Universidad de las Ciencias Informáticas

Facultad 6

Título: “Definición de la arquitectura de software para el Generador Dinámico de Reportes en su versión 2.0.”

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

Autora

Vania Elena Yanes León.

Tutor

Ing. Luis Ernesto Saballo López.

Co-Tutor

Ing. Aldis Joan Abreu Medina.

La Habana, Junio de 2011
“Año 53 de la Revolución”

"El futuro tiene muchos nombres. Para los débiles es lo inalcanzable. Para los temerosos, lo desconocido. Para los valientes es la oportunidad."

Víctor Hugo

DECLARACIÓN DE AUTORÍA

Declaro que soy el único autor de este trabajo y autorizo a la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio.

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

Nombre Autor(a)

Vania Elena Yanes León

_____

Nombre del Tutor

Ing. Luis Ernesto Saballo

_____

Nombre del Tutor

Ing. Aldis Joan Abreu Medina

_____

DATOS DE CONTACTO

Nombre: Luis Ernesto

Apellidos: Saballo López

Título universitario: Ingeniero en Ciencias Informáticas

Año de graduado: 2007

Correo: lsaballo@uci.cu

Nombre: Aldis Joan

Apellidos: Abreu Medina

Título universitario: Ingeniero en Ciencias Informáticas

Año de graduado: 2009

Correo: ajabreu@uci.cu

AGRADECIMIENTOS

A la Revolución, a Fidel y a Raúl por haberme permitido estudiar en esta universidad del futuro y darme la oportunidad de hacer realidad un sueño.
A mi mami y a mi papi por darme todo el amor, la fuerza y el apoyo para salir adelante y por enseñarme que para lograr algo solo basta con proponérselo y esforzarse, a mi hermanito por confiar siempre en mi.
A mi familia por esperar lo mejor de mi y porque cada uno de ellos puso un granito de arena para que pudiera llegar a ser quien soy, en especial a mi tía Denia, mi tío José Alfredo, mi primita Betsy y mi abuelito José.
A mis tutores Luis Ernesto y Aldis por estar a mi lado apoyándome.
A mi tribunal y a mi oponente Jorge Bedoya por corregir cada detalle y estar dispuestos a ayudarme.
A todos mis compañeros del proyecto GDR por ayudarme y aclarar cada una de mis dudas en especial: Yasmany, Raidel, Javier, Marleisy y Juan José, sin todos ustedes no hubiera sido posible.
A las personas que me han ayudado a salir adelante en los momentos fuertes en especial a mi amigo Alberto Garnache, a Yusdenis, a Diana y a Ileane una personita que quiero mucho y considero una gran amiga.
A todos mis compañeros que estuvieron conmigo durante estos 5 años de carrera en los que pasamos buenos y malos momentos pero siempre juntos y contentos, en especial quiero mencionar a: Raiza, Elisa, Taimi, Leidiana, Alberto Hernández, Michael, Flavio, Yanet, Beatriz Lara…., a mis compañeras de apartamento por ser mi pequeña familia aquí en la universidad, a todos muchas gracias por soportarme y por cuidar siempre de mi.
A mis amistades de la infancia en especial: Irene, Neyilí, Annalie, Verónica y Yaikenia.
A todos los profesores que ayudaron en mi formación tanto profesional como en mi vida personal.
En fin a todas las personas que de una forma u otra hicieron posible este resultado.

DEDICATORIA

A mi mamá y a mi papá, por ser la razón de mi vida y el motivo de mí existir.

A mi hermano que lo adoro y espero ser un buen ejemplo para él como hermana mayor.

A mi familia por esperar siempre lo mejor de mí.

RESUMEN

Los generadores de reportes son herramientas complementarias de los sistemas de información. Utilizan un lenguaje transparente para el usuario por medio del cual se realizan consultas a la base de datos para obtener información de ella en forma de reporte. El presente trabajo describe la arquitectura de software para el Generador Dinámico de Reportes en su versión 2.0 para la web desarrollado en PHP.

La arquitectura definida provee los elementos necesarios para el desarrollo de dicho software. Su éxito viene dado por las decisiones arquitectónicas tomadas, por la definición de la plataforma tecnológica, de los lineamientos y mecanismos arquitectónicos, así como la descripción de la arquitectura por medio de las vistas de la arquitectura.

La selección de las herramientas informáticas y las tecnologías para el desarrollo del sistema se llevo a cabo teniendo en cuenta la situación del país con respecto al acceso a las herramientas propietarias y se decidió utilizar aquellas que están bajo licencia libre.

Se realizó una evaluación de la arquitectura definida aplicando el método de evaluación de la arquitectura Architecture Trade-off Analisys Method (ATAM) conjuntamente con el procedimiento Árbol de Utilidad y la técnica Basada en Escenarios obteniendo resultados satisfactorios.

El resultado fundamental se centra en la obtención de una arquitectura de software para este sistema que potencie atributos de calidad, obteniendo mayor flexibilidad, escalabilidad y robustez. Todo esto permite que el sistema pueda ser utilizado a nivel empresarial o embebido en aplicaciones personalizadas para cubrir completamente el ciclo de vida de los reportes de forma dinámica e intuitiva.

Palabras clave: arquitectura de software, generadores de reportes, plataforma tecnológica, PHP.

TABLA DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

INTRODUCCIÓN

Desde hace algunos años las empresas o instituciones han reconocido la importancia de administrar sus principales recursos, dentro de ellos la información es un elemento vital pues influye de manera directa en la forma en que estas operan. Por lo cual los entes encargados de la toma de decisiones han comenzado a comprender que la información no es sólo un subproducto de la conducción empresarial, sino que a la vez alimenta a los negocios y puede ser uno de los tantos factores críticos para la determinación del éxito o fracaso de éstos y con frecuencia son utilizados los Sistemas de Información para la administración y distribución de la información.

Los Sistemas de Información (SI) han cambiado la forma en que operan las organizaciones actuales. A través de su uso se logran importantes mejoras, pues automatizan los procesos operativos, suministran una plataforma de información necesaria para la toma de decisiones y lo más importante, su implantación logra ventajas competitivas. Un gran número de organizaciones cuentan con varios sistemas dispersos pero cada uno posee sus propias fuentes de datos y mecanismos de representación, lo cual provoca que el mantenimiento de la información actualizada a través de los departamentos y unidades de negocios sea extremadamente difícil.

En función de obtener utilidades y crear los valores propios de la organización los procesos de descripción, análisis y representación de la información, así como las nuevas tecnologías asociadas a ellos, adquieren un sentido trascendente pues más que simples medios para la obtención de resultados debe considerárseles como herramientas que contribuyen al desempeño, aprendizaje individual y colectivo, así como la construcción positiva de la empresa. Los responsables de utilizar la información necesitan almacenar, acceder, procesar y visualizar datos de negocio que pueden estar distribuidos tanto dentro de la organización como fuera de ella.

Con este fin, existen en el mundo diferentes herramientas que viabilizan el proceso de generación de reportes que tienen definido un mecanismo para realizar consultas a la base de datos y obtener información de ella a través de reportes. En el caso específico de Cuba se ha podido constatar que existen empresas que generan reportes utilizando disímiles formas y procesos pero ninguna es capaz de seguir el ciclo de vida de cada reporte, incluso algunas no utilizan motores para dicho proceso simplemente se encargan de generarlos para obtener información, lo que provoca mayor dificultad en el manejo y generación de datos.

En el Centro de Tecnología de Gestión de Datos (DATEC) de la Universidad de las Ciencias Informáticas (UCI) se desarrolla el sistema Generador Dinámico de Reportes, con el fin de recuperar información contenida en una base de datos o cualquier otra fuente de donde se requiera extraer

información, diseñar reportes, visualizarlos en diferentes formatos (EXCEL, HTML, PDF). Dicho sistema es utilizado en el país en la Oficina Nacional de Estadísticas (ONE) y en algunos países dentro de ellos Venezuela. Está compuesto por un conjunto de módulos que brindan funcionalidades para el trabajo con los reportes brindándoles soporte durante todo el ciclo de vida. Pero este sistema aun carece de funcionalidades que son de suma importancia para su funcionamiento y la tecnología con la que fue desarrollado en comparación con otras que han surgido se encuentra en desventaja porque resulta muy complicado hacerle modificaciones para nuevas prestaciones. De esta forma el sistema pierde cualidades y afecta tanto a los usuarios como al producto llegando a impedir su comercialización.

Con vistas a minimizar estos inconvenientes se hace necesario desarrollar una nueva versión (versión 2.0) del Generador Dinámico de Reportes siendo lo suficientemente genérico como para ser utilizado en cualquier otra institución que requiera los servicios de un sistema de este tipo teniendo como soporte una arquitectura robusta, flexible y escalable. La arquitectura de los sistemas, al ser la plataforma base de lo que se quiere construir, constituye el pilar fundamental de los sistemas informáticos. De las decisiones arquitectónicas correctas que se tomen en el proceso de definición del producto depende en gran medida el desempeño posterior del mismo.

La definición adecuada de la arquitectura de software posibilita que los sistemas sean mantenibles, extensibles, usables, tanto para desarrolladores como usuarios finales. Un sistema que no cuente con una estructura que potencie su propia evolución queda obsoleto.

La realización del presente trabajo va encaminada a mejorar la problemática relacionada con la definición de una arquitectura estable para la versión 2.0 del Generador Dinámico de Reportes, por lo que se plantea como problema científico: ¿Cómo darle respuesta a las necesidades arquitectónicas del Generador Dinámico de Reportes en su versión 2.0?

La investigación tiene como objeto estudio la arquitectura de software, enmarcado en el campo de acción la arquitectura de software en sistemas dinámicos para la generación de reportes.

El objetivo general de este trabajo es: definir la arquitectura de software para el sistema Generador Dinámico de Reportes V (2.0), que permita la generación y administración de informes de forma dinámica.

En correspondencia con ello, se plantean como objetivos específicos:

- Diseñar la arquitectura de software para el sistema Generador Dinámico de Reportes V (2.0).
- Describir la arquitectura de software para el sistema Generador Dinámico de Reportes V (2.0).

- Evaluar la arquitectura de software diseñada para el sistema Generador Dinámico de Reportes V (2.0).

Para el cumplimiento de estos objetivos se desglosaron las siguientes tareas:

1 Sistematización del marco conceptual referencial sobre arquitecturas para sistemas generadores de reportes.

2 Estudio y selección de los métodos de evaluación basados en la arquitectura.

3 Selección de las herramientas y metodología a usar en el desarrollo del sistema.

4 Identificación y fundamentación de los estilos arquitectónicos y patrones a utilizar en la solución.

5 Diseño de la propuesta arquitectónica que cumpla con los requerimientos del sistema.

6 Representación del diseño arquitectónico.

7 Evaluación de la arquitectura definida aplicando el método seleccionado.

El presente documento se divide en varios capítulos:

El **Capítulo 1 fundamentación teórica**, donde se analizan los principales conceptos asociados a la arquitectura de software, así como lo referente a los estilos arquitectónicos, patrones y lenguaje de descripción de arquitectura.

El **Capítulo 2 descripción arquitectónica**, en este capítulo se hace una propuesta de solución al problema planteado en el diseño teórico de la investigación incluyendo en ella los objetivos y restricciones arquitectónicas. Se realiza la descripción de la arquitectura del sistema a través de las vistas de la arquitectura.

El **Capítulo 3 evaluación de la arquitectura**, se procede a la evaluación de la propuesta arquitectónica diseñada utilizando los métodos basados en escenarios y simulación.

CAPÍTULO 1: Fundamentación Teórica

Introducción

En el presente capítulo se platean los principales conceptos relacionados con el objeto de estudio que son de vital importancia para entender posteriormente la solución propuesta. Se abordarán referentes teóricos de las diferentes herramientas que existen para la generación de reportes, sus características y un análisis crítico de las mismas a través de comparación tomando en cuenta los aspectos más relevantes. Se analizará la Arquitectura de Software como disciplina dentro del proceso de desarrollo de software partiendo del estudio de sus principales concepciones, directrices de los estilos y patrones arquitectónicos, así como los lenguajes de descripción de la arquitectura. Se establecen las herramientas, metodología y lenguajes a utilizar argumentando el por qué de su elección.

