se utilizan combinados, aprovechando sus ventajas y desempeño arquitectónico para dar solución a la arquitectura del sistema. Esto permite que se realice un diseño modular que posibilita un desarrollo incremental del sistema.

A continuación se describe cada una de las capas del sistema teniendo en cuenta la combinación de los estilos que le dan forma al mismo:

- Capa de Presentación:

La capa de presentación o vista contiene las funcionalidades necesarias para permitir que los usuarios interactúen e intercambien información con el sistema. Está soportada sobre los frameworks Extjs 2.2 y OpenJacob para el desarrollo de aplicaciones en Javascript, las cuales residen del lado del cliente en tiempo de ejecución. La comunicación entre la esta capa y la capa lógica de negocio se realiza mediante solicitudes AJAX de esta y las respuestas pueden ser dadas a través de JSON o XML según corresponda a la solicitud.

- Capa Lógica de Negocio:

La capa lógica de negocio o controlador recibe los datos capturados en las interfaces de usuarios, los cuales son procesados o gestionados según las funcionalidades y las reglas del negocio del sistema. Para llevar a cabos algunas de las funcionalidades del sistema se hace necesario el uso de componentes que se encuentran desarrollados en python, estos componentes son los que implementan los algoritmos de minería de datos. La comunicación de estos componentes desarrollados en python con los que se encuentran en php se realiza a través de ficheros. Desde php se crean dos fichero temporales, uno con los datos que van a utilizar los componentes desarrollados en python y el otro es donde se va a escribir el resultado del procesamiento de los datos del primero al ejecutarse el componente desarrollado en python mediante la función shell_exec() de php.

- Capa de Acceso a Datos:

La capa de acceso a datos o modelo contiene las funcionalidades requeridas para acceder a la base de datos y toda la lógica de acceso a los datos. Para el desarrollo de esta capa se hace uso del framework Propel como ORM (object-relational-mapping) que traduce la lógica de los objetos de php 5 y Symfony a la lógica relacional de la base de datos y que a su vez usa a Creole para la abstracción a la base de datos.

- Capa de Datos:

Esta capa contiene el modelo de datos del sistema. Está compuesta por tablas y sus relaciones que almacenan los datos requeridos por el sistema que se encuentran en el servidor de base de datos.

2.7. Vista de Casos de Uso

En esta vista se representa los casos de usos arquitectónicamente significativos, seleccionados a partir de los casos de uso críticos en cada uno de los módulos que conforman el proyecto; estos serian los casos de uso que encierran las funcionalidades principales del sistema de acuerdo a las necesidades del cliente.

2.7.1. Módulo Diseñador de modelos

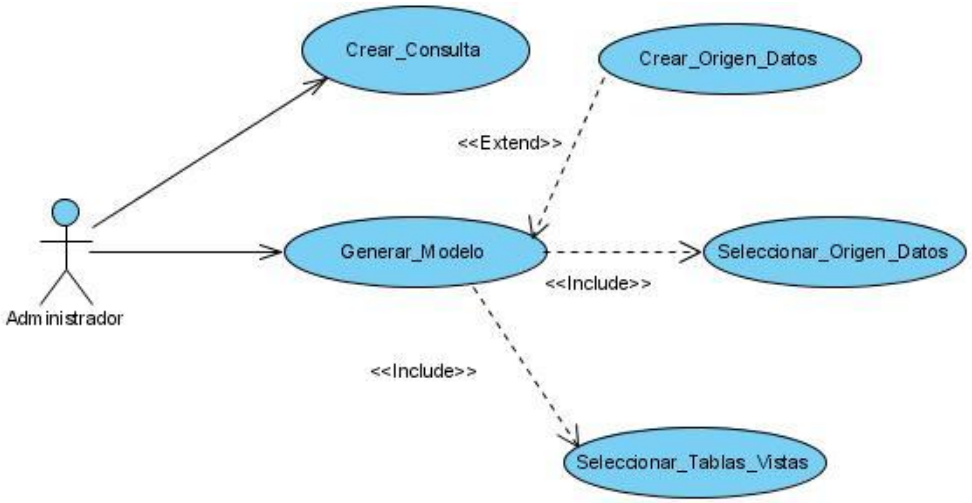


Figura 9 DCU Diseñador de Modelos

Crear_Consulta: Posibilita la creación de consultas a partir de las entidades presentes en la BD del cliente y que esta (la consulta) sea incluida en el modelo.

Crear_Origen_Datos: Posibilita la creación del origen de datos al cual se conectará el usuario para la creación del modelo semántico, pudiendo el mismo ser tanto una Base de Datos como un Servidor de Reportes.

Generar_Modelo: Permite la creación de un modelo semántico de la BD de los clientes, extrayendo los metadatos de las entidades (tablas, vistas y funciones). El Administrador debe haber seleccionado tablas para incluir en un modelo y un origen de datos válido.

Seleccionar_Origen_Datos: Para la creación de un modelo se puede seleccionar un origen de datos creado previamente, el sistema verifica la conexión y si está disponible muestra las tablas, vistas y funciones correspondientes al origen de datos seleccionado.

Seleccionar_Tablas_Vistas: Permite la selección de las entidades a incorporar en el modelo, así como los atributos de cada una de las entidades (campos de la tablas, vistas o funciones) que se desean como fuente de datos para los reportes. Es posible la edición de estos atributos y entidades con la creación de alias para abstraer al usuario final del nombre real que posee en la BD.

2.7.2. Módulo Administración

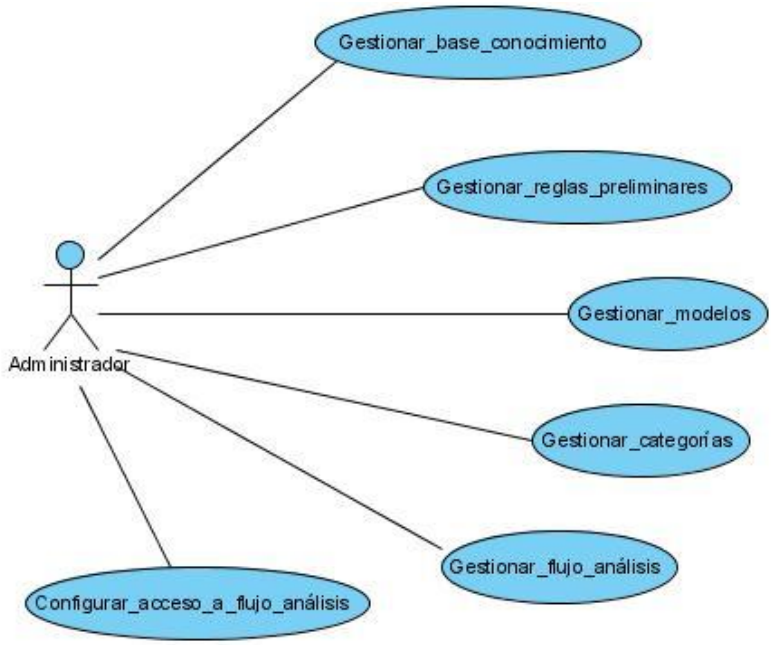


Figura 10 DCU Administración

Gestionar_Base_Conocimiento: Permite adicionar o eliminar la base de conocimiento que contienen la definición de los parámetros que permitirá obtener las reglas preliminares.

Gestionar_Reglas_Preliminares: Posibilita generar o eliminar las reglas preliminares que se van a utilizar en los algoritmos de clasificación supervisada.

Gestionar_Modelos: Permite eliminar, modificar y renombrar un modelo semántico que se encuentre creado en el sistema.

Gestionar_Categorias: Posibilita la adición, eliminación y modificación de las categorías en que se van a agrupar los flujos de análisis.

Gestionar_Flujo_analisis: Permite eliminar, mover de categoría y renombrar los flujos de análisis que fueron previamente creados.

Configurar_Acceso_a_Flujo_Analisis: Permite darle acceso a un usuario a determinados flujos de análisis o categoría de los mismos existente en el sistema.

2.7.3. Módulo Diseñador de flujo de Análisis

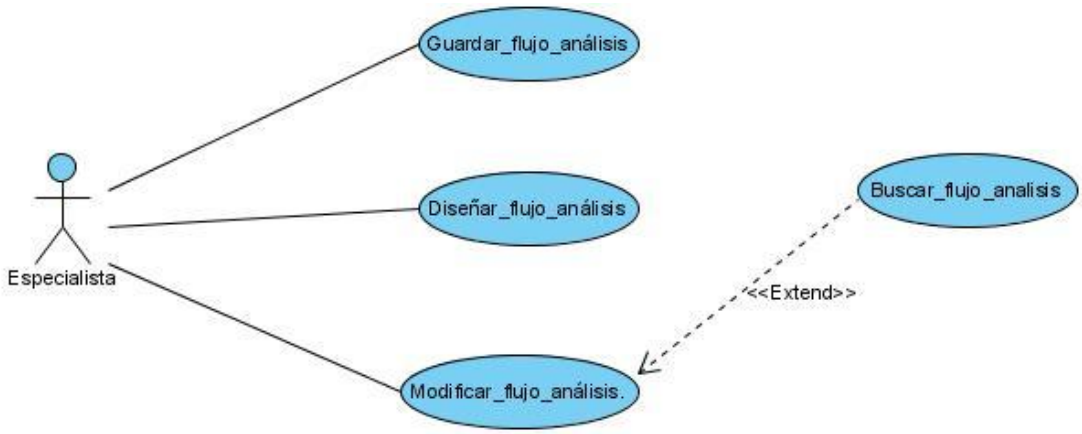


Figura 11 DCU Diseñador de Flujo de Análisis

Guardar_Flujo_Analisis: Permite generar el XML que contiene la estructura del flujo de análisis y guardarlo en la base de datos para su posterior uso. Si el flujo contiene alguna incoherencia no permite guardarlo hasta que la misma sea corregida.

Diseñar_Flujo_Analisis: Permite diseñar un flujo de análisis que no es más que la incorporación de los distintos componentes que representan los algoritmos existentes en el sistema, así como la interconexión entre los mismo; además de la configuración de cada uno de ellos.

Modificar_Flujo_Analisis: Permite modificar un flujo de análisis previamente creado.

Buscar_Flujo_Analisis: Busca en el sistema, por el nombre, un flujo de análisis determinado.

2.7.4. Módulo Visor de flujo de Análisis

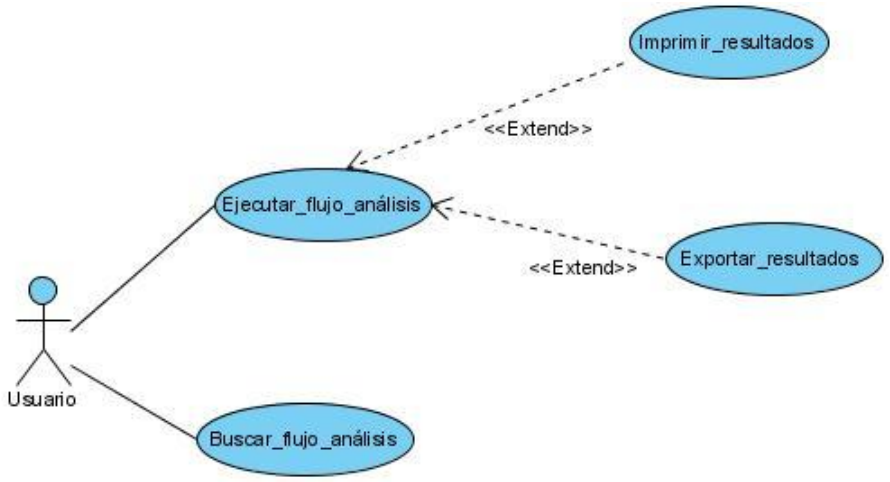


Figura 12 DCU Visor de Flujo de Análisis

Ejecutar_Flujo_Analisis: Permite ejecutar cada uno de los componentes que contiene el flujo de análisis seleccionado, así como visualizar el resultado de los mismos.

Imprimir_Resultados: Imprime los resultados obtenidos en el caso de uso anterior.

Exportar_Resultados: Exporta las resultados obtenidos, en el caso de uso Ejecutar_Flujo_Analisis, en diferentes formatos como PDF, CSV, TXT y otros.

Buscar_Flujo_Analisis: Busca en el sistema, por el nombre, un flujo de análisis determinado.

2.8. Vista Lógica

En esta vista se representa los elementos de diseño más importantes para la arquitectura del sistema. Se describen las clases más importantes, su organización en paquetes y/o subsistemas, así como la relación que existen entre ellos y la organización de estos a su vez en capas.

La siguiente figura muestra la división del sistema en cada uno de los módulos que lo componen, los cuales agrupan los casos de uso que ya han sido descritos en la Vista de Casos de Uso. Esta división en modulo facilita el mantenimiento y la reusabilidad del sistema, pues permite realizar cambios afectando solo al modulo que lo requiera. Además permite el desarrollo de los módulos en paralelo y saber cuál es la dependencia entre los mismos.

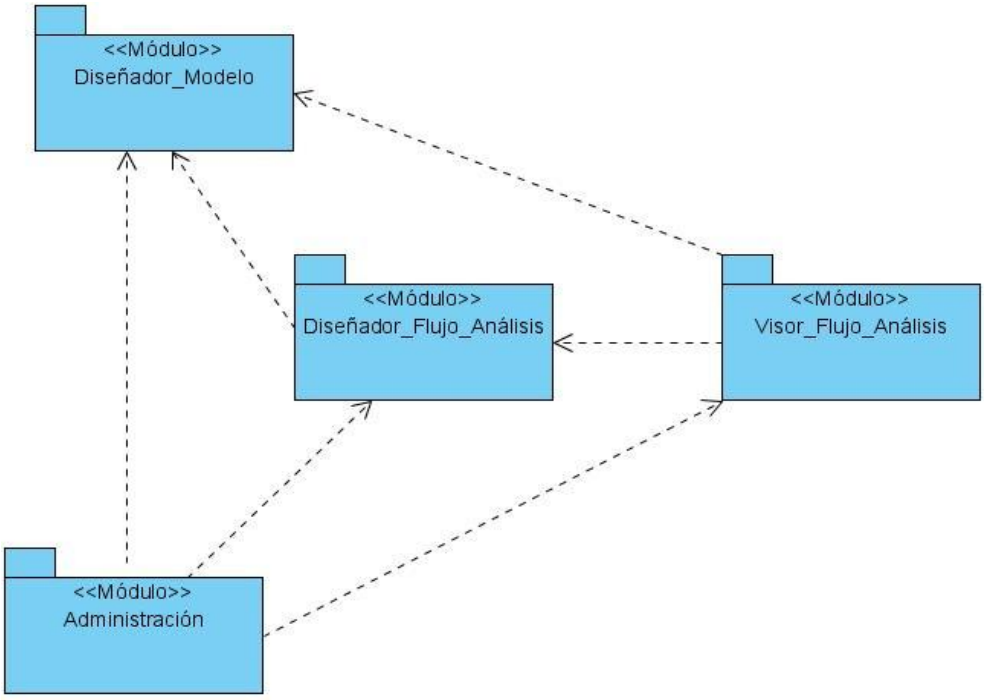


Figura 13 Dependencia entre los módulos de PATDSI

- Administración: Contiene la realización de los casos de uso referentes a la gestión de base de conocimiento, reglas preliminares, categorías y otros.
- Diseñador de flujo de Análisis: Contiene la realización de los casos de uso referentes al diseño y modificación de los flujos de análisis, así como guardar los mismos en la base de datos.