1.1 **La Arquitectura de Software**

La Arquitectura del Software (AS) conjuntamente con los patrones y los métodos ortodoxos conforman la triada de grandes temas de la ingeniería de software, es el diseño de más alto nivel de la estructura de un sistema, es el resultado de ensamblar elementos arquitectónicos de forma adecuada para satisfacer la mayor funcionalidad y requerimientos para el desempeño de un sistema, aportando una visión abstracta de alto nivel, posponiendo el detalle de cada uno de los módulos definidos a pasos posteriores del diseño. El objetivo principal de la AS es aportar elementos que ayuden a la toma de decisiones y al mismo tiempo proporcionar conceptos y un lenguaje común que permitan la comunicación entre los equipos que participen en un proyecto. Para que la arquitectura se convierta en una herramienta útil dentro del desarrollo y mantenimiento de los sistemas de software es necesario que se cuente con una manera precisa de representarla.

1.1.1 **Definición de Arquitectura del Software**

El concepto de Arquitectura de Software involucra los aspectos estáticos y dinámicos más significativos del sistema. Existen muchas definiciones de Arquitectura del Software y no parece que ninguna de ellas haya sido totalmente aceptada o respaldada por los arquitectos de hoy.

La arquitectura de software es una primera aproximación al diseño de alto nivel donde se identifican los principales componentes, su interacción y sus dependencias. Una definición clásica del término la provee RUP (*Rational Unified Process)* (1):

“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" [1]

Una definición reconocida es la de Clements (2):

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

En una definición tal vez demasiado amplia, David Garlan (3) establece que:

La AS constituye un puente entre el requerimiento y el código, ocupando el lugar que en los gráficos antiguos se reservaba para el diseño.

De acuerdo al SEI (*Software Engineering Institute*), la Arquitectura de Software se refiere a "las estructuras de un sistema, compuestas de elementos con propiedades visibles de forma externa y las relaciones que existen entre ellos." [2]

La definición "oficial" de AS se ha acordado que sea la que brinda el documento de IEEE Std 1471-2000, adoptada también por Microsoft, que reza así:

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

Aunque hoy existen múltiples conceptos de arquitectura de software en diferentes planos, esta disciplina se enuncia alrededor de algunas definiciones y principios importantes los cuales se tratarán en epígrafes siguientes como instrumentos para llevar a cabo el desarrollo del siguiente trabajo de diploma.

1.2 Generadores de Reportes

Actualmente existen diversos sistemas de generación de reportes que presentan problemas con el dinamismo a la hora de brindar la información necesaria. Para este trabajo las herramientas son elaboradas con vista a generar un tipo de reporte determinado en un tiempo de desarrollo que obliga al

1 Esta definición está basada en la dada por Mary Show y David Garlan en ―Software Arquitecture -Perspective on an Emerging Discipline‖, 1996.

2 L. Bass, P. Clements, R. Kazman, Software Architecture in Practice, 2nd Edition, Addison Wesley, 2003

usuario a redefinir los requisitos cada vez necesite añadir una nueva funcionalidad. Existen otras, sin embargo, que aunque resuelven el problema planteado tienen un elevado costo de adquisición y son por lo general, orientadas al programador y no al usuario. Para el análisis se tuvo en cuenta las principales herramientas que existen en el mundo.

1.2.1 **Crystal Reports**

Crystal Reports es una aplicación de inteligencia empresarial utilizada para diseñar y generar informes desde una amplia gama de fuentes de datos (bases de datos). Se convirtió en el escritor de informes estándar cuando Microsoft lo liberó con Visual Basic.

Crystal Reports para Visual Studio .NET es la herramienta de elaboración de informes estándar para Visual Studio .NET. Permite crear contenido interactivo con calidad de presentación en la plataforma .NET, lo que ha supuesto una ventaja fundamental para Crystal Reports durante años.

Los desarrolladores pueden utilizar Crystal Reports para Visual Studio para hacer lo siguiente:

- Diseñar informes ilimitados para uso en aplicaciones de Visual Studio con el Diseñador de Crystal Reports integrado.
- Almacenar informes en plataformas Windows y Web y publicar informes Crystal como servicios Web de informes en un servidor Web.
- Integrar el motor de Crystal Reports en aplicaciones de Microsoft Windows de cliente grueso.
- Integrar el motor de Crystal Reports en aplicaciones Web.
- Pueden redistribuir libremente las aplicaciones Web con el motor en tiempo de ejecución incrustado dentro de la organización de desarrolladores.
- Para redistribuir aplicaciones Web fuera de organización los desarrolladores para su uso por terceros, debe obtener autorización escrita.

Sistema Operativo: Microsoft Windows XP con Service Pack (SP) 2, Windows Server 2003 con SP 1 o posterior.

Licencia: se distribuye bajo los términos de la EULA (*End User Licensing Agreement*)[3], por lo que es software privativo y de uso restringido mediante el pago de patente (*CrystalReports.com)*.

3 El EULA (end user license agreement) para SAP (Service Acces Point) Crystal Reports for Visual Studio 2010 será como el de SAP Crystal Reports 2008. Esto quiere decir que el cambio entre el EULA de Visual Studio 2008 y 2010 es que la redistribución de aplicaciones Web gratuita ya no está permitida.Ahora, es obligatorio adquirir SAP Crystal Reports runtime server license (anteriormente llamado Crystal Reports Developer Advantage) para poder redistribuir aplicaciones web que embeban Crystal Reports for Visual Studio 2010 o Crystal Reports 2008.

1.2.2 **Microsoft SQL Server 2005 Reporting Services**

SQL Server 2005 Reporting Services es un componente clave de SQL Server 2005. Es una plataforma de elaboración de informes basada en servidor que se puede utilizar para crear y administrar informes tabulares, matriciales, de gráficos y de formato libre con datos extraídos de orígenes de datos relacionales y multidimensionales. Se pueden visualizar y administrar mediante una conexión basada en la web. Añade una serie de nuevas funcionalidades y soluciona escenarios de uso nuevos: usuarios que desean interactuar con los datos dentro de los propios informes, así como la posibilidad de crear sus informes personales desde cero y compartirlos con otros. Reporting Services contiene los componentes principales siguientes:

- Servidor de informes que aloja y procesa informes en diversos formatos: HTML, PDF, TIFF, Excel, CSV, etc.
- Los informes incluyen características interactivas basadas en la Web
- API que permite a los programadores integrar o extender procesamiento de datos e informes en aplicaciones personalizadas.
- La arquitectura de desarrollo y tiempo de ejecución se ha creado con un diseño modular para admitir extensiones de terceros y posibilidades de integración (*Microsoft Corporation*).
- Microsoft SQL Server 2005 Reporting Services (SSRS) tiene cuatro temas principales. Las características específicas se tratan de manera detallada en las siguientes secciones.
- En SQL Server 2005 Reporting Services, los usuarios pueden enrutar directamente los trabajos de impresión, sin necesidad de exportarlos antes.

Sistema Operativo: Microsoft Windows, preferiblemente en sus versiones para servidores.

Licencia: se distribuye bajo licencia privativa perteneciente a Microsoft Corporation, dicha licencia forma parte de la licencia de Microsoft SQL Server 2000 ó 2005 (Microsoft Corporation).

1.2.3 **Jasper Reports**

Jasper Reports es una poderosa y flexible solución de código abierto para la generación y gestión de informes. Es un módulo que dispone de un depósito de archivos que usa un sistema de carpetas, una aplicación web que muestra todos los informes que están en el depósito y un visor de dichos informes. La aplicación web permite subir todos los informes creados, e inmediatamente estos estás disponibles para todos los usuarios.

Jasper Reports es una aplicación web en Java libre que tiene la habilidad de entregar contenido enriquecido al monitor, a la impresora o a ficheros PDF, HTML, XLS, CSV y XML y no requiere grande

requisitos para ponerle en marcha. Está escrito completamente en Java y puede ser usado en gran variedad de aplicaciones de Java, incluyendo J2EE o aplicaciones web, para generar contenido dinámico. Su propósito principal es ayudar a crear documentos de tipo páginas, preparados para imprimir en una forma simple y flexible. Jasper Reports se usa comúnmente con iReport, un front-end gráfico de código abierto para la edición de informes.

La utilización de esta herramienta permite mejorar la gestión de la empresa mediante la creación y gestión de informes. Con una solución de este tipo, se dispone en el tiempo deseado de los datos indispensables para la gestión eficaz de un departamento. También pueden seguirse fácilmente los resultados obtenidos y la manera en que la actividad progresa en función de los objetivos fijados. Pueden descubrirse las tendencias y los factores claves que afectan a la actividad y a los resultados de la empresa.

Sistema Operativo: puede ser utilizado en cualquier entorno o sistema operativo siempre que exista una implementación de la máquina virtual de Java para dicho entorno, las únicas dependencias que deben satisfacerse son: JDK (*Java Development Kit*) 1.3 o superior y JDBC 2.0 cuando se utilicen fuentes de datos relacionales para los demás casos no existen dependencias.

Licencia: el software es libre pues es distribuido mundialmente bajo los términos de la Licencia Publica para Librerías GNU (*GNU Library Public License*) y está respaldado por una gran comunidad internacional de desarrollo, el proyecto JasperForge.org y la empresa JasperSoft Corporation.

1.2.4 Comparación de las Herramientas de Generación de Reportes

Son varias las herramientas que existen en el mercado para la generación de reportes. No obstante, la elección de la más conveniente depende de las necesidades de los clientes así como el tipo de arquitectura a utilizar como base para soportar una implementación que responda a dichas necesidades. A continuación se ofrece una tabla comparativa resumiendo los aspectos fundamentales de estos sistemas.

Tabla 1. Comparación de las Herramientas de Generación de Reportes

Herramienta	Arquitectura	Licencia	Libre	Sistema Operativo
Jasper Reports	Librería	GNU/GPL	Si	Multiplataforma
SQL Server	Servidor	SQLS 2005	No	Windows 2003

Reporting Services				Server/XP SP2
Crystal Reports	Librería	EULA	No	Windows 2003 Server/XP SP2

Microsoft SQL Server 2005 Reporting Services, es un sistema que cumple con algunos de los requerimientos de los clientes, pero que una de las desventajas, es el hecho de ser software privativo y patentado por Microsoft, además de que no todas las aplicaciones con que cuenta son web, ejemplo el Diseñador de Reporte.

Como resultado del análisis de cada una de estas herramientas se decidió tomar la correcta con el objetivo de desarrollar la versión 2.0 del Generador Dinámico de Reportes (GDR) en este caso se escogió Jasper Reports, que con un conjunto de herramientas web permitirá satisfacer las necesidades de reportes de cualquier empresa o institución que lo requiera, siendo evidente el diseño de una arquitectura que haga posible y soporte la implementación del mismo atendiendo a la solicitud del cliente.

Para el diseño arquitectónico se hace necesario aplicar estilos que soporten la construcción de la arquitectura de software haciéndola flexible facilitando el trabajo de los desarrolladores, pero para ellos es importante utilizar los adecuados, por lo que en el siguiente epígrafe se realizará una descripción de los estilos arquitectónicos que pudieran ser utilizados en la solución.

1.3 **Estilos Arquitectónicos**

Un estilo arquitectónico define las reglas generales de organización en términos de un patrón y las restricciones en la forma y la estructura de un grupo numeroso y variado de sistemas de software. En una forma más específica, un estilo determina el vocabulario de componentes y conectores que pueden ser utilizados en instancias de este estilo, con un conjunto de restricciones en las descripciones arquitectónicas.

Un estilo describe una categoría de sistema que abarca un conjunto de componentes que realizan una función requerida por el sistema, un conjunto de conectores que posibilitan la comunicación, la coordinación y cooperación entre los componentes, las restricciones que definen como se integran los componentes para conformar el sistema, y los modelos semánticos que facilitan al diseñador el entendimiento de todas las partes del sistema. (4)

Los estilos se encuentran en el centro de la arquitectura y constituyen buena parte de su sustancia. Igual que los patrones de arquitectura y diseño, todos los estilos tienen un nombre y se ordenan en

seis o siete clases fundamentales y unos veinte ejemplares, como máximo. A continuación la clasificación propuesta de Mary Shaw y David Garlan en su clásico libro *"Software Architecture"* (1996).

- **Estilos de Flujo de datos.**
  Tuberías y Filtros
- **Estilos centrados en datos**
  Arquitecturas de Pizarra o repositorio
- **Estilos de Llamada y Retorno.**
  Modelo – Vista – Controlador (MVC)
  Arquitectura en Capas
  Arquitectura Orientada a Objetos
  Arquitectura Basada en Componentes
- **Estilo de Código Móvil**
  Arquitectura de Máquinas Virtuales
- **Estilos Peer–To–Peer**
  Arquitectura Basada en Eventos
  Arquitecturas Orientas a Servicios (SOA)

De esta forma, es evidente que un estilo arquitectónico define una familia de sistemas en términos de un patrón de organización estructural. En particular, de acuerdo a los autores Mary Shaw y David Garlan, un estilo arquitectónico define tanto un vocabulario de tipos de componentes y conectores.

1.3.1 Estilos de Llamada y Retorno

Permiten a los diseñadores de software conseguir estructuras relativamente fáciles de modificar y escalar. Este tipo de estilo es interesante en los sistemas que se centran en la interacción del usuario con el sistema. La arquitectura se puede dividir en dos partes, la primera representa la interfaz del usuario con el que este realiza la llamada al sistema, la segunda contiene la lógica de negocio que se realiza tras la correspondiente llamada del usuario.

Se describen a continuación algunas de las arquitecturas que están dentro de este tipo de estilo debido a su importancia en el desarrollo de sistemas.

Arquitectura orientada a objetos

Los componentes del sistema encapsulan datos y operaciones que deben de utilizarse para manipular dichos datos. La comunicación y coordinación entre componentes se realiza mediante envío de mensajes. Es un sistema donde se enfatiza el empaquetamiento entre datos y operaciones que permiten manipular y acceder a dichos datos.

Arquitectura en capas

Se definen como un conjunto de niveles o capas cada nivel interno que se atraviesa se aproxima más al nivel del conjunto de instrucciones máquina. Cada capa solo puede comunicarse con las vecinas y esto se realiza por medio de mensajes. Las capas superiores tienen operadores que actúan sobre los operadores de la capa inmediata inferior y el control se implementa capa por capa, siendo la responsabilidad de una capa controlar la ejecución de los operadores de la capa inmediata inferior. Esta solución aunque menos eficiente facilita la portabilidad en los diseños.

Modelo – Vista – Controlador (MVC): es un patrón de arquitectura de software que separa los datos de una aplicación, la interfaz de usuario, y la lógica de control en tres componentes distintos. (5)**.**

Descripción del patrón

Modelo: es la representación específica de la información con la cual el sistema opera. La lógica de datos asegura la integridad de estos y permite derivar nuevos datos.

Vista: presenta el modelo en un formato adecuado para interactuar, usualmente la interfaz de usuario. Maneja la presentación visual de los datos representados por el Modelo.

Controlador: responde a eventos, usualmente acciones del usuario e invoca cambios en el modelo y probablemente en la vista.

1.4 **Patrones**

Los patrones son formas de describir las mejores prácticas, buenos diseños, y encapsulan la experiencia de forma tal que es posible para otros el reutilizar dicha experiencia. Constituyen mecanismos cuyo objetivo es la solución de problemas que ocurren repetidamente dentro de un contexto muy bien definido. Definen la estructura de un sistema software, los cuales a su vez se componen de subsistemas con sus responsabilidades, también tienen una serie de directivas para organizar los componentes del mismo sistema, con el objetivo de facilitar la tarea del diseño de tal sistema.

Buschmann (6) define patrón como una regla que consta de tres partes, la cual expresa una relación entre un contexto, un problema y una solución.

Características de un buen patrón

Documentar buenos patrones puede ser una tarea muy difícil. Citando a James Coplien, un buen patrón:

1. Resuelve un problema: los patrones capturan soluciones, no principios o estrategias abstractas.

2. Es un concepto probado: capturan soluciones, no teorías o especulaciones. En el caso del *"Design Patterns",* el criterio para decidir si algo era un patrón o no, era que éste debía tener al menos 3 implementaciones reales.
3. La solución no es obvia: muchas técnicas de resolución de problemas (como los paradigmas o métodos de diseño de software) intentan derivar soluciones desde principios básicos. Los mejores patrones generan una solución a un problema indirectamente (un enfoque necesario para los problemas de diseño más difíciles).
4. Describe una relación**:** los patrones no describen módulos sino estructuras y mecanismos.
5. Tiene un componente humano significante**:** el software sirve a las personas. Los mejores patrones aplican a la estética y a las utilidades.
6. Los patrones de software facilitan la reutilización del diseño y de la arquitectura, capturando las estructuras estáticas y dinámicas de colaboración de soluciones exitosas a problemas que surgen al construir aplicaciones.
7. El propulsor de la idea de patrones fue Christopher Alexander en la década de 1970, un patrón es una solución a un problema en un contexto, codifica conocimiento específico acumulado por la experiencia en un dominio. Un sistema bien estructurado está lleno de patrones. (7)

1.4.1 **Categorías de patrones**

Según la escala o nivel de abstracción los patrones se clasifican en:

Patrones de arquitectura: aquellos que expresan un esquema organizativo estructural fundamental para sistemas de software. Especifican las propiedades de la estructura global del sistema y tienen un impacto en la arquitectura de cada subsistema.

Patrones de diseño: expresan esquemas para definir estructuras de diseño (o sus relaciones) con las que se puedan construir sistemas de software. La aplicación de estos patrones no tiene un efecto en la estructura fundamental de un sistema pero pueden tener una influencia considerable en la arquitectura de un subsistema.

Patrones de Idiomas: patrones de bajo nivel específicos para un lenguaje de programación o entorno concreto.

En términos generales, un patrón es un conjunto de información que proporciona respuesta a un conjunto de problemas similares, es decir, un patrón es una solución a un problema en un contexto, donde: el contexto son las situaciones recurrentes a las que es posible aplicar el patrón; el problema es el conjunto de metas y restricciones que se dan en ese contexto; la solución es el diseño a aplicar para conseguir las metas dentro de las restricciones.

➢ Patrones de arquitectura

Son patrones del software los cuales se encargan de definir la estructura de un sistema, estos a su vez se componen de subsistemas con sus responsabilidades, también tienen una serie de directivas para organizar los componentes del mismo sistema, con el objetivo de facilitar la tarea del diseño de tal sistema. Un patrón arquitectónico se enfoca a dar solución a un problema en específico, de un atributo de calidad, y abarca solo parte de la arquitectura.

Aun cuando un patrón arquitectónico transporta una imagen de un sistema, él no es una arquitectura como tal. Un patrón arquitectónico es un concepto que captura elementos esenciales de una arquitectura del software.

Uno de los aspectos más importantes de los patrones arquitectónicos es que encarnan diferentes atributos de calidad. Por ejemplo, algunos patrones representan soluciones a problemas de rendimiento y otros pueden ser utilizados con éxito en sistemas de alta disponibilidad.

Buschmann (6) plantean que los patrones arquitectónicos expresan el esquema de organización estructural fundamental para sistemas de software. Provee un conjunto de subsistemas predefinidos, especifica sus responsabilidades e incluye reglas y pautas para la organización de las relaciones entre ellos. Propone que son plantillas para arquitecturas de software concretas, que especifican las propiedades estructurales de una aplicación - con amplitud de todo el sistema - y tienen un impacto en la arquitectura de subsistemas. La selección de un patrón arquitectónico es, por lo tanto, una decisión fundamental de diseño en el desarrollo de un sistema de software.

➢ Patrones de Diseño

Los patrones de diseño son la base para la búsqueda de soluciones a problemas comunes en el desarrollo de un software y otros ámbitos referentes al diseño de interacción o interfaces. Para que una solución sea considerada un patrón debe poseer ciertas características. Una de ellas es que debe haber comprobado su efectividad resolviendo problemas similares en ocasiones anteriores. Otra es que debe ser reusable, lo que significa que es aplicable a diferentes problemas de diseño en distintas circunstancias.

Un patrón de diseño provee un esquema para refinar los subsistemas o componentes de un sistema de software, o las relaciones entre ellos. Describe la estructura comúnmente recurrente de los componentes en comunicación, que resuelve un problema general de diseño en un contexto particular (6).

Dentro de los patrones de diseño existen los patrones GOF (*Gang of Four* o Pandilla de los cuatro) los cuales juegan un papel muy importante dentro de los patrones de diseño de forma general y son

utilizados en innumerables ocasiones cuando se desarrollan aplicaciones informáticas. También se encuentran los patrones GRASP (Patrones de Asignación de Responsabilidades) que comunican los principios fundamentales de asignación de responsabilidades en el diseño orientado a objetos. (8)

Los patrones de diseño se dividen en tres grupos principales:

- **Patrones creacionales**: solucionan problemas de creación de instancias. Ayudan a encapsular y abstraer dicha creación.
  Patrón de Fábrica Abstracta (*Abstract Factory*), Patrón Constructor (*Builder*),Patrón del Método de Fabricación (*Factory Method*), Patrón Prototipo (*Prototype*), Patrón de Instancia Única (*Singleton*):
- **Patrones estructurales**: solucionan problemas de composición (agregación) de clases y objetos.
  Patrón Adaptador (*Adapter*), Patrón Puente (*Bridge*), Patrón Compuesto (*Composite*), Patrón Decorador (*Decorator*), Patrón de Fachada (*Facade*), Patrón de Peso Mosca o ligero (*FlyweigthPatrón Proxy* (*Proxy*)).
- **Patrones de comportamiento:** soluciones respecto a la interacción y responsabilidades entre clases y objetos, así como los algoritmos que encapsulan. Patrón de Cadena de Responsabilidad (*Chain of Responsibility*),Patrón de Comando (*Command*),Patrón Intérprete (*Interpreter*), Patrón Iterador (*Iterator*), Patrón Mediador (Mediator),Patrón Recuerdo (Memento), Patrón Observador (*Observer*),Patrón de Estado (*State*), Patrón de Estrategia (*Strategy*), Patrón del Método Plantilla (*Template Method*), Patrón Visitante (*Visitor*).

Para el desarrollo de la aplicación se van a utilizar todos los patrones Alta Cohesión, Bajo Acoplamiento, Controlador y Experto que pertenecen al grupo de los GRASP por su utilidad para lograr un buen diseño orientado a objetos, además de estos serán utilizados los patrones GOF específicamente el Singleton, el Factory, el Observer y el Command para lograr una mejor organización en cuanto al código y la reutilización del mismo.

1.5 Lenguajes de Programación y Tecnologías

Existe un gran número de lenguajes de programación web que se pudieran utilizar para el desarrollo del sistema propuesto dentro de ellos se encuentra: Java, Perl, Python o Ruby, pero el cliente especificaba dentro de los requisitos no funcionales que la solución debía ser implementada sobre PHP, para no entrar en contraste con el resto de los servicios y aplicaciones que son utilizados. Además se utiliza XML, XSLT, HTML, CCS y JavaScript para facilitar el trabajo en el desarrollo de la

aplicación aprovechando los beneficios de cada uno de estos lenguajes para las funcionalidades que requieran de ellos. Con el objetivo de lograr interactividad, velocidad y usabilidad en la aplicación se utiliza la técnica de desarrollo web para crear aplicaciones RIA (***R**ich **I**nternet **A**pplications)* AJAX, tecnología explicada más adelante.

1.5.1 **Lenguajes de Programación: PHP**

PHP es un lenguaje de programación interpretado que significa Hypertext Pre-processor, diseñado originalmente para la creación de páginas web dinámicas. Es usado principalmente en interpretación del lado del servidor (*server-side scripting*) para la generación de páginas Web dinámicas, similar al ASP de Microsoft o el JSP de Sun, embebido en páginas HTML y ejecutado en el servidor.