- Visor de flujo de Análisis: Contiene la realización de los casos de uso referentes a la ejecución de los flujos de análisis, la impresión de los resultados y la exportación de los mismos en diferentes formatos.
- Diseñador de modelos: Contiene la realización de los casos de uso referentes a la generación de modelos semánticos, la creación de consultas en los mismos y la creación de orígenes de datos.

La distribución en capas del sistema para cada uno de los módulos se hará de la siguiente forma:

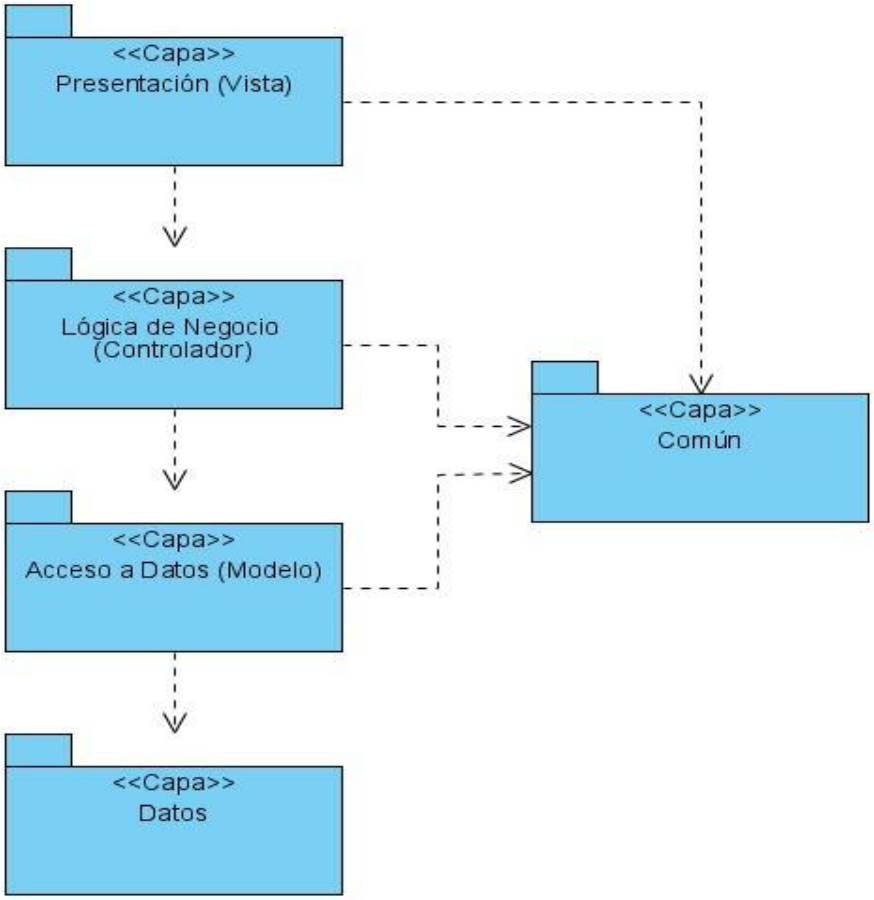


Figura 14 Estructura en capas del sistema

A continuación se presenta la estructura en paquetes de cada una de las capas vistas con anterioridad a excepción de la capa de datos que representa a la base de datos del sistema.

Presentación: Esta capa se encarga del manejo de las interfaces necesarias para que los usuarios interactúen con el sistema. En la siguiente figura se observan los paquetes plantilla, js y css que componen esta capa. El paquete js contiene los ficheros y clases javascript regidos por los frameworks Extjs 2.2 y OpenJacob. El paquete plantilla contiene la plantilla php en la que se cargan los ficheros javascript del paquete js. El paquete css contiene el conjunto de hojas de estilo de la aplicación.

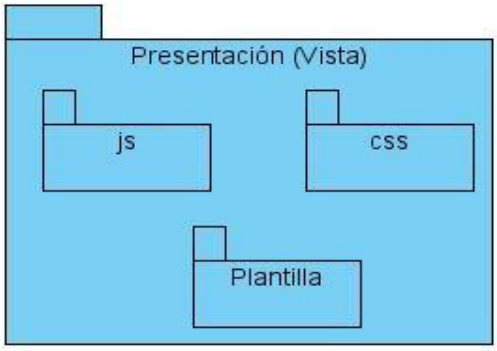


Figura 15 Paquetes de la capa de presentación

Lógica de Negocio: En la siguiente figura se observan los paquetes Controlador Frontal, Acción y Negocio que componen esta capa. El paquete Controlador Frontal es el encargado de atender las peticiones al servidor, es el punto de entrada a la aplicación y su principal función es invocar la acción correspondiente para cada solicitud y cargar la configuración. El paquete Acción contiene las clases action responsables de la lógica del negocio, verifican la integridad de las peticiones y preparan los datos requeridos por la capa de presentación. El paquete Negocio contiene un conjunto de clases que junto a las clases action complementan la lógica de negocio.

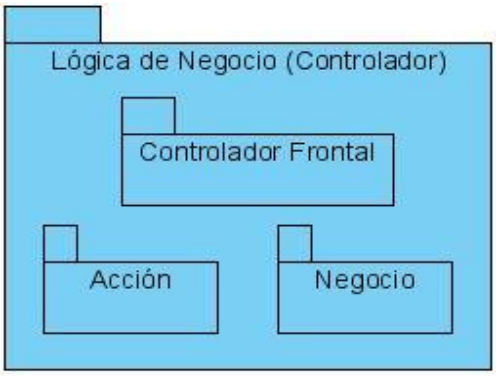


Figura 16 Paquete de la capa de lógica de negocio

Acceso a Datos: En la siguiente figura se observan los paquetes Acceso a Datos y Abstracción Base Datos que conforman esta capa. El paquete Acceso a Datos está compuesto por clase de generadas por Propel mediante el mapeo a la base de datos. El paquete Abstracción Base Datos es gestionado a través de Creole, que proporciona una interfaz entre el código PHP y el código SQL de la base de datos, permitiendo cambiar fácilmente de sistema gestor de bases de datos.

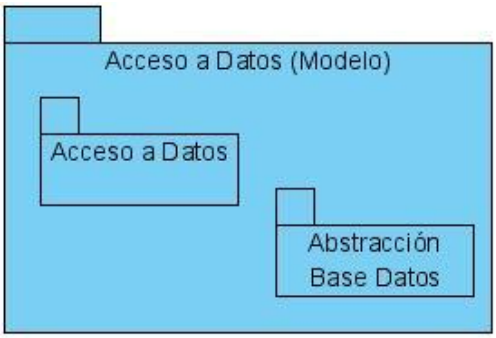


Figura 17 Paquete de la capa de acceso a datos

Común: En la siguiente figura se observan los paquetes Symfony, TCPDF, Extjs, pChart y OpenJacob que representan a los distintos framework y librerías que se utilizan para el desarrollo del sistema. Además se encuentra el paquete Algoritmos que contiene los componentes python que implementan los algoritmos de minería de datos.

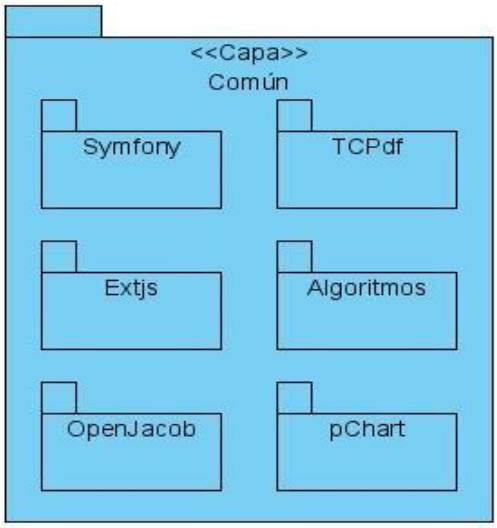


Figura 18 Paquete de la capa Común.

2.9. Vista de Despliegue

La Vista de Despliegue es aquella donde se propone la distribución física del sistema, así como la distribución de los componentes de la aplicación en los mismos y como son satisfechos los requisitos no funcionales de software y hardware, así como la influencia de los mismos en el rendimiento de la aplicación.

La Vista de Despliegue de PATDSI contará con dos escenarios los que serán explicados a continuación:

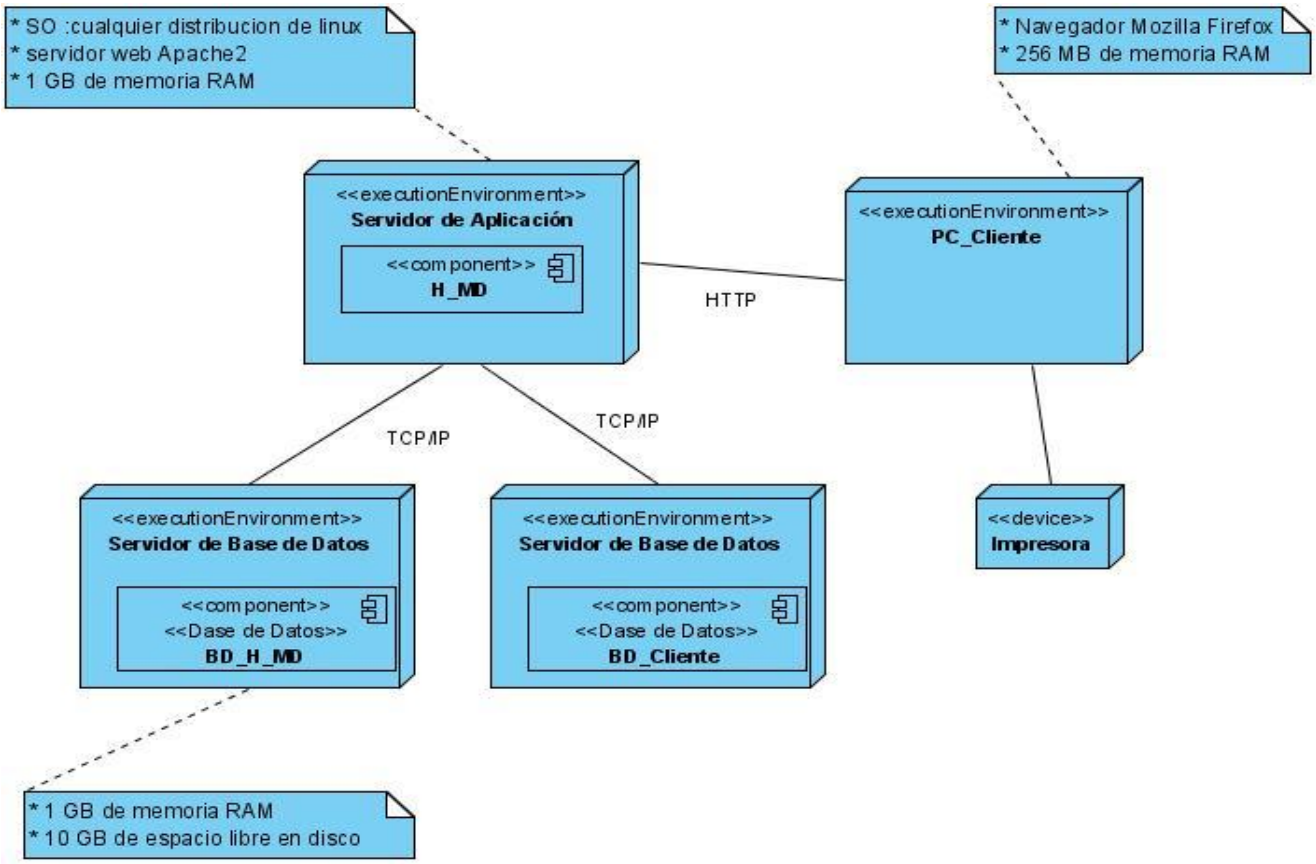


Figura 19 Escenario 1 Diagrama de despliegue

- Nodo **Servidor de Aplicación:** Este será el Nodo que contendrá la Herramienta de Minería de Datos, o sea la aplicación. Debe ser capaz de conectarse al Servidor de Base de Datos, donde se encuentra la base de datos de configuración propia de él, además de al Servidor de Base de Datos que contiene la base de datos de los clientes. Todas las PC Clientes deben ser capaces de conectarse a este servidor para utilizar la aplicación.
- Nodos **Servidor de Base de Datos:**
  - **1-** En este nodo se encontrará la base de datos que contiene la configuración y los componentes propios de la aplicación. El Nodo Servidor de Aplicación debe poder conectarse a este nodo en todo momento por la dependencia de funcionamiento que tiene el mismo con información almacenada en este nodo (Servidor de Base de Datos).

2- En este nodo se encontrará la base de datos cliente que contendrá la información a la que se le desea aplicar los algoritmos de minería de datos que contiene la aplicación, o sea la que será utilizada como fuente de datos por esta. Este nodo debe permitir la conexión del Nodo Servidor de Aplicación.

- Nodo **PC Cliente:** Desde este nodo los clientes podrán acceder a la aplicación que se encuentra en el Nodo Servidor de Aplicación utilizando el navegador Mozilla Firefox.
- El dispositivo **Impresora** será utilizado por los clientes para imprimir los resultados que se obtengan a partir de la utilización de la aplicación. Puede estar conectado tanto en paralelo como por USB.

Los nodos Servidor de Aplicación y Servidor de Base de Datos (1) deben trabajar con sistema operativo libre, además el nodo PC cliente debe contar con el navegador Mozilla Firefox instalado (epígrafe 2.4).