La mayor parte de su sintaxis ha sido tomada de C, Java y Perl con algunas características específicas de sí mismo. La meta del lenguaje es permitir rápidamente a los desarrolladores la generación dinámica de páginas. No es un lenguaje de marcas como podría ser HTML, XML o WML. Está más cercano a JavaScript o a C.

Las principales características de PHP son: su rapidez; su facilidad de aprendizaje; su soporte multiplataforma tanto de diversos Sistemas Operativos, como servidores HTTP y de bases de datos; y el hecho de que se distribuye de forma gratuita bajo una licencia abierta.

Paradigma: multiparadigma.

Tipo de dato: dinámico.

Sistema operativo: multiplataforma.

Soporte para bases de datos: MysSQL, PostgreSQL, Oracle, MS SQL Server, Sybase mSQL, Informix, entre otras.

Ventajas

- El código fuente escrito en PHP es invisible al navegador y al cliente porque es el servidor el que se encarga de ejecutar el código y enviar su resultado HTML al navegador. Esto hace que la programación en PHP sea segura y confiable.
- Completamente orientado al desarrollo de aplicaciones web dinámicas con acceso a información almacenada en una Base de Datos.
- Es libre, por lo que se presenta como una alternativa de fácil acceso para todos.
- Permite aplicar técnicas de programación orientada a objetos.
- Biblioteca nativa de funciones sumamente amplia e incluida.

1.5.2 **Tecnología AJAX**

AJAX, acrónimo de ***A****synchronous* ***Java****Script* ***A****nd* ***X****ML* (JavaScript asíncrono y XML), es una técnica de desarrollo web para crear aplicaciones interactivas o RIA (*Rich Internet Applications*). Estas aplicaciones se ejecutan en el cliente, es decir, en el navegador de los usuarios mientras se mantiene la comunicación asíncrona con el servidor en segundo plano. De esta forma es posible realizar cambios sobre las páginas sin necesidad de recargarlas, lo que significa aumentar la interactividad, velocidad y usabilidad en las aplicaciones.

Es una técnica válida para múltiples plataformas y utilizable en muchos sistemas operativos y navegadores, dado que está basado en estándares abiertos como JavaScript y DOM (*Document Object Model*). AJAX es una combinación de cuatro tecnologías ya existentes:

- XHTML (o HTML) y hojas de estilos en cascada (CSS) para el diseño que acompaña a la información.
- DOM accedido con un lenguaje de scripting por parte del usuario, especialmente implementaciones ECMAScript como JavaScript y JScript, para mostrar e interactuar dinámicamente con la información presentada.
- El objeto XMLHttpRequest para intercambiar datos de forma asíncrona con el servidor web. En algunos frameworks y en algunas situaciones concretas, se usa un objeto iframe en lugar del XMLHttpRequest para realizar dichos intercambios.
- XML es el formato usado generalmente para la transferencia de datos solicitados al servidor, aunque cualquier formato puede funcionar, incluyendo HTML preformateado, texto plano, JSON y hasta EBML.

1.6 Lenguaje de Modelado

El problema de la descripción de una arquitectura de software es encontrar una técnica que cumpla con los propósitos del desarrollo de software; en otras palabras, la comunicación entre las partes interesadas, la evaluación y la implementación (9). Para poder expresar las características del sistema se aplica una convención gráfica o algún lenguaje avanzado de alto nivel de abstracción

1.6.1 Lenguajes de Descripción de Arquitectura

Los lenguajes de descripción de arquitecturas, o ADLs (*Architecture Description Languages*), deben proporcionar un modelo explícito de componentes, conectores y sus respectivas configuraciones. Permiten modelar una arquitectura mucho antes que se lleve a cabo la programación de las aplicaciones que la componen, analizar su adecuación, determinar sus puntos críticos y eventualmente

simular su comportamiento, o sea proveen una forma de construir una estructura de alto nivel.
La definición más simple es la de Tracz (10) que define un ADL como una entidad consistente en cuatro “Cs”: componentes, conectores, configuraciones y restricciones (*constraints*).

1.6.2 **Lenguaje Unificado de Modelado**

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; está respaldado por el OMG (*Object Management Group*). Es un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema. UML ofrece un estándar para describir un "plano" del sistema (modelo), incluyendo aspectos conceptuales tales como procesos de negocio y funciones del sistema, y aspectos concretos como expresiones de lenguajes de programación, esquemas de bases de datos y componentes reutilizables.
Al desarrollar una aplicación puede ser de gran valor describir estas vistas dentro de un documento de arquitectura pues estas cinco vistas pretenden describir la arquitectura del sistema.
Para resumir se pudiera decir que existen dos facciones claramente definidas en la comunidad de arquitectos: la primera está vinculada con el Rational y UML impulsando la utilización de UML como si fuera un ADL; la segunda ha señalado algunas limitaciones de UML como lenguaje de modelado para definir arquitecturas por la forma de expresar ciertas características, sobre todo las dinámicas de las estructuras que no es suficiente para los arquitectos (11).Pero por el contexto donde se desarrolla y se inscribe el proyecto es necesario asumir la primera variante por ir en consecuencia con la estandarización del proceso de desarrollo de software.

1.7 **Frameworks**

En el desarrollo de software, un framework es una estructura conceptual y tecnológica de soporte definida, normalmente con artefactos o módulos de software concretos, con base en la cual otro proyecto de software puede ser organizado y desarrollado. Típicamente, puede incluir soporte de programas, bibliotecas y un lenguaje interpretado entre otros programas para ayudar a desarrollar y unir los diferentes componentes de un proyecto.
Los objetivos principales que persigue un framework son: acelerar el proceso de desarrollo, reutilizar código ya existente y promover buenas prácticas de desarrollo como el uso de patrones. Un framework Web, por tanto, puede ser definido como un conjunto de componentes (por ejemplo clases en Java y descriptores y archivos de configuración en XML) que componen un diseño reutilizable que facilita y agiliza el desarrollo de sistemas Web. Existen numerosos frameworks como CakePHP, Prado,

PHPDevShel, Zend Framework, ExtJS, Kumbia, Symfony entre otros muchos. Para su evaluación como posibles a utilizar fueron seleccionados Symfony, ExtJS y CakePHP por estar entre los más usados.

1.7.1 **Symfony**

Symfony es uno de los frameworks PHP más populares entre los usuarios y las empresas, pues permite que los programadores sean mucho más productivos a la vez que crean código de más calidad y más fácil de mantener. Symfony es maduro, estable, profesional y está muy bien documentado.

Symfony fue diseñado para ajustarse a los siguientes requisitos:

- Fácil de instalar y configurar en la mayoría de plataformas (y con la garantía de que funciona correctamente en los sistemas Windows y *nix estándares).
- Independiente del sistema gestor de bases de datos. Su capa de abstracción y el uso de Propel, permiten cambiar con facilidad de Sistema Gestor de Base de Datos (SGBD) en cualquier fase del proyecto.
- Doctrine es uno de los ORM que tiene Symfony que facilita la elaboración de consultas muy complejas y mejora el rendimiento.
- Utiliza programación orientada a objetos, de ahí que sea imprescindible PHP 5.
- Sencillo de usar en la mayoría de casos, aunque es preferible para el desarrollo de grandes aplicaciones Web que para pequeños proyectos.
- Aunque utiliza MVC (Modelo Vista Controlador), tiene su propia forma de trabajo en este punto, con variantes del MVC clásico como la capa de abstracción de base de datos, el controlador frontal y las acciones.
- Sigue la mayoría de mejores prácticas y patrones de diseño para la web.
- Fácil de extender, lo que permite su integración con las bibliotecas de otros fabricantes.
- Una potente línea de comandos que facilitan generación de código, lo cual contribuye a ahorrar tiempo de trabajo.

1.7.2 **ExtJS**

ExtJS es una librería JavaScript para la creación de aplicaciones enriquecidas del lado del cliente. Sus características principales son: gran desempeño, componentes de interfaz de usuario personalizables, con buen diseño y documentación.

ExtJS tiene un modelo de licencia dual. Para aplicaciones web comerciales hay que adquirir una

licencia y para aplicaciones web open source, que sean compatibles con GNU GPL license v3, se puede utilizar la licencia gratuita. Sirve de puente entre las librerías JS más usadas (Prototype, JQuery, YUI). Debido a que se inicio como una extensión de YUI esta presenta una cierta ventaja de compatibilidad respecto a las otras dos.

En la solución se utilizará Symfony para ejecutar toda la lógica del negocio y ExtJS para el desarrollo de las interfaces debido a las facilidades que brindan ambos frameworks y a sus características que hacen que tengan múltiples aplicaciones.

1.8 Metodología de desarrollo

Metodología de desarrollo en ingeniería de software es un marco de trabajo usado para estructurar, planificar y controlar el proceso de desarrollo en sistemas de información.

1.8.1 Proceso Unificado Abierto (OpenUP)

OpenUP es un Proceso Unificado Abierto que mantiene las mismas características de RUP, que contiene el desarrollo iterativo, casos de uso y escenarios de conducción de desarrollo, gestión de riesgos y el enfoque centrado en la arquitectura. Es un proceso mínimo y suficiente porque solo se incluye el contenido necesario y fundamental. Tiene los componentes básicos que pueden servir de base a procesos específicos y la mayoría de los elementos están declarados para fomentar el intercambio de información entre los equipos de desarrollo y mantener un entendimiento compartido del proyecto, sus objetivos, alcance y avances.

Fue liberado por el EPF (*Eclipse Process Framework*), se construyó sobre una donación realizada por la IBM del Basic Unified Process. Fue entregada a Eclipse a fines de 2005 y renombrado como OpenUp en 2006.

OpenUP estructura el ciclo de vida de un proyecto en cuatro fases: concepción, elaboración, construcción y transición. El ciclo de vida del proyecto provee a los interesados un mecanismo de supervisión y dirección para controlar los fundamentos del proyecto, su ámbito, la exposición a los riesgos, el aumento de valor y otros aspectos.

Está organizado en dos dimensiones diferentes pero interrelacionadas: **el método y el proceso.**

En el **método** se definen los roles, tareas, artefactos y lineamientos sin tener en cuenta como son utilizados en el ciclo de vida del proyecto. El **proceso** es donde se aplican los elementos del método de forma ordenada en el tiempo. A partir de un mismo conjunto de elementos del método se pueden crear muchos ciclos de vida para diferentes proyectos.

- **Flujos de Trabajo de OpenUp**

OpenUp divide el ciclo de vida del proyecto en cuatro fases:

1. Concepción: primera de las 4 fases en el proyecto del ciclo de vida, acerca del entendimiento del propósito y objetivos obteniendo suficiente información para confirmar que el proyecto debe hacer. El objetivo de ésta fase es capturar las necesidades de los stakeholder en los objetivos del ciclo de vida para el proyecto.
2. Elaboración: se trata los riesgos significativos para la arquitectura. El propósito de esta fase es establecer la base la elaboración de la arquitectura del sistema.
3. Construcción: esta fase está enfocada al diseño, implementación y prueba de las funcionalidades para desarrollar un sistema completo. El propósito de esta fase es completar el desarrollo del sistema basado en la Arquitectura definida.
4. Transición: es la última fase, cuyo propósito es asegurar que el sistema es entregado a los usuarios, y evalúa la funcionalidad y performance del último entregable de la fase de construcción.

1.9 **Herramientas CASE**

Las herramientas CASE (*Computer Aided Software Engineering*, Ingeniería de Software Asistida por Computadora) son diversas aplicaciones informáticas destinadas a aumentar la productividad en el desarrollo de software reduciendo el costo de las mismas en términos de tiempo y de dinero. Estas herramientas pueden ayudar en todos los aspectos del ciclo de vida de desarrollo del software en tareas como el proceso de realizar un diseño del proyecto, cálculo de costes, implementación de parte del código automáticamente con el diseño dado, compilación automática, documentación o detección de errores entre otras.

1.9.1 **Visual Paradigm**

Visual Paradigm para UML es una herramienta UML profesional que soporta el ciclo de vida completo del desarrollo de software: análisis y diseño orientados a objetos, construcción, pruebas y despliegue. El software de modelado UML ayuda a una más rápida construcción de aplicaciones de calidad, mejores y a un menor coste. Permite dibujar todos los tipos de diagramas de clases, código inverso, generar código desde diagramas y generar documentación. La herramienta UML CASE también proporciona abundantes tutoriales de UML, demostraciones interactivas de UML y proyectos UML.

Algunas características

- Ingeniería inversa - Código a modelo, código a diagrama.
- Generación de bases de datos - Transformación de diagramas de Entidad-Relación en tablas de base de datos.

- Licencia: gratuita y Comercial.
- Fácil de instalar y actualizar.

Se escogió Visual Paradigm como herramienta para el desarrollo por sus potenciales de interoperabilidad con otras herramientas utilizadas así como las facilidades que brinda desde el punto de vista del diseño gráfico.

1.10 **Gestores de Base de Datos**

Un Sistema Gestor de base de datos (SGBD) es un conjunto de programas que permiten crear y mantener una Base de datos, asegurando su integridad, confidencialidad y seguridad. Permite almacenar y posteriormente acceder a los datos de forma rápida y estructurada. Las aplicaciones más usuales son para la gestión de empresas e instituciones públicas, además de ser ampliamente utilizado en entornos científicos con el objeto de almacenar la información experimental.

1.10.1 **PostgreSQL**

PostgreSQL es un SGBD Objeto-Relacional basado en el proyecto POSTGRES, de la universidad de Berkeley. Es una derivación libre de este proyecto, y utiliza el lenguaje SQL. Fue pionero en muchos de los conceptos del sistema objeto-relacional actual. Este proyecto lleva más de una década de desarrollo, siendo hoy día, el sistema libre más avanzado, soportando la gran mayoría de las transacciones SQL y control concurrente.

Entre sus principales ventajas se destacan las siguientes:

- Permite la gestión de diferentes usuarios, como también los permisos asignados a cada uno de ellos.
- Posee una gran escalabilidad, haciéndolo idóneo para su uso en sitios Web que atienden un gran número de solicitudes.
- Puede ser instalado un número ilimitado de veces sin temor de sobrepasar la licencia.
- Posee estabilidad y confiabilidad legendarias.
- Es extensible a través del código fuente disponible sin costos adicionales.
- Es multiplataforma, disponible en la mayoría de los sistemas operativos.
- Permite implementar reglas, vistas, disparadores, subconsultas y funciones.
- Posee herramientas para generar SQL portable para compartir con otros sistemas compatibles con SQL.

1.11 **Ambiente de Desarrollo**

Un entorno de desarrollo informático IDE (*Integrated Development Environment*) es un programa informático compuesto por un conjunto de herramientas de programación. Consiste en un editor de código, un compilador, un depurador y un constructor de interfaz gráfica (GUI). Los IDE proveen un marco de trabajo amigable para la mayoría de los lenguajes de programación tales como C++, Python, Java, C#, Delphi, Visual Basic, etc.

1.11.1 **Eclipse**

Eclipse es un entorno de desarrollo integrado de código abierto multiplataforma, es una herramienta para desarrolladores Java que permite crear aplicaciones JEE y web, incluye un entorno integrado de desarrollo para Java. Posee una estructura modular, extensible mediante plugins, que le permite trabajar con cualquier tipo de recurso: gráficas, vídeo, modelos 3D, contenido web, etcétera. Otros lenguajes que también pueden utilizarse en Eclipse son: C/C++, PHP, Ruby, TCL o JavaScript.
Como IDE para Java cuenta algunas funciones interesantes, entre ellas: desarrollo de aplicaciones en grupo, unidad integrada de depuración y pruebas, compilación y construcción incremental.

1.11.2 **NetBeans**

La plataforma NetBeans permite que las aplicaciones sean desarrolladas a partir de un conjunto de componentes de software llamados módulos. Un módulo es un archivo Java que contiene clases de Java escritas para interactuar con las APIs de NetBeans y un archivo especial (*manifest file*) que lo identifica como módulo. Debido a que los módulos pueden ser desarrollados independientemente, las aplicaciones basadas en la plataforma NetBeans pueden ser extendidas fácilmente por otros desarrolladores de software. El IDE de NetBeans es gratuito, y de código abierto para desarrolladores de software. Provee una estructura para los proyectos, propone un esqueleto para organizar código fuente, el editor conjuntamente integra los lenguajes como HTML, JavaScript y CSS.

1.12 **Calidad en la Arquitectura de Software**

La Evaluación Arquitectónica es el proceso de predecir atributos de calidad del sistema basado en el desarrollo de una Arquitectura de Software. La evaluación explícita de la arquitectura de los Sistemas de Software (SS) con respecto a los requerimientos de calidad, minimizará los riesgos de construir un sistema que falle y consecuentemente disminuirá el costo de desarrollo del sistema (Bosch, 2000). De esta manera, el interés se centra en soportar la arquitectura en este proceso.

De acuerdo con Clements et al. (2002), hasta hace poco no existían métodos de utilidad general para evaluar arquitecturas de software. Si alguno existía, sus enfoques eran incompletos, *ad hoc*, y no repetibles, por lo que resultaban poco confiables. En virtud de esto, se han propuesto múltiples métodos de evaluación, entre los cuales se destacan: SAAM (*Software Analysis Architecture Method)*, ATAM (*Architecture Trade-off Analysis Method*), ARID (*Active Reviews for Intermediate Designs*).

1.12.1 **Método de Evaluación para Arquitecturas Parciales (ARID)**

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

1.12.2 **Método de Análisis de Acuerdos de Arquitectura (ATAM)**

ATAM (*Architecture Trade-off Analysis Method*): está inspirado en tres áreas distintas: los estilos arquitectónicos, el análisis de atributos de calidad y el método de evaluación SAAM (*Software Architecture Analysis Method*). Este método de evaluación indica cuan bien una arquitectura particular satisface las metas de calidad, y provee ideas de cómo esas metas de calidad interactúan entre ellas, como realizan concesiones mutuas (tradeoff) entre ellas. Se concentra en la identificación de los estilos arquitectónicos o enfoques arquitectónicos utilizados. El propósito de ATAM es evaluar las consecuencias de las decisiones arquitectónicas sobre los atributos de calidad necesarios. Es un método de identificación de riesgos, o sea detecta las áreas de riesgos potenciales en la arquitectura de un sistema. Puede hacerse en una fase temprana en el ciclo de vida del desarrollo de software. Evalúa de una forma temprana los artefactos del diseño arquitectónico.

1.12.3 **Método de Análisis de Arquitecturas de Software (SAAM)**

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

El método SAAM se sugiere cuando el atributo de calidad modificabilidad es el de mayor interés.

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

1.13 Conclusiones del Capítulo

En el presente capítulo se han abordado los aspectos relacionados con objeto de estudio, con el objetivo de proveer una solución que permita conformar la versión 2.0 del Sistema Generador Dinámico de Reportes. Se realizó un estudio de las herramientas para la generación de reportes tomándose a Jasper Reports como el motor de reportes a utilizar. Fueron abordados los principales conceptos relacionas con la arquitectura de software y se fundamentaron los principales estilos y patrones, lenguajes de descripción de arquitectura. Se seleccionaron las herramientas y metodología a utilizar y su integración con el objeto de estudio, enmarcando las principales actividades y artefactos a desarrollar por el equipo de arquitectura. Tales aspectos dan paso a las acciones del capítulo 2 y sirven de base para proporcionar una solución simple, escalable y confiable para definir una arquitectura sólida, robusta y bien concebida.

CAPÍTULO 2: Descripción Arquitectónica

Introducción

En este capítulo se hace una propuesta de solución al problema planteado en el diseño teórico de la investigación incluyendo en ella los objetivos y restricciones arquitectónicas. Se realiza la descripción de la arquitectura del sistema a través de las vistas de la arquitectura teniendo en cuenta los requisitos funcionales y no funcionales del sistema para lograr que éste sea escalable, configurable y portable.

2.1 Proceso de Generación de Reportes

El Sistema Generador Dinámico de Reportes (SGDR) presenta una arquitectura Cliente–Servidor para lograr una mayor escalabilidad y flexibilidad en su uso. Donde el cliente y el servidor son dos sistemas desacoplados no dependientes, permitiendo con esto la integración de dicho sistema con cualquier aplicación o software que necesite incluir en sus funcionalidades la generación de reportes sin tener la necesidad de utilizar la interfaz de usuario presentada por el mismo, de ahí que el cliente este compuesto por todos los elementos que conforman la interfaz y en el servidor se lleven a cabo todos los procesos que tiene lugar durante el ciclo de vida de un reporte que son soportados fundamentalmente por el motor de reportes, manteniendo una comunicación entre ambos sistemas mediantes JSON (*JavaScript Object Notation*) que es un formato ligero para el intercambio de datos y es un subconjunto de la notación literal de objetos de JavaScript que no requiere el uso de XML.

El ciclo de vida de un reporte pasa por tres etapas: creación, administración y entrega .Cada aplicación que se encargue de la generación de reportes debe incluir dichas funcionalidades y el SGDR soporta todo este proceso auxiliándose de cada módulo especializado para cumplir funciones en dependencia de la etapa que representa. En la Figura 1 se muestra la integración de las aplicaciones del sistema y sus módulos en el ciclo de vida del reporte.

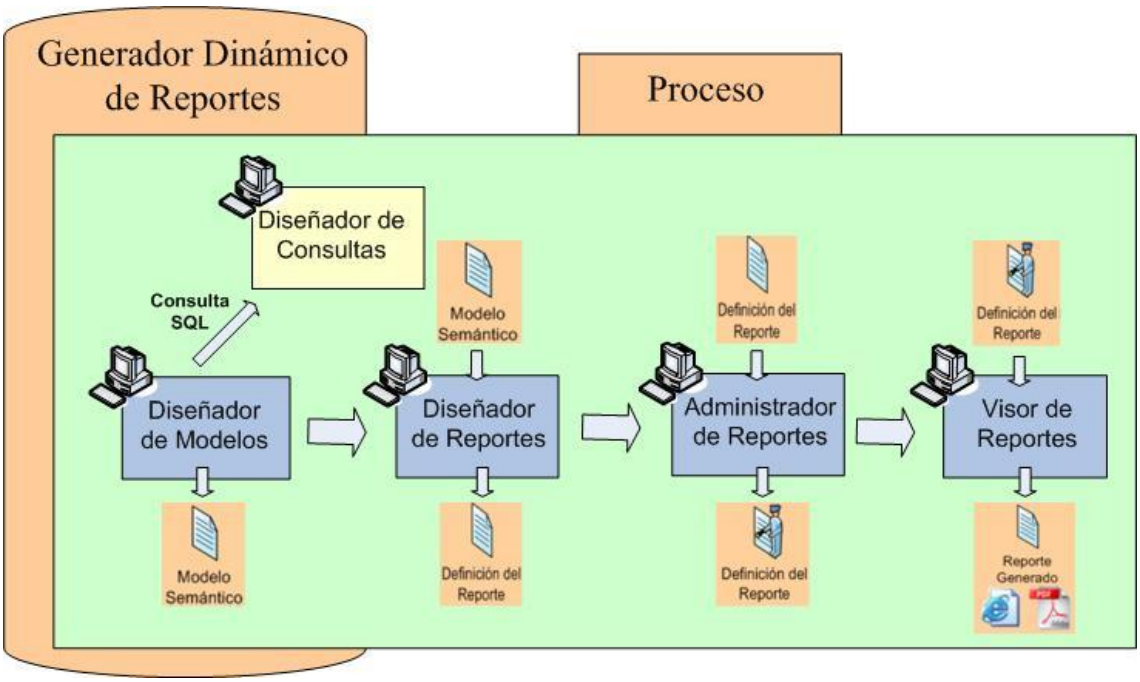


Figura 1. Ciclo de Vida de los Reportes

El Diseñador de Modelos comienza el ciclo de vida pues crea una abstracción de los orígenes de datos llamado modelo semántico que describe la estructura de los datos, y el Diseñador de Reportes se encarga de conformar el informe a partir del modelo semántico obtenido del Diseñador de Modelos generando como salida la definición del reporte, también el Diseñador de Consultas sirve como origen de datos para casos específicos donde se necesite obtener información de la relación entre dos tablas diferentes de una base de datos y dichos módulos soportan el proceso de creación de informes. Posteriormente en la fase de administración el módulo Administrador de Reportes es el encargado de llevar a cabo las tareas para la gestión de los modelos semánticos y las definiciones de los reportes. Por último la etapa de entrega de reportes que se soporta por el Visor de Reportes el cual posibilita la visualización y exportación del informe en diferentes formatos.