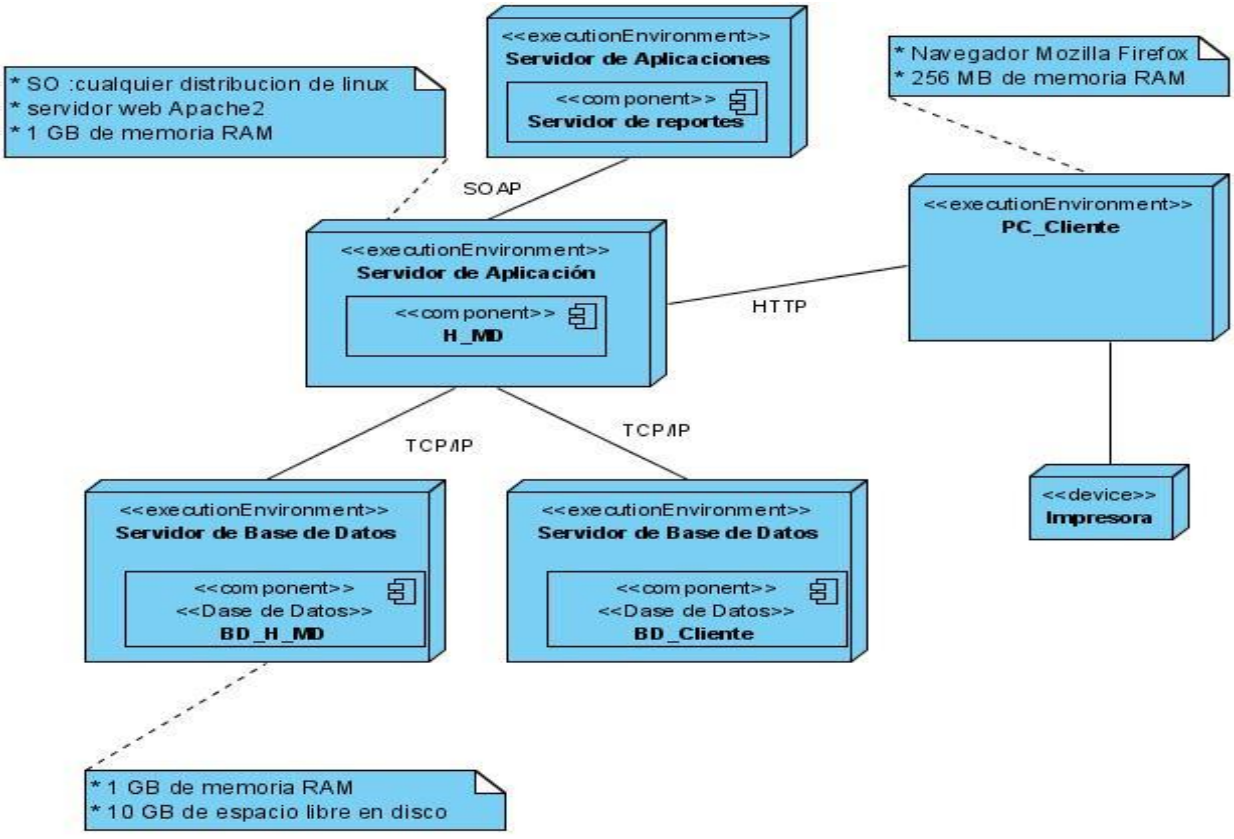


Figura 20 Escenario 2 Diagrama de despliegue

En este escenario los nodos Servidor de Aplicación, Servidor de Base de Datos (1 y 2), PC Cliente y el dispositivo Impresora son los mismos que se describen en el escenario 1.

- Nodo **Servidor de Aplicaciones:** Este nodo contendrá el Servidor de Reporte el que será utilizado por la aplicación como fuente de datos, o sea los reportes que este contiene, para aplicársele los algoritmos de minería de datos.

2.10. Vista de Implementación.

La Vista de implementación muestra y describe el conjunto de subsistemas, paquetes y componentes de implementación en que se descompone el sistema. Un componente es una unidad física de implementación que encapsula una o más clases del diseño. Estos se pueden agrupar en subsistemas de implementación, para mayor organización y claridad del diagrama. En resumen, la vista de implementación de la arquitectura muestra la organización de los elementos físicos reales del sistema.

La organización física del sistema está dada por la estructura que propone Symfony en forma de árbol (Ver Anexo 5 Estructura de la raíz del proyecto de Symfony.), donde la raíz la representa el proyecto. Dentro de un proyecto, las operaciones se agrupan de forma lógica en aplicaciones, donde cada una de ellas normalmente se ejecutan de forma independiente. Cada aplicación está formada por uno o más módulos que se dividen en acciones, que representan cada una de las operaciones que estos puede realizar. (22)

Entre las carpetas más importantes se encuentran las siguientes:

- ***apps:*** Contiene un directorio por cada aplicación del proyecto que a su vez contiene la carpeta modules/ que separa los componentes específicos de cada módulo.
- ***lib:*** Guardar el código común a todas las aplicaciones del proyecto. El subdirectorio model/ guarda el modelo de objetos del proyecto. Por su parte symfony/ agrupa las clases que implementan el framework.
- ***web:*** Constituye la raíz web del sistema. Contiene los únicos archivos accesibles desde Internet.

En correspondencia a esta estructura que ofrece Symfony, en la siguiente figura se muestra como queda conformada la organización física del sistema en módulos y una representación de sus principales acciones.

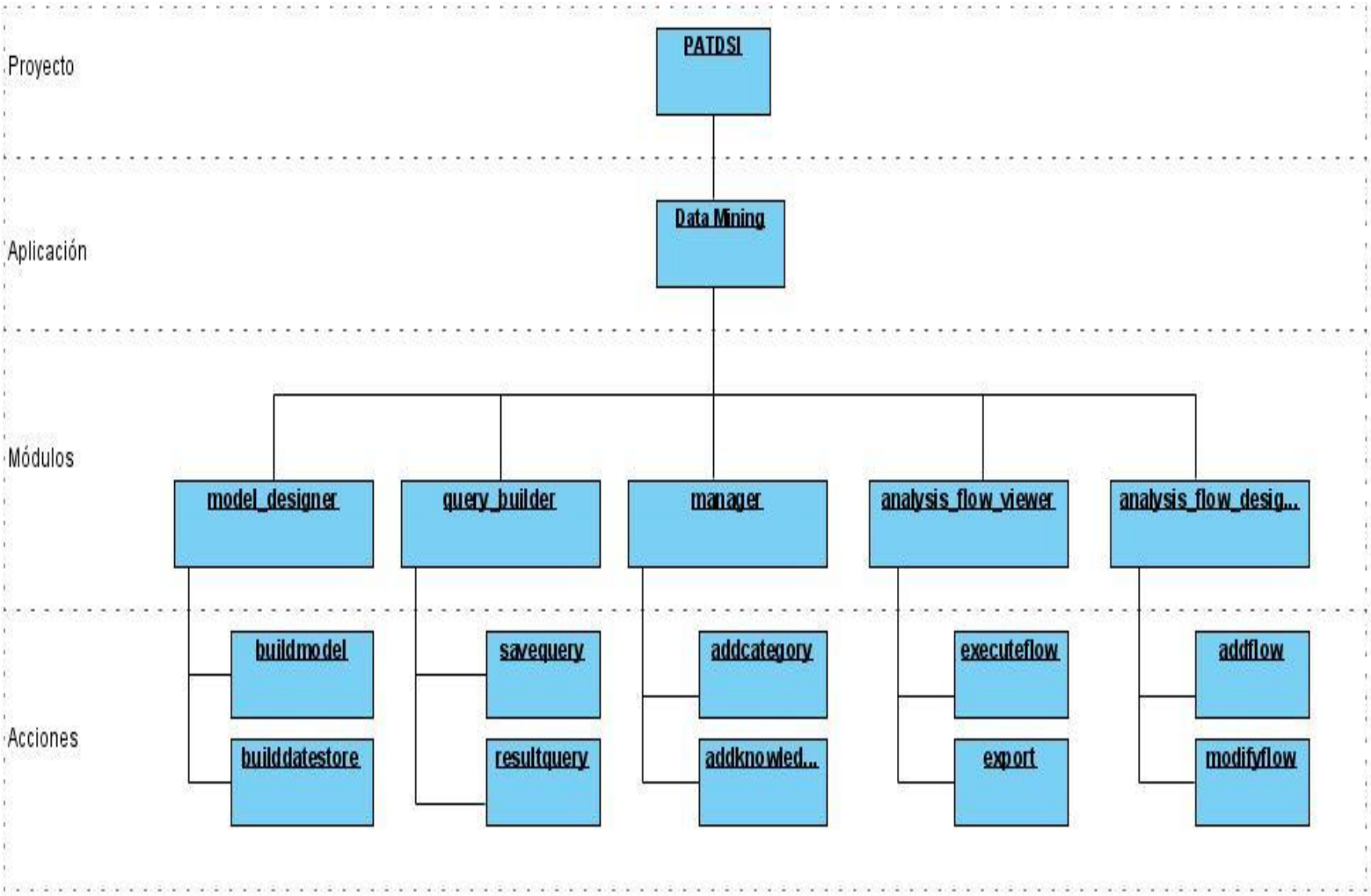


Figura 21 Estructura del sistema según Symfony

En la siguiente figura se representa la relación de dependencia entre componentes de implementación correspondientes a cada una de las capas en que se encuentra distribuida la arquitectura del sistema. Visto horizontalmente se encuentra las capas de Presentación, de Lógica del Negocio, de Lógica Acceso a Datos y de Datos las cuales cuentan con los subsistemas encargados de realizar las tareas

correspondientes a cada una de ellas, mientras que paralela a estas capas se encuentra la capa Común la cual agrupa los framework, librerías y componentes desarrollados en Python que son utilizados por los componentes de cada una de las capas anteriores.

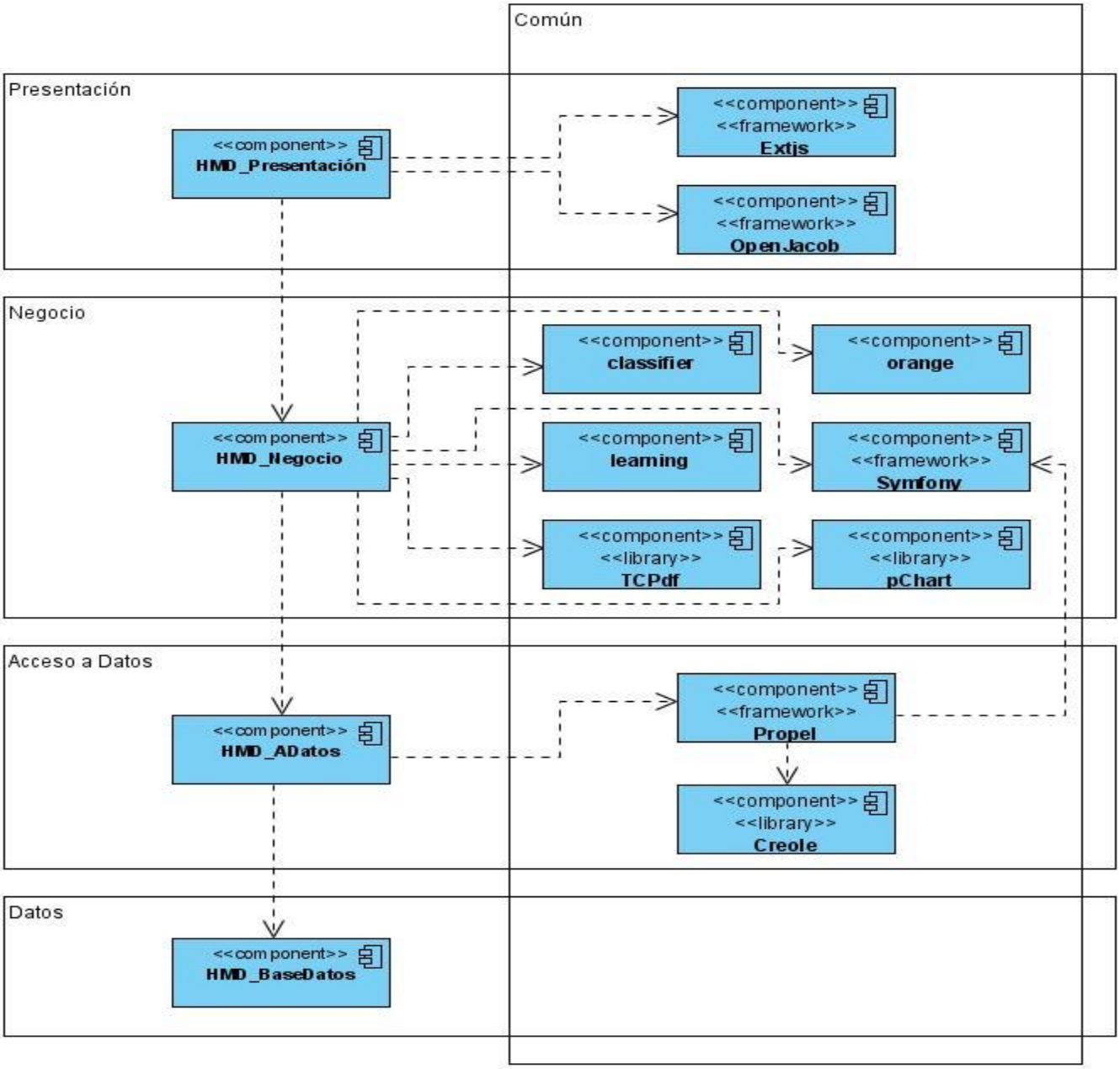


Figura 22 Vista de Implementación distribuida en capas

Para realizar una descripción más detallada del sistema se presenta a continuación los principales componentes que componen las capas de Presentación, de Lógica del Negocio y de Lógica Acceso a Datos.

2.10.1. Capa de Presentación

En la siguiente figura se agrupan el conjunto de componentes pertenecientes a esta capa que conforman la presentación del sistema. El paquete js contiene los framework Extjs y OpenJacob, así como los componentes javascript (commun_components, manager, query_builder, model_designer analysis_flow_designer y analysis_flow_viewer) y las relaciones de dependencia entre estos. El componente commun_components contiene ficheros javascript que son comunes para los componentes analysis_flow_designer y analysis_flow_viewer. El componente css agrupa los ficheros hojas de estilo que son utilizados por los componentes javascript.

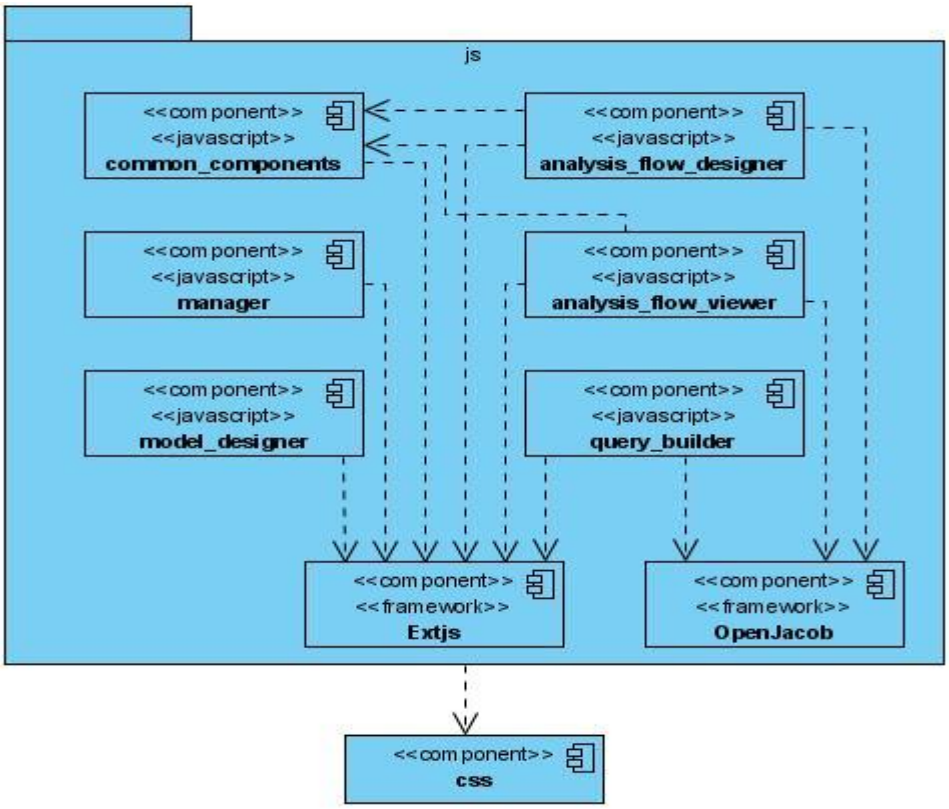


Figura 23 Diagrama de componentes de la capa de presentación

2.10.2. Capa de Lógica de Negocio

En la siguiente figura se representa los componentes que conforman esta capa. El paquete lib contiene las librerías pChart y TCPDF, el framework Symfony y los componentes desarrollados en Python (Orange, Lerning y Classifier) que permiten el funcionamiento de los componentes .php manager, query_builder, model_designer, analysis_flow_designer y analysis_flow_viewer que representan las acciones y objetos del negocio que componen la lógica del sistema.

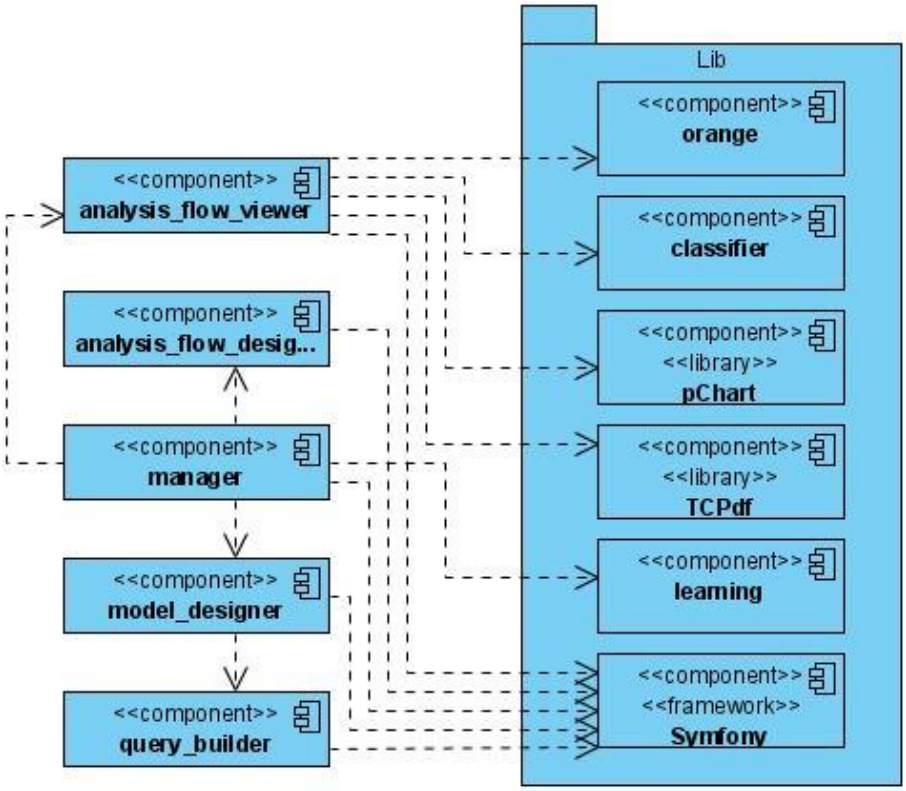


Figura 24 Diagrama de componentes de la capa de negocio

2.10.3. Capa de Acceso a Datos.

En la siguiente figura están representados los componentes om, map, model, Propel y Creole que permiten el acceso a los datos almacenados en la base de datos del sistema. El componente om contiene las clases bases del modelo de datos, que son las que se generan directamente a partir del esquema (descripción del modelo relacional) de la base de datos. El componente model representa el conjunto de

clases que se encargan del acceso a datos y el componente map posee la metainformación relativa a las tablas que son necesarias para la ejecución de la aplicación.

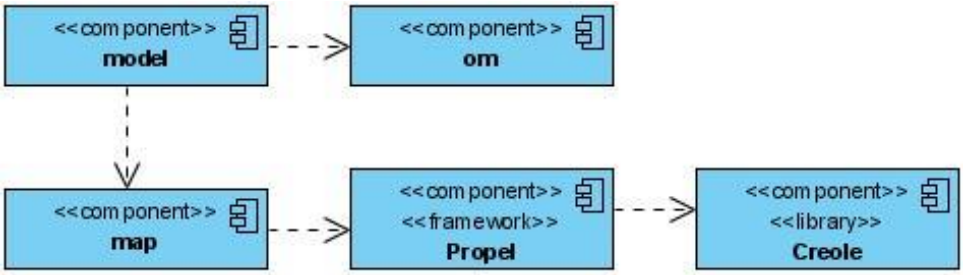


Figura 25 Diagrama de componentes de la capa de acceso a datos

2.11. Vista de Datos

En esta vista se muestra los elementos persistentes del sistema mediante el modelo de datos y la descripción de las tablas más importantes.

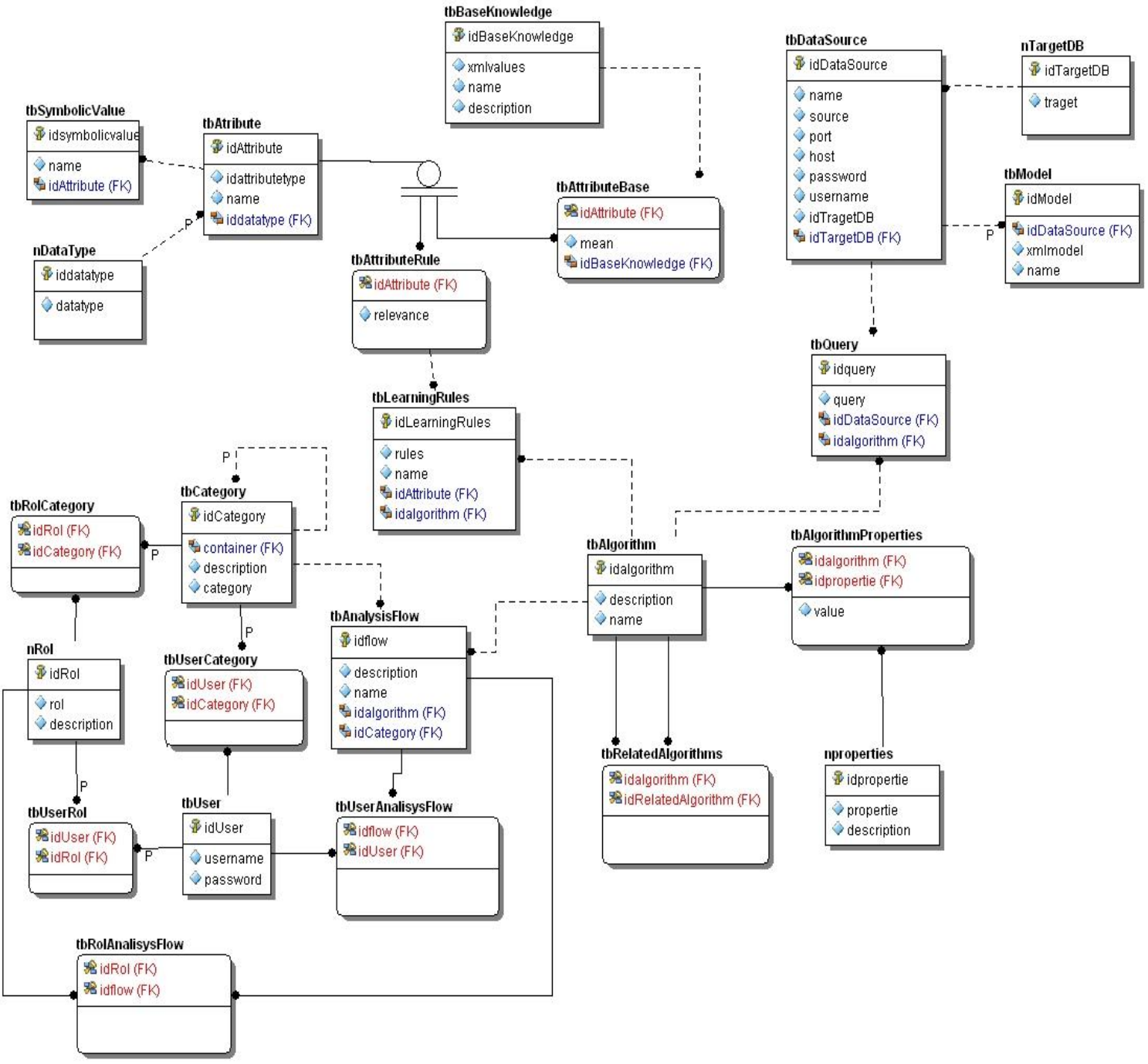


Figura 26 Modelo de datos del sistema

A continuación se divide el modelo de datos en varios subsistemas de entidades que responden a funciones específicas para su mejor descripción:

Subsistema Diseñador de modelos: Conjunto de entidades que almacena la información que define la estructura de los modelos semánticos y la fuente de datos a la cual pertenecen.

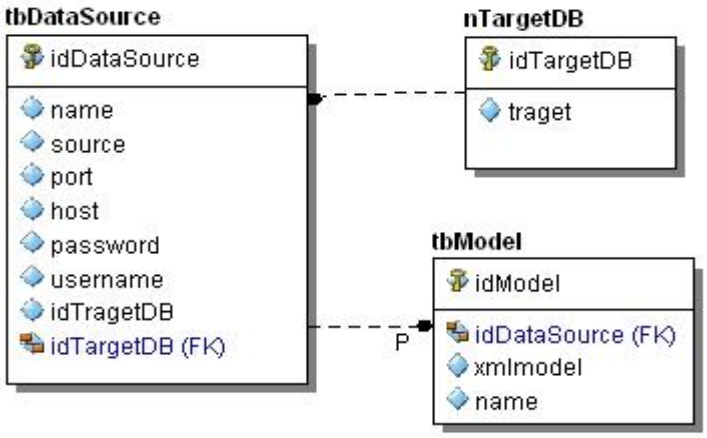


Figura 27 Modelos de datos del subsistema Diseñador de modelos.

Elemento	Identificador	Descripción
Tablas	tbModel	Almacena la definición de los modelos semánticos.
	tbDataSource	Almacena la información de los orígenes de datos sobre los cuales serán creados los modelos semánticos
	nTargetDB	Tabla nomencladora que almacena el conjunto de gestores de bases de datos soportados por el sistema para crear orígenes de datos.

Tabla 1 Descripción de las principales tablas del modelo de datos del subsistema Diseñador de Modelos

Subsistema Diseñador de flujos de análisis: Conjunto de entidades que almacena la información que define la estructura de los flujos de análisis.

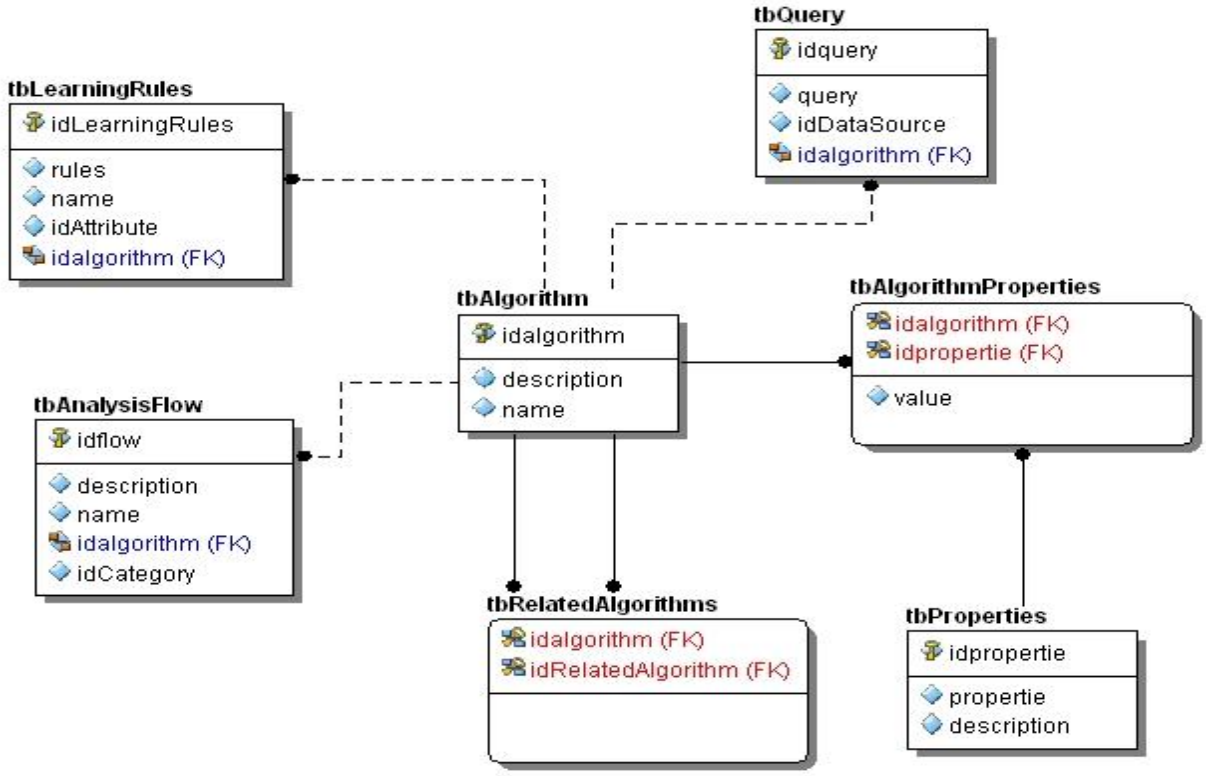


Figura 28 Modelos de datos del subsistema Diseñador de flujos de análisis.