2.2 Objetivos y Restricciones Arquitectónicas

El objetivo de la arquitectura que se presenta es lograr un sistema escalable, configurable y portable que sea capaz de brindar servicios de generación y administración de reportes de forma dinámica. Se conforma teniendo en cuenta los requerimientos funcionales y no funcionales del software, pero estos aspectos no son los únicos que inciden en su estructura pues además es necesario tener en cuenta las restricciones impuestas por al ambiente donde el sistema debe operar, la necesidades de reutilización de sus componentes así como la compatibilidad entre diferentes sistemas y la adopción de estándares

son ejemplos de elementos a tener en cuenta durante el diseño arquitectónico. Además pueden existir un conjunto de principios y políticas de la arquitectura que guíen el desarrollo del proyecto que deben ser observadas así como cualquier decisión arquitectónica derivada de esto.

2.2.1 **Requerimientos**

Los requerimientos y restricciones del sistema tienen un impacto significativo en la arquitectura por lo cual en el documento de Especificación de Requisitos del proyecto se recogen tanto los requisitos funcionales como los no funcionales que debe cumplir, a continuación se describen los requisitos no funcionales fundamentales que le dan forma a la arquitectura propuesta.

RNF 1. Requisito de Usabilidad

El sistema debe ser intuitivo y tener un alto nivel de usabilidad permitiendo que usuarios sin mucho conocimiento informático, en aproximadamente 30 días puedan explotarlo al 100%.La herramienta debe ser WEB, pero con características muy similares a las aplicaciones de escritorio en cuanto al diseño de las interfaces visuales y los tiempos de respuesta de la interacción del usuario con el sistema.

RNF 1.1. Permitir la personalización de los reportes.

El usuario podrá diseñar un reporte lo más ajustado posible a sus necesidades, se permitirá cambiar la tipografía del texto, agregar imágenes, ubicar en el área de diseño los campos en la sección que el usuario desee y usando características de arrastrar y soltar.

RNF 1.2. Buscar de manera rápida y sencilla un reporte que se desee visualizar.

El usuario podrá localizar un reporte de manera rápida, si es posible tenerlo ubicado en alguna categoría que permita tener la posibilidad de organizarlos por áreas temáticas comunes.

RNF 1.3. Diseñar una consulta SQL de manera sencilla y ágil.

El usuario podrá diseñar una consulta SQL de manera ágil y sencilla sin necesidad de ser un experto en el lenguaje SQL.

RNF 2. Requisitos de Fiabilidad.

RNF 2.1. Permanecer online constantemente para atender a todas las solicitudes.

Debido a que el sistema mantendrá en todo momento una cola de eventos a ejecutar para la generación y envío automático de reportes, es necesario que este permanezca online constantemente para atender dichas solicitudes o alguna otra desde un usuario directamente. En caso de fallo, pudiera estar fuera de servicio por un período de 72 horas máximo. La precisión y exactitud de las salidas del

sistema se corresponden con la calidad y exactitud de la información contenida en las base de datos desde donde se extraigan los reportes. El sistema no será responsable por la falta de veracidad de dicha información. Algunos errores que pueden resultar críticos son:

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

RNF 3. Requisitos de Eficiencia.

La eficiencia del sistema depende en su mayor parte de la velocidad de conexión a la base de datos donde se encuentre y al volumen de información que contengan las mismas.

RNF 3.1. El sistema debe ser rápido y ofrecer tiempos de respuesta relativamente bajos.

Con este fin se implementará un mecanismo de paginación de los reportes que permitirá que los mimos sean obtenidos de manera progresiva disminuyendo de esta forma el tráfico de la información por la red y las consultas que de devuelven grandes cantidades de registros.

RNF 3.2. El sistema debe permitir la concurrencia.

El sistema debe permitir que existan al menos 100 usuarios conectados de forma simultánea.

RNF 3.3. Tiempo de máximo de respuesta para la obtención de un reporte.

El tiempo máximo para la obtención de un reporte no debe sobrepasar los 5 minutos.

RNF 3.4. Tiempo de promedio de respuesta para la obtención de un reporte.

Como promedio un reporte debe demorar alrededor de 10 segundos.

RNF 4. Requisitos de Arquitectura y Plataforma.

El sistema debe ser implementado en el lenguaje de programación PHP versión 5.3.3 o superior. Como marco de trabajo se usará Symfony el cual propone una arquitectura modular en tres capas: el modelo, la vista y el controlador.

Una de las bibliotecas fundamentales en el desarrollo de la herramienta será el marco de trabajo ExtJS la cual es una librería en JavaScript que permite el diseño de interfaces visuales interactivas usando tecnologías como AJAX y permite crear aplicaciones WEB con apariencia similar a las de escritorio. Otra librería importante y necesaria en el desarrollo de la herramienta será Jasper Reports la cual constituye el núcleo del proceso de generación de reportes.

RNF 5. Requisitos de Software.

RNF 5.1. El sistema debe ser desarrollado sobre software libre.

RNF 5.2. El sistema debe correr y brindar el servicio desde un servidor con Sistema Operativo libre. (Debian GNU/Linux, rama Stable).

RNF 5.3. El sistema debe ser implementado en lenguaje de programación PHP, versión 5.3.3.

RNF 5.4. El sistema debe utilizar PostgreSQL versión 9 o superior como gestor de bases de datos.

RNF 5.5. El sistema debe brindar servicio a través del servidor web Apache 2.2.16.

RNF 5.6. Todas las propiedades, los objetos y los metadatos del Sistema de Reportes deben mantenerse almacenados en una base de datos de PostgreSQL.

RNF 5.7. El Sistema de Reportes debe utilizar diferentes extensiones de procesamiento de datos para interactuar con distintos tipos de orígenes de datos.

RNF 5.8. El Sistema de Reportes debe permitir la conexión al gestor de bases de datos PostgreSQL como fuente de los datos para la generación del reporte.

RNF 6. Requisitos de Hardware

RNF 6.1. El Sistema de Reportes debe instalarse en un servidor que cuente al menos con los siguientes requisitos mínimos de hardware: procesador Intel Pentium 4 1.7 GHz o AMD similar, 512 MB de RAM, 40 GB de espacio en disco duro.

RNF 6.2. La base de datos del Sistema de Reportes debe instalarse en un servidor que cuente al menos con los siguientes requisitos de hardware: procesador Intel Pentium 4 1.7 GHz o AMD similar, 512 MB de RAM, 40 GB de espacio en disco duro.

RNF 6.3. Para utilizar el sistema los usuarios o clientes deberán conectarse desde una PC con un navegador Web estándar (Mozilla Firefox v 1.5 o superior) y requisitos mínimos de hardware: procesador Intel Pentium 4 1.7 GHz, o AMD similar, 256 MB RAM, 20 GB de espacio en disco duro.

2.3 Tamaño y Rendimiento

El rendimiento de una arquitectura para una aplicación de tipo cliente-servidor que utiliza una base de datos como método de permanencia de la información, depende mayormente del Sistema Gestor de Base de Datos (SGBD) utilizado y su configuración.

Gracias a las sofisticadas características de PostgreSQL como el Control de Concurrencia Multi-Versión (MVCC) para conseguir una mucho mejor respuesta en ambientes de grandes volúmenes de datos pues permite que mientras un proceso escribe en una tabla otros accedan a la misma tabla sin necesidad de bloqueos, la creación de puntos de recuperación, los espacios de tablas, replicación asincrónica, un avanzado planificador y optimizador de consultas y un completo sistema de registros para la tolerancia a fallos, el rendimiento y la integridad de los datos está asegurada permitiendo que

las bases de datos no tengan límite de tamaño, las tablas pueden ser de hasta 32 TB (TeraBytes) de información, se puede almacenar hasta 1.6 TB por tupla, 1 GB por campo, la cantidad de tuplas por tabla no tiene límites, el límite de columnas por tabla es de entre 250 y 1600 dependiendo del tipo de las columnas y se pueden crear tantos índices como se desee. El sistema cuenta con 32 tablas en un esquema, en la que la mayor de las tablas no excede las 12 columnas y el campo que más información recoge es un xml de tipo "text" con un tamaño estimado inferior a los 10 KB, por lo que PostgreSQL asegura un óptimo rendimiento para el sistema.

Otro punto que influye en el rendimiento de una aplicación es el lenguaje de programación en el que esté desarrollada, PHP ha demostrado que una de sus principales ventajas es precisamente la rapidez de su ejecución a pesar de ser un lenguaje interpretado y no compilado, contando además con la integración con el más difundido servidor web a nivel global, Apache2.

2.4 **Vistas de la Arquitectura**

La arquitectura es un conjunto organizado de elementos, se utiliza para especificar las decisiones estratégicas acerca de la estructura y funcionalidad del sistema, las colaboraciones entre sus distintos elementos y su despliegue físico para cumplir las responsabilidades bien definidas. Resultaría muy complejo una única representación de la arquitectura del sistema y poco útil para los involucrados porque tendría información irrelevante para la mayoría de éstos. Es por ello que se necesitan representaciones que contengan elementos de importancia para cierto grupo de involucrados.

Cada uno de ellos se describe de una manera más comprensible si se utilizan distintos modelos o vistas pues constituyen una descripción parcial de una misma arquitectura y es necesario que exista cierto solapamiento entre ellos puesto que todas las vistas deben ser coherentes entre sí dado que describen la misma cosa. Para la descripción de la arquitectura propuesta se utilizarán las 4+1 vistas descritas por Philippe Kruchten en su libro *"The 4+1 View Model o Software Architecture"* incluyendo además una Vista de Datos.

2.4.1 **Vista de Casos de Uso**

El Sistema Generador Dinámico de Reportes presenta como principal objetivo la generación de reportes de forma dinámica como su nombre indica, este proceso se garantiza mediante la interrelación entre varias aplicaciones.

La vista en cuestión contiene los casos de uso y los escenarios que abarcan el comportamiento más importante del sistema, se confecciona a través del diagrama de casos de uso del sistema que facilita la comprensión y asimilación para las personas que desconocen del proyecto. Para lograr esto se

necesita la realización de varios casos de uso con un peso priorizado desde el punto de vista arquitectónico. Se han dividido en paquetes para su organización, de forma que cada aplicación representa un paquete específico y recoge los casos de uso arquitectónicamente significativos.

- **Diseñador de Modelos**

El Diseñador de modelos colecciona todos los casos de uso que son utilizados para crear, editar, definir y publicar modelos semánticos a partir de un origen de datos seleccionados. Un modelo semántico no es más que una descripción de la estructura de la base del negocio. El diseño del reporte se realiza a partir de estos y no directamente de la base de datos lo que brinda una mayor seguridad al sistema además de evitar sobrecarga de la base de datos con continuas peticiones que pueden sobrecargarla e interrumpir el trabajo de otras personas que la estén utilizando. Dichos modelos son guardados en ficheros XML y almacenados en el Servidor de Reportes donde podrán se consultados cada vez que sea necesario.

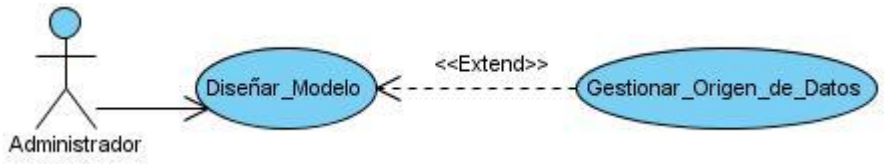


Figura 2. DCU Significativos del Diseñador de Modelos

Diseñar_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 Diseñador de Modelos se encarga de incluir tablas en un modelo a partir de un origen de datos válido para lo cual debe haberlas seleccionado antes.