Elemento	Identificador	Descripción
Tablas	tbAnalysisFlow	Almacena la definición de los flujos de análisis diseñados.
	tbLearningRules	Almacena las reglas de aprendizajes utilizadas por algunos algoritmos.
	tbQuery	Almacena las consultas que definen los datos a los cuales se les aplicarán los algoritmos.

	tbProperties	Almacena la información de las propiedades que contienen los algoritmos.
	tbAlgorithm	Almacena la información de los algoritmos que conformaran un flujo de análisis.

Tabla 2 Descripción de las principales tablas del modelo de datos del subsistema Diseñador de Flujo de Análisis

Subsistema Administrador: Conjunto de entidades que almacena la información que define la estructura de las bases de conocimientos, las reglas de aprendizaje, las categorías y los usuarios.

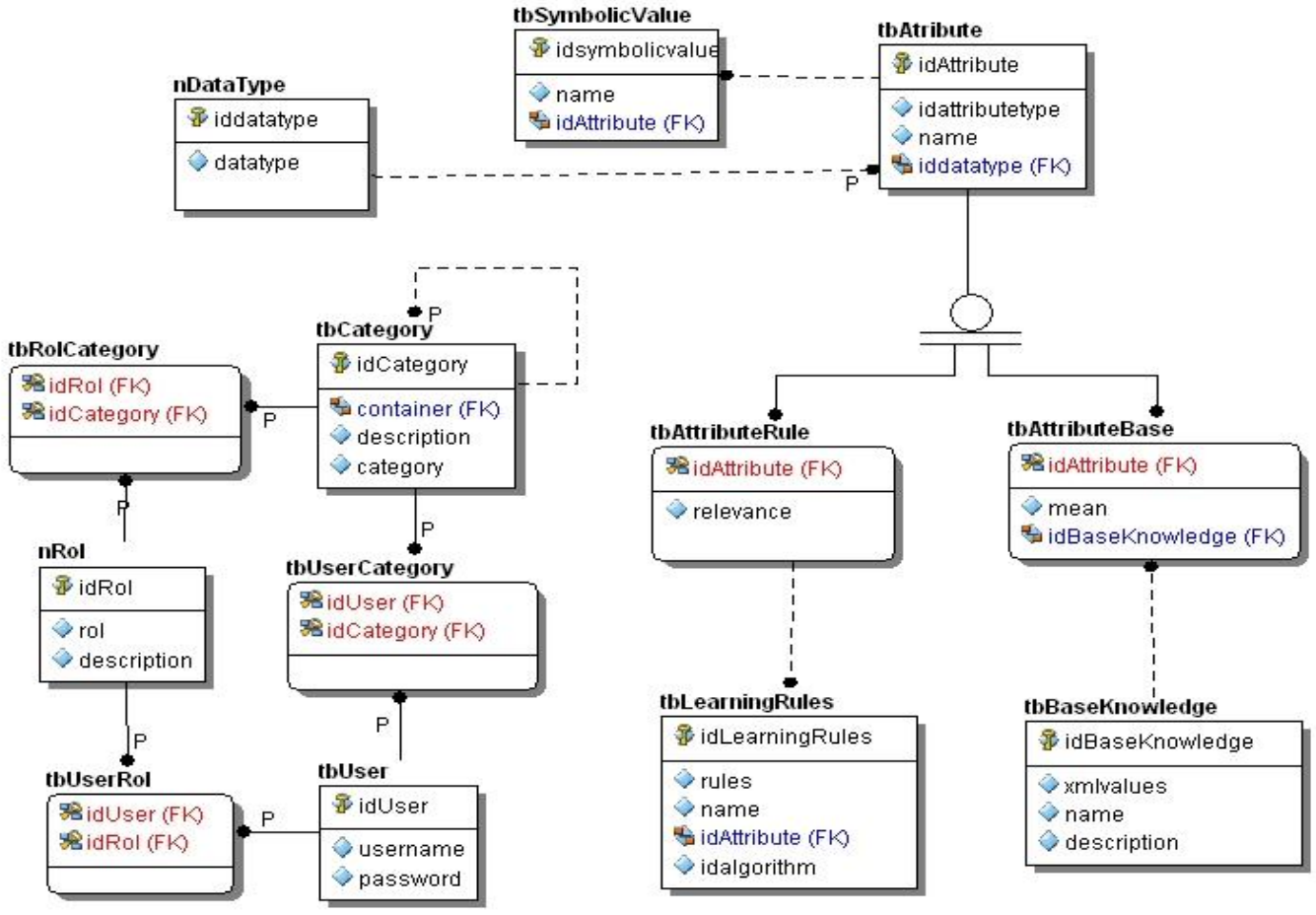


Figura 29 Modelos de datos del subsistema Administrador.

Elemento	Identificador	Descripción
	tbCategory	Almacena las categorías creadas a las que se asociarán los flujos de análisis o a otras categorías.
	tbUser	Almacena las cuentas de usuarios que accederán al sistema.
	nRol	Tabla nomencladora que almacena el conjunto de roles que serán asignados a los usuarios.
	tbBaseKnowledge	Almacena la información de las bases de conocimiento.
	tbAttributeBase	Almacena la información de los atributos con que contará una base de conocimiento.
	tbAttributeRule	Almacena la información de los atributos con que contará una regla de aprendizaje.
	tbSymbolicValue	Almacena la información de los valores simbólicos con que contará un atributo.
	nDataType	Tabla nomencladora que almacena el conjunto de tipos de datos que serán asignados a los atributos.

Tabla 3 Descripción de las principales tablas del modelo de datos del subsistema Administrador

2.12. Conclusiones

En este capítulo quedó diseñada la propuesta de Arquitectura de Software de la Herramienta de Minería de Datos, definiendo las vistas de Casos de Uso, la de Lógica, la de Despliegue, la de Implementación y la de Datos, así como los estilos arquitectónicos teniendo en cuenta los objetivos y restricciones del sistema. Además se fundamentaron los framework y librerías, lenguajes y demás herramientas y tecnologías seleccionadas para dar solución al sistema.

Capítulo 3

Evaluación de la solución

3.1. Introducción

En este capítulo básicamente se hace un estudio de los atributos de calidad presentes en la arquitectura, de la importancia existente en evaluar la misma, así como de algunos métodos existentes para la evaluación de esta.

3.2. ¿Por qué evaluar una Arquitectura?

En el campo del software, la arquitectura identifica los elementos más importantes de un sistema así como sus relaciones, es decir da una visión global del sistema. Esto es de suma importancia porque necesitamos la arquitectura para entender el sistema, organizar su desarrollo, plantear la reutilización del software y hacerlo evolucionar.

Por lo tanto cuanto más temprano se encuentre un problema en la arquitectura de un proyecto de 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 (23).

El mal diseño de una arquitectura puede llevar a un proyecto al fracaso. Todos los requerimientos de calidad pueden quedar insatisfechos.

La arquitectura también determina la estructura del proyecto: configuración, presupuesto, alcance, entre otros aspectos. Es mejor cambiar la arquitectura antes que otros artefactos, que están basados en ella, se estabilicen (23).

3.3. ¿Cuándo una Arquitectura puede ser evaluada?

Generalmente, la evaluación de la arquitectura ocurre después que esta ha sido especificada, pero antes que empiece la implementación. En un proceso iterativo y/o incremental, la evaluación se puede realizar al final de cada ciclo. Sin embargo, uno de los atractivos de la evaluación de arquitecturas es que se puede efectuar en cualquier etapa de la vida de una arquitectura. En particular, existen dos variaciones útiles: temprana y tardía (23).

3.3.1. Evaluación temprana

La evaluación no tiene porque esperar a que la arquitectura este totalmente especificada. Esta puede ser utilizada en cualquier etapa del proceso de creación de la arquitectura, para examinar las decisiones arquitectónicas ya tomadas y decidir entre las opciones que están pendientes.

Por supuesto, la completitud y fidelidad de la evaluación es directamente proporcional a la completitud y fidelidad de la descripción de la arquitectura (23)

3.3.2. Evaluación tardía

Esta variación toma lugar no solo cuando la arquitectura está terminada, también cuando la implementación esta completa. Este caso ocurre cuando la organización hereda un sistema legado. La técnica para evaluar un sistema legado es la misma que para evaluar un sistema recién nacido. Una evaluación es útil para entender el sistema legado, y saber si este cumple con los requerimientos de calidad y comportamiento.

En general, una evaluación debe realizarse cuando hay suficiente de la arquitectura como para justificarlo. Una buena regla sería: realizar una evaluación cuando el equipo de desarrollo empieza a tomar decisiones que dependen de la arquitectura y el costo de deshacerlas sobrepasa al costo de realizar una evaluación (23)

3.4. Atributos de calidad en la Arquitectura

No es del todo cierto que podamos decir que el sistema alcanzará todas sus metas de calidad con solo mirar la arquitectura. Pero muchos atributos de calidad yacen directamente en el reino de la arquitectura

(ver Anexo 6 Atributos de Calidad en la arquitectura). Una arquitectura puede ser evaluada en base a los siguientes atributos de calidad:

- **Disponibilidad (Availability):** Es la porción de tiempo en que el sistema está levantado y corriendo.
- **Funcionalidad (Functionality):** Es la habilidad del sistema de hacer el trabajo para el cual fue construido (24)
- **Desempeño (Performance):** Grado en el cual un sistema o componente cumple con su funcionalidad, dentro de ciertas restricciones dadas, como velocidad, exactitud o uso de memoria. (25).
- **Interoperabilidad (Interoperability):** Es la medida de la habilidad de que un grupo de partes del sistema trabajen con otro sistema.
- **Portabilidad (Portability):** Es la habilidad del sistema de correr sobre diferentes ambientes. Estos ambientes pueden ser de hardware, de software o una combinación de ambos.
- **Escalabilidad (Scalability):** Es el grado con el que se pueden ampliar el diseño arquitectónico, de datos o procedimental.
- **Reusabilidad (Reusability):** Es la capacidad de diseñar un sistema de forma tal que su estructura o parte de sus componentes puedan ser reutilizados en futuras aplicaciones.

3.5. Técnicas de evaluación de arquitecturas de software

Las técnicas existentes en la actualidad para evaluar arquitecturas permiten hacer una evaluación cuantitativa sobre los atributos de calidad a nivel arquitectónico, pero se tienen pocos medios para predecir el máximo (o mínimo) teórico para las arquitecturas de software. Sin embargo, debido al costo de realizar este tipo de evaluación, en muchos casos los arquitectos de software evalúan cualitativamente, para decidir entre las alternativas de diseño (26) . Bosch (26) propone diferentes técnicas de evaluación de arquitecturas de software, a saber: evaluación basada en escenarios, evaluación basada en simulación, evaluación basada en modelos matemáticos y evaluación basada en experiencia.

3.5.1. Evaluación basada en escenarios

De acuerdo con Kazman (27) un escenario es una breve descripción de la interacción de alguno de los involucrados en el desarrollo del sistema con éste. Este 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. Este último elemento es el que permite establecer cuál es el atributo de calidad asociado.

Los escenarios proveen un vehículo que permite concretar y entender atributos de calidad. Entre las ventajas de su uso están:

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

Actualmente las técnicas basadas en escenarios cuentan con dos instrumentos de evaluación relevantes, a saber: el Utility Tree propuesto por Kazman (27), y los Profiles, propuestos por Bosch (26).

3.5.2. Evaluación basada en simulación

Bosch (26) establece que la evaluación basada en simulación utiliza una implementación de alto nivel de la arquitectura de software.

El proceso de evaluación basada en simulación sigue los siguientes pasos (26):

- Definición e implementación del contexto.
- Implementación de los componentes arquitectónicos.
- Implementación del perfil.
- Simulación del sistema e inicio del perfil.
- Predicción de atributos de calidad.

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.

3.5.3. Evaluación basada en modelos matemáticos

Bosch establece que la evaluación basada en modelos matemáticos se utiliza para evaluar atributos de calidad operacionales. 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 (26).

Los siguientes pasos son los que sigue esta técnica para la evaluación:

- Selección y adaptación del modelo matemático.
- Representación de la arquitectura en términos del modelo.
- Estimación de los datos de entrada requeridos.
- Predicción de atributos de calidad.

3.5.4. Evaluación basada en experiencia

Bosch establece que en muchas ocasiones los arquitectos e ingenieros de software otorgan valiosas ideas que resultan de utilidad para la evasión de decisiones erradas de diseño. Aunque 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 puede ser la base de otros enfoques de evaluación (26).

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.

3.6. Métodos de Evaluación de la Arquitectura

3.6.1. SAAM - Software Architecture Analysis Method

SAAM es un método para la evaluación de Arquitecturas basado en escenarios que se centra para ello, principalmente, en el atributo de calidad modificabilidad. Este método ha demostrado en la práctica que es útil para evaluar otros atributos de calidad de forma rápida tales como portabilidad, extensibilidad,

integrabilidad, así como el cubrimiento funcional que tiene la arquitectura sobre los requerimientos del sistema.

SAAM puede ser utilizado para evaluar una o más arquitecturas. Si son evaluadas varias arquitecturas el mismo permite la comparación entre estas obteniendo como resultado final una tabla con las fortalezas y debilidades de cada una en los distintos escenarios. Si se evalúa una sola se obtendrá como resultado final un reporte con los componentes computacionales donde la arquitectura no alcanza el nivel requerido. En ninguno de los casos anteriores se emite un valor definitivo acerca de la calidad arquitectónica.

Roles en SAAM:

Los roles que se identificaron para este método son los siguientes:

- Interesados externos (Organización cliente, usuarios finales, administradores del sistema, etc.)
- Interesados internos (Arquitectos de Software, analistas de requerimientos)
- Equipo SAAM (Líder, expertos en el dominio de la aplicación, expertos externos en arquitectura)

Metodología:

- **Paso 1. Desarrollo de escenarios**

  La meta principal es capturar las principales actividades que el sistema debe soportar. Los escenarios encontrados deben reflejar los atributos de calidad de interés y también mostrar la interacción entre las diferentes personas que utilizarán el sistema: usuarios finales, administrador, mantenedor, desarrolladores, etc.

- **Paso 2. Descripción de la Arquitectura**
  En este paso se presentan las arquitecturas candidatas. . Se deben indicar los principales elementos de la arquitectura.
- **Paso 3. Clasificación de escenarios**
  Se clasifican los escenarios en directos o indirectos (es directo si la arquitectura puede soportar el escenario sin cambios).
- **Paso 4. Evaluación de escenarios**

Para cada escenario indirecto, se deben listar los cambios necesarios en la arquitectura para soportarlo, y el costo de llevarlos a cabo debe ser estimado.

- **Paso 5. Interacción de escenarios**
  Se deben determinar las interacciones de escenarios sobre cada componente de cada arquitectura.
- **Paso 6. Evaluación general**
  Un peso es asignado a cada escenario en términos de su influencia para que el sistema sea exitoso. El peso puede ser elegido de acuerdo a los objetivos del negocio, costos, riesgos, etc.

Fortalezas y debilidades:

Las principales fortalezas que posee este método son:

- Los interesados comprenden con facilidad las arquitecturas evaluadas.
- La documentación es mejorada.
- El esfuerzo y el costo de los cambios pueden ser estimados con anticipación.
- Analogía con el concepto de bajo acoplamiento y alta cohesión.

Las debilidades son:

- La generación de escenarios está basada en la visión de los interesados.
- No provee una métrica clara sobre la calidad de la arquitectura Evaluada.

3.6.2. ATAM – Architecture Tradeoff Analysis Method

ATAM es un método que especifica cuán bien satisface una arquitectura los atributos de calidad, además de que provee ideas de cómo interactúan y realizan concesiones mutuas esos atributos de calidad entre sí. Este método puede ser utilizado además para analizar sistemas legados cuando estos necesitan ser integrados con otros sistemas, modificados, etc.

Pasos del ATM:

El método consta de nueve pasos, divididos en cuatro grupos

- **Presentación:** donde la información es intercambiada.

- **Investigación y análisis:** donde se valoran los atributos claves de calidad requeridos, uno a uno con las propuestas arquitectónicas.
- **Pruebas:** donde se revisan los resultados obtenidos contra las necesidades relevantes de los clientes.
- **Informes:** donde se presentan los resultados del ATAM.

Presentación:

Paso 1: Presentar el ATAM (23)

- Los pasos del ATAM en resumen
- Las técnicas que serán utilizadas para la obtención y análisis
- Las salidas de la evaluación

Paso 2: Presentar las pautas del negocio

- Las funciones más importantes del sistema
- Toda restricción técnica
- Las guías de la arquitectura

Paso 3: Presentar la arquitectura

- Las restricciones técnicas
- Otros sistemas
- Propuestas arquitectónicas

Investigación y análisis:

Paso 4: Identificar las propuestas arquitectónicas

- Las propuestas arquitectónicas son identificadas por el arquitecto, pero no son analizadas.

Paso 5: Generar el árbol de utilidad de los atributos de calidad

- Los atributos de calidad que comprometen la utilidad del sistema (performance, availability, entre otros) son obtenidos, especificados en escenarios (con estímulos y respuestas) y priorizados. (ver Anexo 7 Árbol de utilidad).

Paso 6: Analizar las propuestas arquitectónicas

- Basados en los escenarios de mayor prioridad identificados en el paso 5, las propuestas arquitectónicas que cumplen con estos escenarios, son obtenidas y analizadas (por ejemplo, una propuesta arquitectónica que logra una meta de performance, será objeto de un análisis de performance). (ver Anexo 8 Propuesta arquitectónica (análisis))
- Durante este paso los riesgos arquitectónicos, los no riesgos, los sensitivity points y tradeoff points son identificados.

Paso 7: Lluvia de ideas y priorización de escenarios

Este paso consiste en la generación de nuevos escenarios para:

- Representar los intereses de los clientes que no hayan sido comprendidos

Paso 8: Analizar las propuestas arquitectónicas

- Se reitera las actividades del paso 6 utilizando el ranking de escenarios del paso 7.
- Estos escenarios se consideran casos de prueba para confirmar el análisis realizado hasta ahora.

Paso 9: Presentar los resultados

- El documento de propuestas arquitectónicas
- El conjunto de escenarios priorizados
- El árbol de utilidad
- Los riesgos descubiertos
- Los no riesgos documentados

- Los sensitivity points y tradeoff points encontrados.

3.6.3. ARID - An Evaluation Method for Partial Architectures

ARID es utilizado cuando es conveniente realizar la evaluación de diseños parciales en las etapas tempranas del desarrollo. Hacer revisiones en las etapas intermedias provee una valiosa visión de la viabilidad de la arquitectura a construir, además de que permite descubrir errores e inconsistencias en la misma. El método ARID cumple con estas características y se basa en las mejores cualidades de los métodos basados en escenarios (ATAM o SAAM) y las revisiones activas de diseños (ADRS).

De los ADRS toma la participación activa de los entrevistados lo cual es ideal por dos motivos:

- Asegura alta fidelidad en las respuestas.
- La participación activa puede captar la atención del grupo de compradores.

Del ATAM se queda con la idea de la generación de los escenarios por parte de todos los interesados, los usuarios le dirán a los diseñadores que es lo que ellos necesitan o mejor dicho que es lo que ellos esperan y si los diseñadores demuestran en la pruebas que el diseño cumple con los requerimientos también va a atraer el interés de los compradores.

¿Qué es ADRS? (23)**:**

ADRS son técnicas efectivas para asegurar la calidad dando diseños detallados del software.

ADR es utilizado para la evaluación de diseños detallados de unidades del software como los componentes o módulos. Las preguntas giran en torno a la calidad y completitud de la documentación y la suficiencia, el ajuste y la conveniencia de los servicios que provee el diseño propuesto.

Roles en ARID:

- Equipo de verificación (líder de la evaluación, arquitecto).
- Arquitecto.
- Revisores

Pasos de ARID:

ARID está compuesto por 9 pasos separados en dos fases.

- **Fase 1:** Pre reunión: Reunión entre el arquitecto y el líder de la evaluación.
- **Fase 2:** Evaluación: Los revisores son reunidos y la evaluación comienza

FASE 1

- **Identificar los revisores:** ingenieros de software que van a usar el diseño.
- **Preparar la presentación del diseño:** el arquitecto prepara un informe que explica el diseño, el mismo deberá ser lo suficientemente detallado como para que una capacitada audiencia pueda usar el diseño.
- **Preparar los escenarios:** el arquitecto y el líder preparan los escenarios bases que sirven para ilustrar los conceptos a los revisores.
- **Preparar los materiales:** copias de la presentaciones, escenarios.

FASE 2

- **Presentación del método:** el líder utiliza 30 minutos para explicar los pasos de la evaluación a los participantes.
- **Presentación del diseño:** el arquitecto realiza una presentación del diseño mostrando los ejemplos.
- **Lluvia de ideas y priorización de escenarios:** se proponen escenarios relevantes para solucionar problemas.
- **Aplicación de los escenarios:** El líder del grupo de revisión es el encargado de probar que el diseño provea los escenarios, comenzando por los de mayor prioridad, hasta que concluya el tiempo estimado de la evaluación, se hayan analizado los escenarios de mayor prioridad, o la conclusión de la revisión satisfaga a los implicados.
- **Resumen:** Al final, el facilitador recuenta la lista de puntos tratados, pide opiniones de los participantes sobre la eficiencia del ejercicio de revisión, y agradece por su participación.

Estos pasos se pueden ver más argumentados en el Anexo 9 Pasos del ARID.

3.7. Evaluando la arquitectura

Si las decisiones arquitectónicas determinan los atributos de calidad del sistema, entonces es posible evaluar dichas decisiones con respecto a su impacto sobre estos atributos.

Con vista a evaluar el diseño arquitectónico antes propuesto, se decidió utilizar el método ARID, que utiliza la técnica de evaluación basada en escenarios con el instrumento perfiles, donde se hace un análisis de los principales escenarios arquitectónicos. Para ello se seleccionaron los atributos de calidad más importantes en aquellos escenarios de acuerdo a un perfil específico.

Debido a que en la presente investigación los encargados de llevar a cabo la evaluación de la arquitectura propuesta serán los integrantes del grupo de arquitectura que definieron la misma, no se siguieron estrictamente los pasos propuestos por el ARID. Los arquitectos del sistema tienen un dominio completo del diseño arquitectónico del mismo, y presentan conocimientos del método a aplicar para la evaluación, por lo que no se realizó la Fase1, pasando directamente a aplicarse los tres últimos pasos de la Fase 2.

A continuación se exponen los escenarios seleccionados por su interés dentro de la arquitectura del sistema.

Escenario #1	Aumento del modelo de datos del sistema.
Atributo	Modificabilidad, Escalabilidad
Perfil	Mantenibilidad, Crecimiento
Estímulo	Adicionar tablas nuevas al modelo de datos
Respuesta	Generar el esquema de la base de datos y mapear las entidades en objetos.
Relación Atributo-Escenario	
Symfony utiliza a Propel como ORM, el cual es el encargado de generar todas las clases del acceso a datos. Para el desarrollo de esta capa se traduce a la lógica de los objetos de php 5 y Symfony	

toda la lógica relacional de la base de datos, lo cual se realiza mediante un esquema de la base de datos que es generado automáticamente por Propel y a partir de ese esquema genera las clases del modelo mapeando las entidades de la base de datos como objetos php. Propel genera dos tipos de clases del modelo, las bases y las personalizadas. Las personalizadas heredan de las bases y son las que contienen los métodos agregados por los desarrolladores ya que cada vez que se genere el modelo nuevamente se borran solamente las clases bases. Estas facilidades que brinda Propel contribuyen a que el sistema gane en mantenibilidad y escalabilidad al permitir que el crecimiento de la capa de datos no afecte grandemente la de acceso a datos y el impacto en la lógica de negocio y presentación sea mínimo.

Tabla 4 Escenario #1

Escenario #2	Migración del sistema gestor de base de datos
Atributo	Modificabilidad, Configurabilidad
Perfil	Mantenibilidad, Configuración
Estímulo	Necesidad de la migración con carácter temporal o definitivo de PostgreSQL a otro SGBD
Respuesta	Cambiar el gestor de base de datos PostgreSQL por el nuevo gestor definido
Relación Atributo-Escenario	
El estilo arquitectónico MVC utilizado en el sistema divide el modelo en dos partes, una encargada del acceso a datos y la otra de la abstracción de la base de datos la cual permite que no se vean afectado la vista ni el controlador por un cambio del sistema gestor de base de datos. La abstracción de la base de datos es gestionada por Creole que es usado por Propel, el cual es un ORM	

encargado de generar el modelo del MVC. El cambio del sistema gestor de base de datos es fácil de resolver ya que solo afecta al archivo de configuración databases.yml que se encuentra en el directorio config/ y que contiene la información mínima que se necesita para realizar peticiones a la base de datos. Como se puede observar los componentes afectados son mínimos y relacionados con la configuración del sistema por lo que se resalta la configurabilidad, la flexibilidad, modificabilidad y por tanto el mantenimiento del sistema.

Tabla 5 Escenario #2

Escenario #3	Definición de los datos a los cuales se le realizará el análisis
Atributo	Integridad, Flexibilidad, Disponibilidad
Perfil	Integridad, Flexibilidad, Desempeño
Estímulo	Definir de forma sencilla y con o sin conexión a un origen de datos la información a la cual se le realizará el análisis.
Respuesta	Creación de un modelo semántico.
Relación Atributo-Escenario	
Los datos a procesar se obtienen a partir de una consulta realizada a la base de datos del cliente, por lo que con el objetivo de simplificar el tamaño y complejidad que pueda tener se define un modelo semántico que ofrece una abstracción de la base de datos fuente que contiene únicamente las tablas de las que se extraerá la información a analizar. Esta abstracción simplifica la definición de los datos a analizar y es representada extrayendo los metadatos necesarios de la base de datos en un fichero XML, que describe su estructura. El modelo semántico mejora la disponibilidad del sistema permitiendo tener una definición de la porción de la base de datos fuente que se requiera,	

así como la integridad al no tener que estar realizando operaciones directamente sobre la base de datos evitando que se pueda dañar o perder los datos que posee. Además que el tener la metainformación de la base de datos en un XML permite apreciar la flexibilidad del sistema ya que se puede representar la base de datos independientemente del sistema gestor de base de datos que se utilice.

Tabla 6 Escenario #3

Escenario #4	Comunicación entre las capas de presentación y la lógica de negocio.
Atributo	Rendimiento
Perfil	Desempeño
Estímulo	Aumentar la velocidad de respuesta del servidor web.
Respuesta	Utilización de tecnología Ajax.
Relación Atributo-Escenario	
Ajax es una tecnología asíncrona, en el sentido de que los datos adicionales se requieren al servidor y se cargan en segundo plano sin interferir con la visualización ni el comportamiento de la página. La capa de presentación se comunica con la lógica de negocio mediante una solicitud Ajax, en la cual se utiliza el objeto XMLHttpRequest para intercambiar los datos de forma asíncrona con el servidor web. Para la transferencia de los datos solicitados se utiliza JSON o XML según corresponda. De esta forma es posible realizar cambios sobre las páginas sin necesidad de recargarlas, lo que significa aumentar la interactividad, velocidad y usabilidad del sistema. Estas facilidades que brinda Ajax permiten que el sistema gane en rendimiento mejorando su desempeño.	

Tabla 7 Escenario #4

Al ser llevado a cabo el paso 3 de la Fase 2 se observaron algunos escenarios del sistema y su relación con los atributos de calidad. Algunos no se pudieron evaluar, porque es imposible en este estado, determinar si el sistema contará con determinadas características, a la hora de prestar un servicio solicitado; los que fueron analizados son aquellos que recibieron mayor votación por los involucrados en el proceso de evaluación, presentándose en el documento los más relevantes dentro de estos.

A partir de todos los elementos que se han reflejado, se puede afirmar que la solución cumple con los atributos Flexibilidad, Rendimiento, Modificabilidad, Escalabilidad, Configurabilidad, Integridad y Disponibilidad.