Gestionar_Origen_de_Datos: permite adicionar un nuevo origen de datos de donde se extrae la información requerida. Además brinda la opción de modificar la configuración para la conexión a un origen de datos existente, así como eliminar un origen de datos que no se requiera su utilización siempre y cuando dicho origen a eliminar no este asociado a ningún reporte existente.

- **Diseñador de Consultas**

El Diseñador de Consultas es el encargado de crear relaciones mediante consultas a más de una tabla contenida en el/los modelos registrados en el sistema.

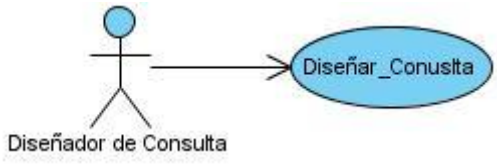


Figura 3. DCU Significativos del Diseñador de Consultas

Diseñar_Consulta: la consulta es construida mediante las tablas y/o vistas insertadas por el diseñador de consultas, las relaciones entre ellas, así como las condiciones y límites que establezca en la misma.

- **Diseñador de Reportes**

El Diseñador de reportes organiza los casos de uso encargados de crear, diseñar y almacenar los reportes en el servidor. Cuenta con las funcionalidades necesarias para diseñar los reportes de manera interactiva, siendo posible visualizar cómo va quedando el informe. Permite crear diferentes tipos de reportes dependiendo de las plantillas existentes. Los reportes son basados en un fichero XML que contiene su definición. Este XML es vital para el funcionamiento del sistema ya que comunica las diferentes aplicaciones y permite que éstas se integren conformando finalmente el reporte.

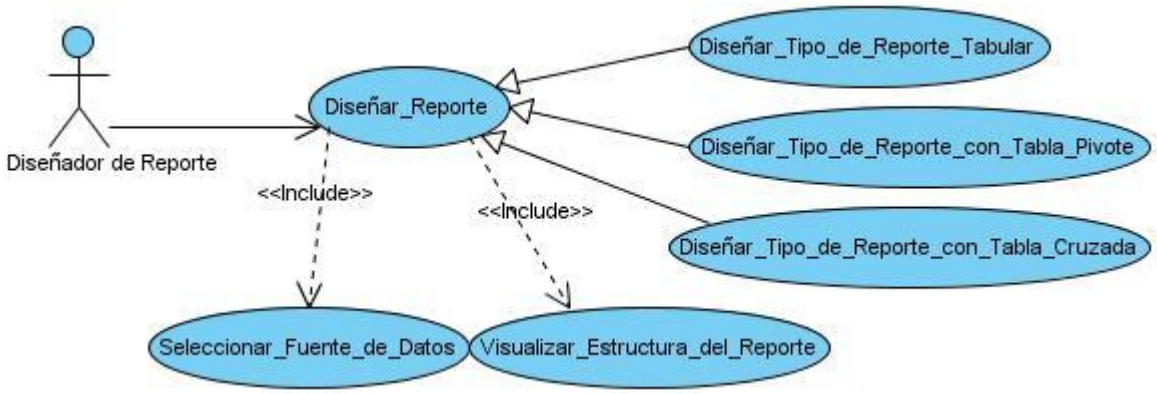


Figura 4. DCU Significativos del Diseñador de Reportes

Diseñar_Reporte: el presente caso de uso permite al actor realizar el diseño de reportes tabulares, con tablas cruzadas o tablas pivotes. El mismo ha de estar compuesto por campos asociados a un modelo registrado en el sistema. El reporte puede ser visualizado en formato HTML. El módulo permite salvar y cargar reportes y plantillas. Posibilita que se le incorporen componentes al reporte como: líneas, imágenes, gráficos, entre otros y sean posicionados en las diferentes partes que componen el reporte (encabezado, cuerpo o pie del documento). Una vez ubicado un componente se puede personalizar con la definición de propiedades de estilos (color, tipo de letra, alineación, entre otros).

Seleccionar_Fuente_de_Datos: el caso de uso se inicia cuando el caso de uso base invoca la

construcción de la interfaz de selección de los datos asociados a un modelo. Del modelo se selecciona una consulta, tabla o procedimiento y de cada uno de ellos se seleccionan los campos de interés para el actor. El caso de uso retorna una fuente de datos asociada a un origen de datos terminando de esta forma.

Visualizar_Estructura_del_Reporte: permite actualizar la estructura del reporte ("Inspector" de reporte) o actualizar los valores de los elementos contenidos en el reporte mediante la "Tabla de propiedades".

- **Administrador de Reportes**

El Administrador de Reportes empaqueta todos los procesos administrativos que se utilizan para gestionar los informes vía web. Se puede usar para la búsqueda de las categorías y reportes a utilizar. Éstas se pueden gestionar dependiendo del rol del usuario. La seguridad basada en roles es uno de los aspectos más importantes en el Administrador de Reportes. Dentro de las tareas que se administran se encuentran las siguientes: gestionar reportes, gestionar categorías, configurar la seguridad, crear la programación y entrega de reportes, entre otras.

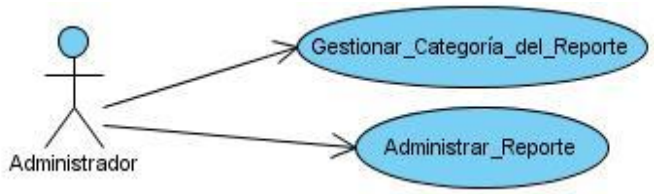


Figura 5. DCU Significativos del Administrador de Reportes

Administrar_Reporte: permite mostrar un listado de todos los reportes y plantillas que han sido creados. Además proporciona la posibilidad de eliminar un reporte o plantilla determinada, así como mover un reporte de categoría o copiarlo en otra categoría.
Gestionar_Categoría: posibilita la adición, eliminación y modificación de categorías que es una forma para organizar los reportes en el sistema.

- **Visor de Reportes**

Agrupa los casos de uso involucrados en el proceso de visualización de los reportes en los diferentes formatos de salida, dando la posibilidad al usuario de su impresión, vista previa, filtrado, entre otras.

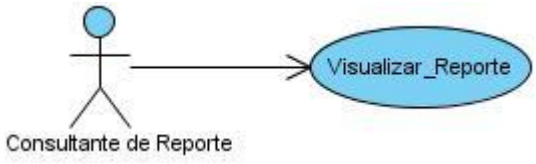


Figura 6. DCU Significativos del Visor de Reportes

Visualizar_Reporte: luego de la selección de un reporte previamente creado, posibilita la visualización de una vista previa del mismo y pueden ser exportados en diferentes formatos: HTML, PDF, EXCEL, CSV. También permite buscar un reporte específico en la lista de reportes.

2.4.2 **Vista Lógica**

La vista lógica contiene las clases del diseño más importantes, organizadas por paquetes y subsistemas en capas de trabajo. Es representada por uno o varios diagramas de clases que son un subconjunto del modelo de diseño. La arquitectura lógica debe tener la estructura necesaria para que sea lo suficientemente flexible en la arquitectura física que se utilizará. Esta vista se presenta a través de tres niveles de arquitectura, cada uno de los cuales corresponde a un refinamiento del anterior. El primer nivel describe el estilo arquitectónico de la solución, el segundo nivel especifica la composición en módulos y la relación entre ellos, el tercer nivel es el que presenta mayor detalle, pues describe la estructura de los módulos en forma de diagramas de clases del diseño.

- **Visión General de la Arquitectura**

Dada las características del ambiente de despliegue del sistema, así como los objetivos y restricciones arquitectónicas mencionadas con anterioridad se requiere el uso y la combinación de diferentes estilos arquitectónicos para su implementación. El estilo Cliente Servidor al tratarse de una aplicación web se evidencia con un servidor de bases de datos (PostgreSQL) un servidor web (con función de servidor de aplicaciones, Apache 2) y varios clientes que acceden al sistema a través de un navegador, siendo tanto el Cliente como el Servidor sistemas desacoplados no dependientes.

Symfony toma lo mejor de la arquitectura MVC y la implementa de forma que el desarrollo de aplicaciones sea rápido y sencillo. La arquitectura del sistema está organizada en 6 capas, que constituye una variante del estilo 3 capas, pues se agrega el modelo de datos que normalmente se encuentra en un servidor de bases de datos o la capa lógica se divide en dos explorador y servidor web. La Figura 7 muestra la arquitectura lógica de 6 capas del sistema.

Las capas de la 2 a la 5 reúnen toda la lógica del negocio y tiene embebido otro estilo arquitectónico el MVC, que es el que provee Symfony como framework a utilizar en el desarrollo del sistema para que

sea rápido y sencillo. En la solución la Vista brindada por Symfony no se utiliza como una interfaz para la comunicación e intercambio de información con la capa de Presentación sino en su lugar se utiliza el framework ExtJS para crear las interfaces, el Modelo (Abstracción y Acceso a Datos) gestiona los datos que necesita el Controlador para ejecutar la lógica del negocio, se utiliza Propel como ORM para acceder de forma efectiva a la base de datos desde un contexto orientado a objetos. Es importante destacar que el sistema utiliza una variante del MVC, en este caso, el patrón Controlador Frontal. Consiste en la existencia de un único controlador en el sistema el cual soluciona el problema de la descentralización presentes en el patrón MVC. Es el único punto de entrada a la aplicación, carga la configuración y determina la acción a ejecutarse. La utilización de este patrón nos permite centralizar diferentes aspectos del sistema como son el tratamiento de excepciones, el tratamiento de la configuración, las peticiones y respuestas del servidor web, la persistencia de los datos, entre otros.

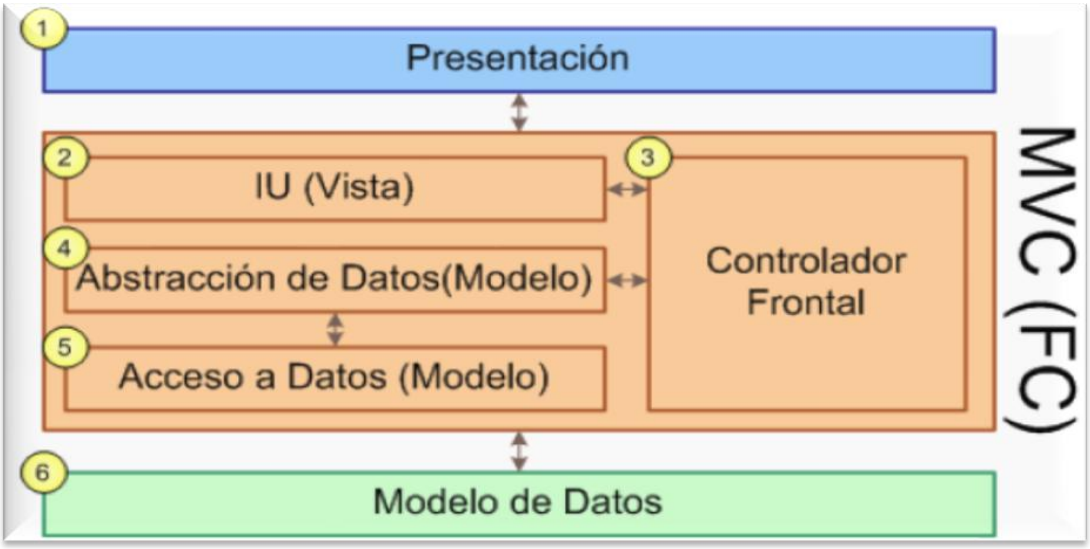


Figura 7. Estructura en Capas del Sistema

1. La capa de Presentación es la encargada del manejo de las interfaces para la interacción con el sistema. Está conformada sobre los frameworks ExtJS 3.3 y OpenJacob para el desarrollo de aplicaciones en JavaScript, lo que le aporta mayor usabilidad y productividad en el desarrollo.

2. Esta capa incluye la lógica para decidir que debe ver el usuario, los caminos de navegación y cómo interpretar las entradas. Es la encargada de interactuar con la Presentación y suministrarle los datos necesarios a visualizar. Es importante destacar que esta capa forma parte del patrón MVC utilizado en la lógica del negocio y representado en la Figura 7, representa la Vista, la cual se encarga de la interacción con el Controlador.