Teniendo en cuenta lo expuesto anteriormente, se define de manera general y a partir de las consideraciones generadas, como resultado de la combinación de los atributos contemplados que el diseño arquitectónico evaluado de manera parcial, dentro de una etapa temprana del desarrollo, resulta adecuado.

3.8. Conclusiones

Tras el estudio de los métodos de evaluación se eligió para su utilización el ARID, debido a que la propuesta arquitectónica se hace en una fase temprana del desarrollo del sistema. El mismo fue aplicado a la solución propuesta, quedando la misma de esta forma evaluada y demostrando su cumplimiento con los atributos de calidad seleccionados.

Conclusiones Generales

Durante el desarrollo de la investigación se cumplió con el objetivo y las tareas propuestas:

1. Se realizó un estudio de los principales conceptos relacionados con la arquitectura, tales como Patrones, Frameworks, Vistas y Estilos.
2. Se realizó un estudio de las principales herramientas de minería de datos existentes en el mundo hoy en día.
3. Se seleccionó y argumentó la metodología de desarrollo: Proceso Unificado de Desarrollo (RUP) y se realizó una valoración de las principales tareas que debe llevar a cabo el rol de arquitecto según esta metodología.
4. Se definieron los estilos arquitectónicos a utilizar, los lenguajes, los IDE, los objetivos y restricciones a tener en cuenta para la realización de la aplicación, siempre contando con los requisitos de los clientes. Todas las elecciones fueron fundamentadas.
5. Se describió la arquitectura mediante las 5 vistas seleccionadas, 4 de ellas de las definidas por RUP (Caso de uso, Lógica, Implementación y Despliegue), lo que garantizó que se tuviera una visión de todos los elementos arquitectónicamente significativos para el Proceso de Desarrollo.
6. Se realizó un estudio detallado de los principales métodos de evaluación de arquitecturas software.
7. Se evaluó la arquitectura por el método seleccionado, siendo el mismo considerado el más efectivo en este caso debido a que la propuesta arquitectónica se hace en una fase temprana del desarrollo del sistema.
8. Se logró con el desarrollo del presente trabajo, diseñar una arquitectura que cumple con los requisitos, especificados por el cliente, del sistema al cual da solución informática, por tanto se cumplió con el objetivo de diseñar una arquitectura robusta, flexible y escalable.

Recomendaciones

Se recomienda:

- El refinamiento constante de la arquitectura propuesta, durante el ciclo de desarrollo.
- Realizar una evaluación de los costes de la tecnología utilizada, aunque la misma en su totalidad es libre, es necesario evaluar los costes del hardware que la soportan.
- Continuar profundizando en la validación de la arquitectura propuesta. En este sentido sería bueno aplicar el método ATAM, una vez que el sistema se encuentre en una fase más avanzada en su desarrollo, puesto que este brindaría un resultado completo y detallado de aquellos riesgos que afectan la arquitectura.

Referencias y Bibliografía

1. **FAYYAD, U.M., y otros.** *Advances in knowledge and data mining.* Cambridge (Massachussets) : AAAI/MIT Press, 1996.

2. **MOLINA, L.C. y RIBEIRO, S.** Descubrimiento conocimiento para el mejoramiento bovino usando técnicas de data mining. *Descubrimiento conocimiento para el mejoramiento bovino usando técnicas de data mining.* Barcelona : s.n., 2001, págs. 123-130. .

3. **Reutemann, P., Pfahringer, B. y Frank, E.** *Proper: A Toolbox for Learning from Relational Data with Propositional and Multi-Instance Learners.* 17th Australian Joint Conference on Artificial Intelligence (AI2004) : Springer-Verlag., 2004.

4. *Introducción a la Arquitectura de Software.* **Reynoso, Carlos Billy.** UNIVERSIDAD DE BUENOS AIRES : s.n., Marzo de 2004, Vol. Versión 1.0.

5. **Shaw, Mary y Garlan, David.** *Software Architecture, Perspectives on an emerging discipline.* s.l. : Prentice Hall, 1996.

6. **Jacobson, Ivar, Booch, Grady y Rumbaugh, James.** *El Proceso Unificado de Desarrollo de Software.* Madrid : Pearson Education.S.A, 2000. 84-7829-036-2.

7. **Kruchten, Philippe.** Architectural Blueprints--The 4+1 View Model of Software Architecture. s.l. : IEEE Software, 1995, págs. 42-50.

8. **Booch, Grady.** *Object-Oriented Design with Applications.* s.l. : The Benjamin/Cummings Publishing Company, 1991.

9. **Abdurazik, Aynur.** *Suitability of the UML as an Architecture Description Language with applications to testing.* 2000. Reporte ISE-TR-00-01.

10. **Buschmann, Frank, Henney, Kevlin y Schmidt, Douglas C.** *Patterns Oriented Software Achitecture: A Pattern Language for Distributed Computing, Volumen 4.* s.l. : John Wiley & Sons,Ltd, 2007.

11. **Pressman, Roger S.** Capítulo 14. Diseño Arquitectónico. *Ingeniería del Software. Un enfoque práctico.* Madrid : Mc Graw Hill, 2001, págs. 237-258.

12. **Reynoso, Carlos y Kicillof, Nicolás.** *Estilos y Patrones en la Estrategia de Arquitectura de Microsoft.* UNIVERSIDAD DE BUENOS AIRES : s.n., 2004. Versión 1.0.

13. *A Component- and Message-Based Architectural Style for GUI Software.* **Richard Taylor, Nenad Medvidovic, Kenneth Anderson, James Whitehead, Jason Robbins, Kari Nies, Peyman.** Seattle : ACM Press, 1995. 17th International Conference on Software Engineering. págs. 295-304.

14. **Evdemon, John.** Principios de diseño de servicios: patrones y antipatrones de servicios. *www.microsoft.com.* [En línea] Agosto de 2005. [Citado el: 10 de Enero de 2009.] http://www.microsoft.com/spanish/msdn/articulos/archivo/121205/voices/SOADesign.mspx.

15. **Schneider, Jean Guy.** *Components, Scripts, and Glue:A conceptual Framework for Software Composition.Tesis Doctoral.* Institut fur Informatik und angewandte Mathematik , Universitat Bern : s.n., 1999.

16. *Lenguajes de descripción arquitectónica de Sofware (ADL).* **Reynoso, Carlos y Kicillof, Nicolás.** Universidad de Buenos Aires : s.n., Marzo de 2004.

17. **Allen y J., Robert.** *A Formal Approach to Software Architecture.* School of Computer Science, Carnegie Mellon University,Pittsburgh : s.n., 1997. CMU-CS-97-144.

18. **IEEE-1471.** *Recommended Practice for. Architectura lDescription of Software-Intensive Systems.* 2000.

19. **IBM Corp.** *Ayuda del Rationa.* s.l. : IBM Corporation 1987, 2006.

20. **Zend Technologies Ltd.** www.zend.com. [En línea] [Citado el: 25 de marzo de 2009.] http://www.zend.com/en/products/studio/.

21. **The Eclipse Foundation.** www.eclipse.org. [En línea] [Citado el: 25 de 3 de 2009.] http://www.eclipse.org/legal/epl-v10.html.

22. **Potencier, Fabien y Zaninotto, François.** *Symfony la guía definitiva.* 2008.

23. **Dávila Mauricio, Martín Germán,Diego Crutas, Andrés García.** Evaluación de Arquitecturas de Software. *Facultad de Ingeniería Gestion de Software.*

24. **Kazman, R., Clements, P., Klein, M.** *Evaluating Software Architectures.Methods and case studies.* 2001.

25. **IEEE.** IEEE standard glossary of software engineering terminology. *IEEE Std 610.12.* 12 de Diciembre de 1990.

26. **Bosch, J.** *Design & Use of Software Architectures.* 2000.

27. **Kazman, R., Clements, P., Klein, M.** *Evaluating Software Architectures.* 2001.

28. **Clements, Paul y colectivo de autores.** *Documenting Software Architectures: Views and Beyond.* s.l. : Addison Wesley, 2002. 0-201-70372-6.

29. **Cuesta, Carlos E.** *Arquitentura de Software Dinamica Basada en Reflexion. Tesis Doctoral.* Departamento de Informatica, Universidad de Valladolid : s.n., 2002.

30. **Ferri., César.** Mi página de Weka. [En línea] [Citado el: 15 de febrero de 2009.] http://users.dsic.upv.es/~cferri/weka/.

31. **Fowler, Martin y colectivo de autores.** *Patterns of Enterprise Application Architecture.* s.l. : Addison Wesley, 2002. 0-321-12742-0.

32. **Jaime G. Orjuela Parra .** sophia.javeriana.edu.co. *sophia.javeriana.edu.co.* [En línea] [Citado el: 10 de marzo de 2009.] http://sophia.javeriana.edu.co/~cbustaca/Arquitectura%20Software/Clases/Ensayos_2008/Jaime_Orjuela_P17.pdf.

33. **Orallo., José Hernández.** [En línea] 2002. [Citado el: 2 de marzo de 2009.] http://users.dsic.upv.es/~jorallo/master/Laboratori-KDD.pdf.

34. **Schmidt, Douglas, y otros.** *Pattern-Oriented Software Architecture, Patterns for Concurrent and Networked Objects, Volume 2.* s.l. : John Wiley & Sons, 2000. 0471606952.

35. **Ramírez, César Ferri, Orallo, José Hernández y Quintana, María José Ramírez.** *"INTRODUCCIÓN A LA MINERÍA DE DATOS".* s.l. : Pearson, 2004.

36. **ORACLE.** ORACLE.com. [En línea] [Citado el: 26 de febrero de 2009.] http://www.oracle.com/global/es/products/solutions/business_intelligence/olap.html.

37. **SPSS Inc.** SPSS.com. [En línea] 2009. [Citado el: 11 de febrero de 2009.] http://www.spss.com/es/clementine/.

38. **Flanagan, David y Ferguson, Paula.** *JavaScript: The Definitive Guide, 4th Edition.* s.l. : O'Reilly & Associates, 2002. ISBN 0-596-00048-0 .

39. **Pérez, Javier Eguíluz.** Introducción a JavaScript. *www.librosweb.es.* [En línea] 7 de junio de 2008. [Citado el: 20 de febrero de 2009.] http://www.librosweb.es/javascript/.

40. **The Apache Software Foundation.** httpd.apache.org. [En línea] [Citado el: 25 de febrero de 2009.] http://httpd.apache.org/docs/2.2/mod/mod_rewrite.html.

41. Open_jACOB. [En línea] [Citado el: 15 de febrero de 2009.] http://www.openjacob.org/index.html.

42. **R. Kazman, M. Klein, P. Clements.** Evaluating Software Architectures for Real-Time. 13 de Julio de 1999.

Glosario de Términos

- **Hardware (soporte físico):** se utiliza generalmente para describir los artefactos físicos de una tecnología. Es el conjunto de elementos físicos que componen una computadora Disco Duro, CD-ROM, etc.
- **Software (soporte lógico):** los componentes intangibles de una computadora, es decir, al conjunto de programas y procedimientos necesarios para hacer posible la realización de una tarea específica.
- **IEEE:** corresponde a las siglas de The **I**nstitute of **E**lectrical and **E**lectronics **E**ngineers, el Instituto de Ingenieros Eléctricos y Electrónicos, una asociación técnico-profesional mundial dedicada a la estandarización, entre otras cosas. Es la mayor asociación internacional sin fines de lucro formada por profesionales de las nuevas tecnologías, como ingenieros eléctricos, ingenieros en electrónica, científicos de la computación e ingenieros en telecomunicación.
- **HTML:** siglas de HyperText Markup Language (*Lenguaje de Marcado de Hipertexto*), es el lenguaje de marcado predominante para la construcción de páginas web. Es usado para describir la estructura y el contenido en forma de texto, así como para complementar el texto con objetos tales como imágenes.
- **Herramientas CASE:** (Computer Aided Software Engineering, Ingeniería de Software Asistida por Ordenador) son diversas aplicaciones informáticas destinadas a aumentar la productividad en el desarrollo de software reduciendo el coste de las mismas en términos de tiempo y de dinero.
- **Plugin:** Un plugin o componente enchufable (o plug-in -en inglés "enchufar"-, también conocido como addin, add-in, addon o add-on) es una aplicación informática que interactúa con otra

aplicación para aportarle una función o utilidad específica, generalmente muy específica, como por ejemplo servir como driver (controlador) en una aplicación, para hacer así funcionar un dispositivo en otro programa. Ésta aplicación adicional es ejecutada por la aplicación principal. Los plugins típicos tienen la función de reproducir determinados formatos de gráficos, reproducir datos multimedia, codificar/decodificar emails, filtrar imágenes de programas gráficos.

- **XML:** sigla en inglés de E**x**tensible **M**arkup **L**anguage («lenguaje de marcas extensible»), es un metalenguaje extensible de etiquetas desarrollado por el World Wide Web Consortium (W3C). Es una simplificación y adaptación del SGML y permite definir la gramática de lenguajes específicos (de la misma manera que HTML es a su vez un lenguaje definido por SGML). Por lo tanto XML no es realmente un lenguaje en particular, sino una manera de definir lenguajes para diferentes necesidades.
- **RAM (Random Access Memory):** memoria de acceso aleatorio ó memoria de acceso directo.
- **TCP/IP:** es un conjunto de protocolos de red que permiten la transmisión de datos entre redes de computadoras.
- **Servidor**: Una aplicación informática o programa que realiza algunas tareas en beneficio de otras aplicaciones llamadas clientes.
- **Cliente/Servidor**: Esta arquitectura consiste básicamente en que un programa (cliente informático) realiza peticiones a otro programa (servidor) que les da respuesta.
- **ASP** (**A**ctive **S**erver **P**ages): es una tecnología del lado servidor de Microsoft para páginas web generadas dinámicamente, que ha sido comercializada como un anexo a Internet Information Server (IIS).

Anexos

Anexo 1 Distribución de los patrones de POSA

Estructura	Sistemas Distribuidos	Sistemas Interactivos	Sistemas Adaptables
• Layers • Pipes & Filtres • Blackboard	• Broker	• Model-View-Controller • Presentation-Abstraction-Control	• Microkernel • Reflection

Anexo 2 Distribución de los patrones de PEEA

Dominio Lógico	Mapeo Objeto / Relacional	Presentación Web
• Transaction Script • Domain Model • Table Module • Service Layer	➢ *Patrones Arquitecturales de Fuentes de Datos* • Table Data Gateway • Row Data Gateway • Active Record • Data Mapper ➢ *Patrones de comportamiento Objeto /Relacionales* • Unit of Work • Identity Map • Lazy Load ➢ *Patrones estructurales*	• Model View Controller • Page Controller • Front Controller • Template View • Transform View • Two Step View • Application Controller

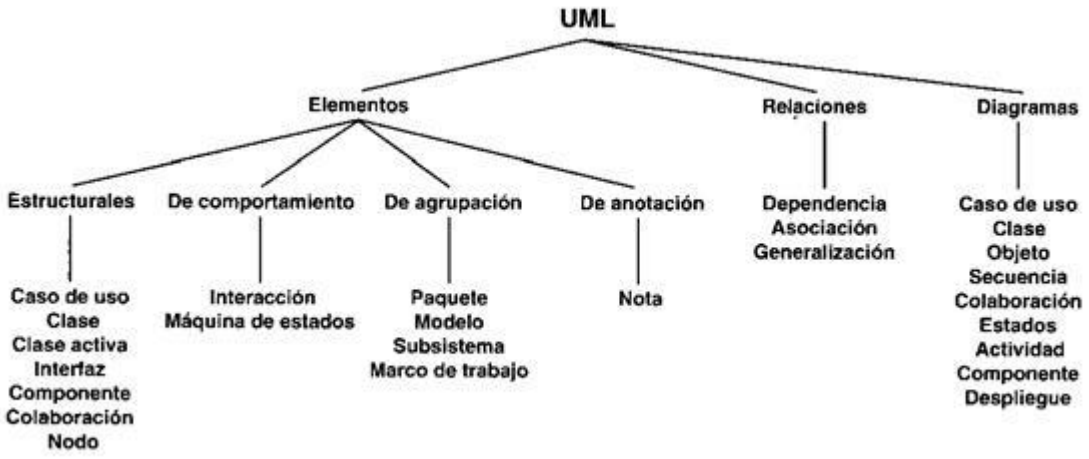
	Objeto / Relacionales • Foreign Key Mapping • Identity Field • Association Table Mapping • Dependent Mapping • Embedded Value • Serialized LOB • Single Table Inheritance • Class Table Inheritance • Concrete Table Inheritance • Inheritance Mappers ➢ *Patrones Objeto/Relacional para el mapeo de metadatos* • Metadata Mapping Query Object • Repository	

Anexo 3 Clasificación de patrones de arquitectura de PEEA.

Distribución	Concurrencia fuera de línea	Estado de Sesión	Base
• Remote Facade • Data Transfer Object	• Optimistic Offline Lock • Pesimistic Offline Lock • Coarse-Grained Lock • Implicit Lock	• Client Session State • Server Session State • Database Session State	• Gateway • Mapper • Layer Supertype • Separated Interface • Registry • Value Object

			• Money • Special Case • Plugin • Service Stub • Record Set

Anexo 4 Componentes UML



Anexo 5 Estructura de la raíz del proyecto de Symfony.

Proyecto	Aplicación	Módulo	Web

apps/
  frontend/
  backend/
batch/
cache/
config/
data/
  sql/
doc/
lib/
  model/
log/
plugins/
test/
  unit/
  functional/
web/
  css/
  images/
  js/
  uploads/
	apps/
	  [nombre aplicacion]/
	  config/
	  i18n/
	  lib/
	  modules/
	  templates/
	  layout.php
	  error.php
	  error.txt
		apps/
		  [nombre aplicacion]/
		  modules/
		  [nombre modulo]/
		  actions/
		  actions.class.php
		  config/
		  lib/
		  templates/
		  indexSuccess.php
		  validate/
			web/
			  css/
			  images/
			  js/
			  uploads/

Directorio Proyecto	**Descripción**
apps/	Contiene un directorio por cada aplicación del proyecto.
batch/	Contiene los scripts de PHP que se ejecutan mediante la línea de comandos o mediante la programación de tareas para realizar procesos en lotes (*batch processes*)

cache/	Contiene la versión cacheada de la configuración y (si está activada) la versión cacheada de las acciones y plantillas del proyecto. El mecanismo de cache (que se explica en el Capítulo 12) utiliza los archivos de este directorio para acelerar la respuesta a las peticiones web. Cada aplicación contiene un subdirectorio que guarda todos los archivos PHP y HTML preprocesados
config/	Almacena la configuración general del proyecto
data/	En este directorio se almacenan los archivos relacionados con los datos, como por ejemplo el esquema de una base de datos, el archivo que contiene las instrucciones SQL para crear las tablas e incluso un archivo de bases de datos de SQLite
doc/	Contiene la documentación del proyecto, formada por tus propios documentos y por la documentación generada por PHPdoc
lib/	Almacena las clases y librerías externas. Se suele guardar todo el código común a todas las aplicaciones del proyecto. El subdirectorio model/ guarda el modelo de objetos del proyecto
log/	Guarda todos los archivos de log generados por Symfony. También se puede utilizar para guardar los logs del servidor web, de la base de datos o de cualquier otro componente del proyecto. Symfony crea un archivo de log por cada aplicación y por cada entorno
plugins/	Almacena los plugins instalados en la aplicación
test/	Contiene las pruebas unitarias y funcionales escritas en PHP y compatibles con el framework de pruebas de Symfony. Cuando se crea un proyecto, Symfony crea algunos pruebas básicas
web/	La raíz del servidor web. Los únicos archivos accesibles desde Internet son los que se encuentran en este directorio

Directorio Aplicación	Descripción
config/	Contiene un montón de archivos de configuración creados con YAML. Aquí se almacena la mayor parte de la configuración de la aplicación, salvo los parámetros propios del framework. También es posible redefinir en este directorio los parámetros por defecto si es necesario.
i18n/	Contiene todos los archivos utilizados para la internacionalización de la aplicación, sobre todo los archivos que traducen la interfaz. La internacionalización también se puede realizar con una base de datos, en cuyo caso este directorio no se utilizaría
lib/	Contiene las clases y librerías utilizadas exclusivamente por la aplicación
modules/	Almacena los módulos que definen las características de la aplicación
templates/	Contiene las plantillas globales de la aplicación, es decir, las que utilizan todos los módulos. Por defecto contiene un archivo llamado layout.php, que es el layout principal con el que se muestran las plantillas de los módulos

Directorio Módulo	Descripción
actions/	Normalmente contiene un único archivo llamado actions.class.php y que corresponde a la clase que almacena todas las acciones del módulo. También es posible crear un archivo diferente para cada acción del módulo
config/	Puede contener archivos de configuración adicionales con parámetros exclusivos del módulo

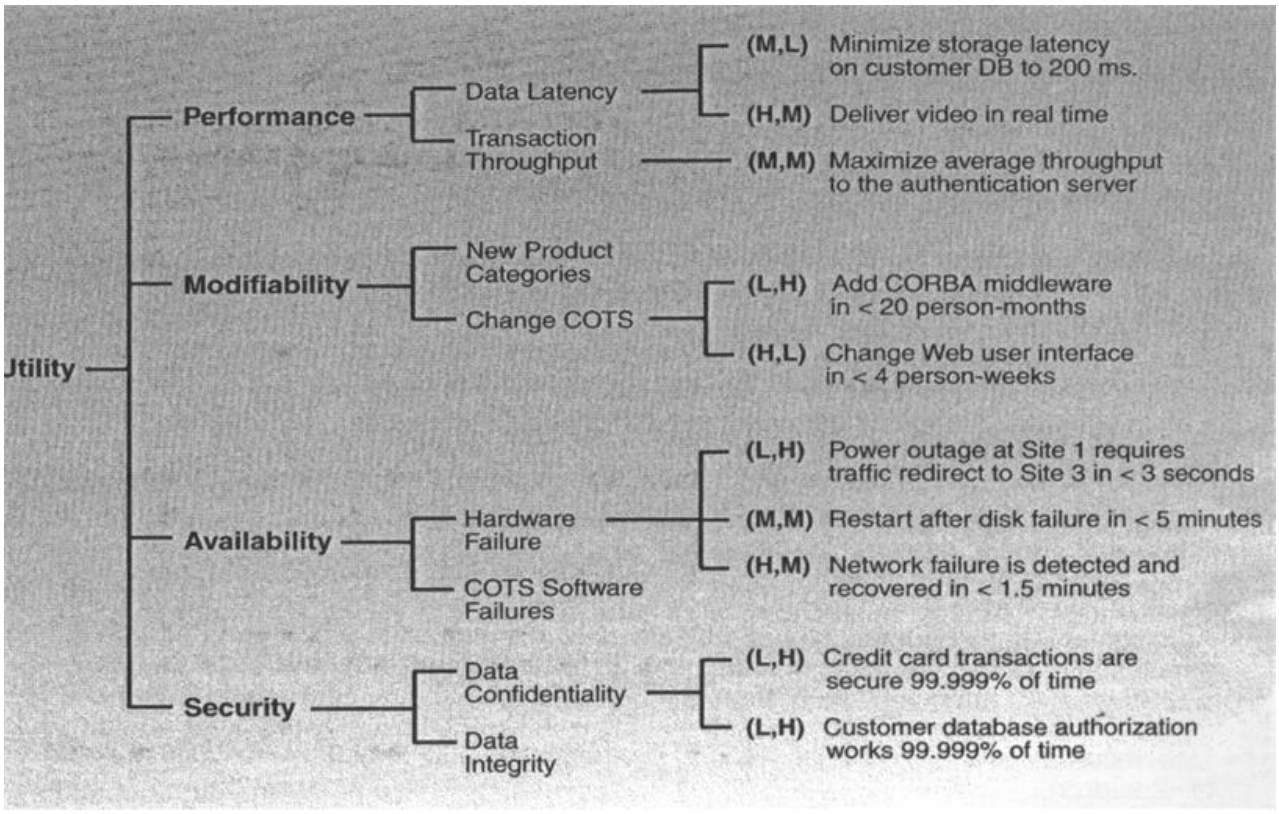
lib/	Almacena las clases y librerías utilizadas exclusivamente por el módulo
templates/	Contiene las plantillas correspondientes a las acciones del módulo. Cuando se crea un nuevo módulo, automáticamente se crea la plantilla llamada indexSuccess.php
validate/	Contiene archivos de configuración relacionados con la validación de formularios

Directorio Web	**Descripción**
css/	Contiene los archivos de hojas de estilo creados con CSS (archivos con extensión .css)
images/	Contiene las imágenes del sitio con formato .jpg, .png o .gif
js/	Contiene los archivos de JavaScript con extensión .js
uploads/	Se almacenan los archivos subidos por los usuarios.

Anexo 6 Atributos de Calidad en la arquitectura

IEEE Std 1061	ISO Std 9126	MITRE Guide to Total Software Quality Control	
Eficiencia	Funcionalidad	Eficiencia	Integridad
Funcionalidad	Confiabilidad	Confiabilidad	Supervivenciabilidad
Mantenibilidad	Usabilidad	Usabilidad	Corretitud
Portabilidad	Eficiencia	Mantenibilidad	Verificabilidad
Confiabilidad	Mantenibilidad	Expandibilidad	Flexibilidad
Usabilidad	Portabilidad	Interoperabilidad	Portabilidad
		Reusabilidad	

Anexo 7 Árbol de utilidad



Anexo 8 Propuesta arquitectónica (análisis)

Análisis de una Propuesta Arquitectónica				
Escenario #: 1	**Escenario** Versionar un proceso con 300 elementos en menos de 10 segundos			
Atributo	Performance – Throughput			
Entorno	Proceso con 300 elementos			
Estímulo	Se invoca el versionado			
Respuesta	Versiona el proceso en menos de 10 segundos			
Decisión Arquitectónica	**Sensitivity**	**Tradeoff**	**Riesgo**	**No riesgo**
Cada subsistema accede directamente a la BD	S1	T1,T2	R1,R2	
Un subsistema que encapsule el acceso a datos (p.e.: uso de Hibernate)	S2	T3		
Razonamiento	S1. Disminuye la confiabilidad. S2. Aumenta la confiabilidad. R1. Aumenta el riesgo que los datos queden inconsistentes ante una eventual falla. R2. Aumento de dependencia entre la aplicación y la BD. T1. Mejora la performance, pero disminuye la maintainability. T2. Mejora la performance, pero disminuye la reusability. T3. Aumenta la reusability, pero empeora la performance.			

Anexo 9 Pasos del ARID

Fase 1: Actividades Previas	
1. Identificación de los encargados de la revisión	Los encargados de la revisión son los ingenieros de software que se espera que usen el diseño, y todos los involucrados en el diseño. En este punto, converge el concepto de *encargado de revisión* de ADR e *involucrado* del ATAM.
2. Preparar el informe De diseño	El diseñador prepara un informe que explica el diseño. Se incluyen ejemplos del uso del mismo para la resolución de problemas reales. Esto permite al facilitador anticipar el tipo de preguntas posibles, así como identificar áreas en las que la presentación puede ser mejorada.
3. Preparar los escenarios base	El diseñador y el facilitador preparan un conjunto de escenarios base. De forma similar a los escenarios del ATAM y el SAAM, se diseñan para ilustrar el concepto de escenario, que pueden o no ser utilizados para efectos de la evaluación.
4. Preparar los materiales	Se reproducen los materiales preparados para ser presentados en la segunda fase. Se establece la reunión, y los involucrados son invitados.
Fase 2: Revisión	
5. Presentación del ARID	Se explica los pasos del ARID a los participantes.
6. Presentación del diseño	El líder del equipo de diseño realiza una presentación, con ejemplos incluidos. Se propone evitar preguntas que conciernen a la implementación o argumentación, así como alternativas de diseño. El objetivo es verificar que el diseño es conveniente.
7. Lluvia de ideas y establecimiento de prioridad de escenarios	Se establece una sesión para la lluvia de ideas sobre los escenarios y el establecimiento de prioridad de escenarios. Los involucrados proponen escenarios a ser usados en el diseño para resolver problemas que esperan encontrar. Luego, los escenarios son sometidos a votación, y se utilizan los que resultan ganadores para hacer pruebas sobre el diseño.
8. Aplicación de los escenarios	Comenzando con el escenario que contó con más votos, el facilitador solicita pseudo-código que utiliza el diseño para proveer el servicio, y el diseñador no debe ayudar en esta tarea. Este paso continúa hasta que ocurra alguno de los siguientes eventos: ✓ Se agota el tiempo destinado a la revisión ✓ Se han estudiado los escenarios de más alta prioridad ✓ El grupo se siente satisfecho con la conclusión alcanzada. Puede suceder que el diseño presentado sea conveniente, con la exitosa aplicación de los escenarios, o por el contrario, no conveniente, cuando el grupo encuentra problemas o deficiencias.
9. Resumen	Al final, el facilitador recuenta la lista de puntos tratados, pide opiniones de los participantes sobre la eficiencia del ejercicio de revisión, y agradece por su participación.