3. El 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.
4. La capa de Abstracción de Datos incluye las entidades del negocio mapeadas como objetos, dentro del MVC representa una parte del Modelo. En la solución se utiliza Propel como ORM.
5. La capa de Acceso a Datos es la encargada de interactuar con la capa Modelo de Datos para recibir, insertar, actualizar y eliminar información. Es importante destacar que la Capa de Acceso a Datos no es la encargada de administrar o almacenar los datos, esta meramente provee la interfaz entre la lógica del negocio y la base de datos. Se utiliza PDO para la generación de esta capa y para los accesos al Modelo de Datos.
6. El Modelo de Datos es la capa encargada de la creación, recepción, actualización y eliminación de los datos de forma física, soportado en la solución por PostgreSQL 8.3.

Conectores

Los conectores permiten la comunicación entre los distintos componentes o elementos definidos en el estilo arquitectónico, los conectores que se utilizaron para la comunicación en ambos sentidos por cada una de las capas son:

Presentación - Negocio

La relación que se establece entre la capa de Presentación y la capa subyacente se realiza a través de una solicitud AJAX. El acceso a los datos se realiza mediante el objeto XMLHttpRequest que actúa como una interfaz empleada para realizar las peticiones HTTP al servidor web. Para los datos transferidos se utiliza una codificación basada en texto, en este caso: JSON o XML según corresponda la solicitud.

Negocio (Acceso a Datos) – Modelo de Datos

PHP Data Objects (PDO) es una extensión que provee una capa de abstracción de acceso a datos para PHP 5 como paquete de abstracción, constituye una interfaz para la comunicación con la base de datos completamente orientada a objetos. Se utiliza TCP/IP como protocolo de comunicación.

➢ Elementos significantes del modelo arquitectónico

En la Figura 8 se muestra un diagrama con los diferentes módulos presentes y las relaciones de dependencia. Los casos de uso correspondientes a cada subsistema se ven realizados en cada uno de estos módulos, lo que le brinda a la aplicación mayor flexibilidad y mantenibilidad, posibilitando realizar cambios localizados en el módulo que lo requiera, sin afectar el resto.

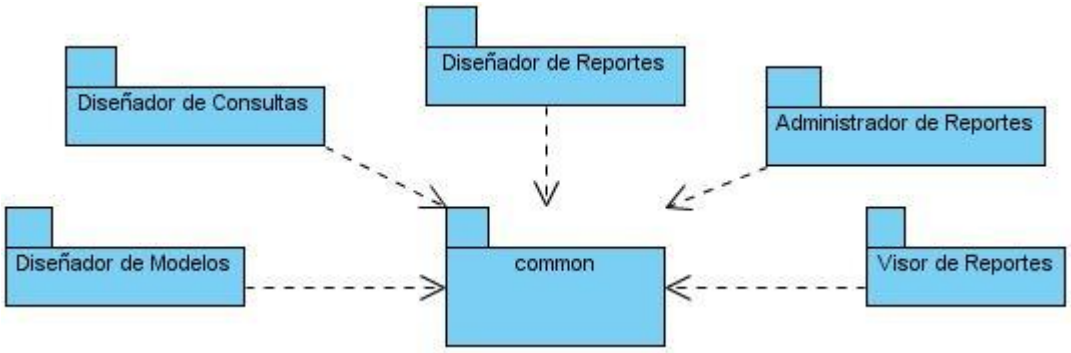


Figura 8. Diagrama de Módulos del Sistema

Cada uno de los módulos tiene una trazabilidad directa con los paquetes definidos en la Vista de Casos de Uso, pues ellos son los encargados de la realización de los casos de uso pertenecientes a cada paquete.

A continuación se describe cada módulo y la realización de los casos de uso que realiza mediante un diagrama de las clases del diseño significativas:

Diseñador de modelos: contiene la realización de los casos de uso que permiten la creación de modelos y orígenes de datos para la conformación del modelo semántico de la base de datos.

Como se puede observar en la Figura 9 las clases presentes en la capa de presentación están estereotipadas como Javascript, en todos los diagramas de clases de cada módulo se encuentran en esta capa la inclusión de los frameworks representados por los ficheros ext_all.js, ext_base.js y draw2d.js. El fichero layout_browser.js es el controlador de la capa de presentación pues determina que fichero js debe cargar en consecuencia con la solicitud recibida, en este módulo incluye el main.js que es la fachada y carga los ficheros dataSource.js, models.js y queryBuilder_main.js que representan las clases con las funcionalidades para la construcción de la interfaz de usuario. El main.js realiza las peticiones AJAX al controlador, que contiene las acciones necesarias para la ejecución de los casos de uso y para ello interactúa con el paquete de la lógica de negocio y con el Modelo, específicamente con la entidad Model y Datasource.

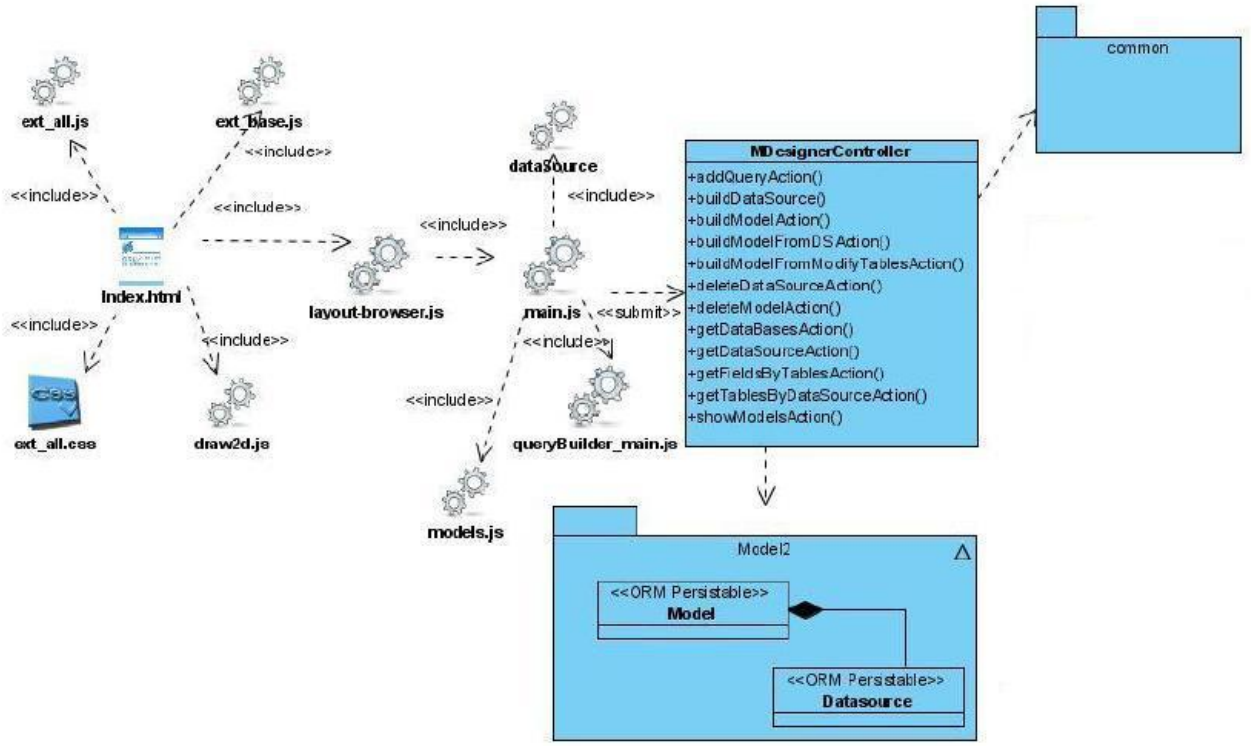


Figura 9. Diagrama de Clases Diseñador de Modelos

Diseñador de reportes: recoge la realización de los casos de uso principales del sistema, que permiten el diseño, generación y publicación de los reportes.

En la Figura 10 se representa el diagrama de clases para la realización de los casos de uso relacionados con el Diseñador de reportes. El fichero Factory.js se encarga de la construcción del workflow.js que envía las peticiones al controlador. Las acciones invocadas por el controlador se encargan de la creación del reporte, la vista previa, guardar plantilla, entre otras. Las entidades utilizadas en la realización de los casos de uso son Category, que persiste la categoría a la cual pertenece un reporte, Report, guarda el XML de definición del reporte y Template, se refiere a las plantillas que pueden estar asociadas a algunos reportes.

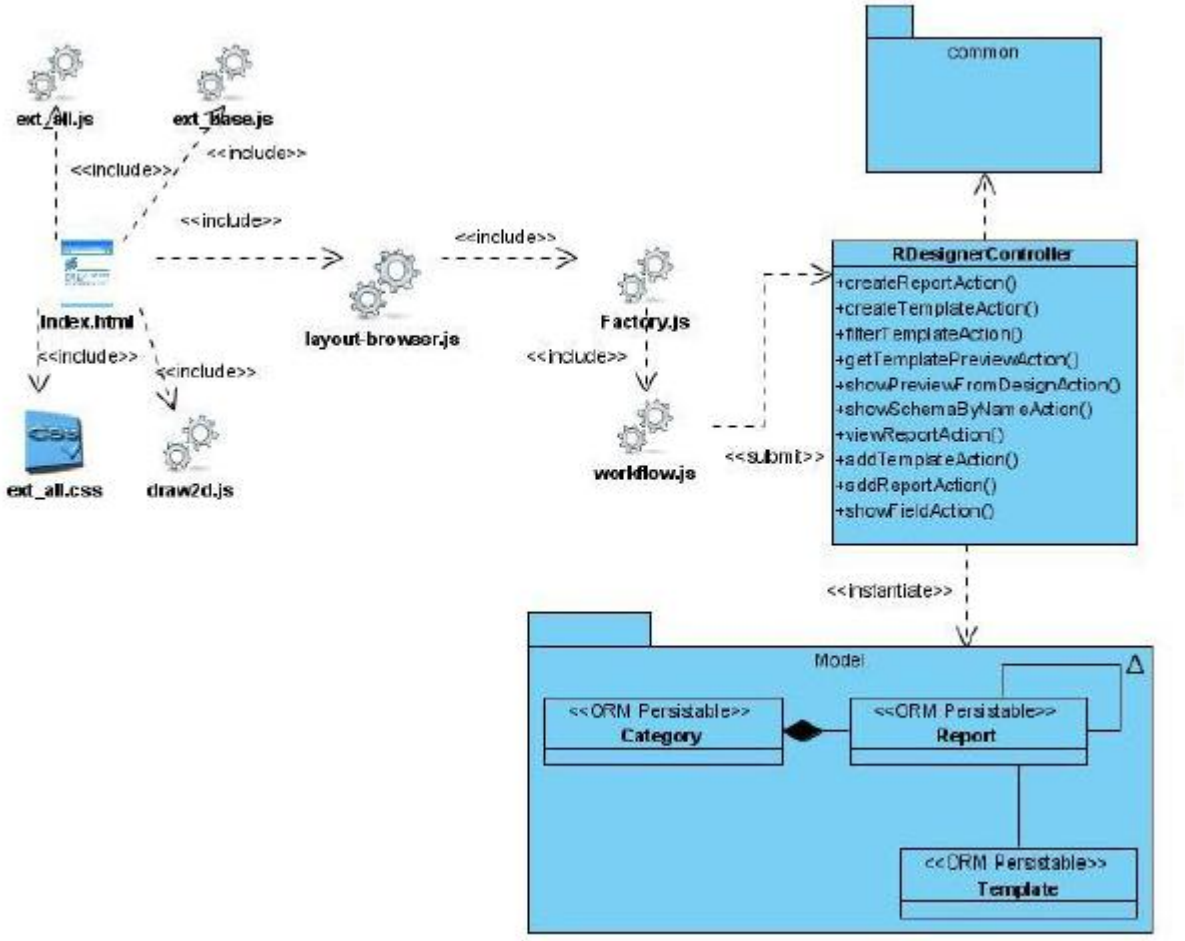


Figura 10. Diagrama de Clases Diseñador de Reportes

Visor de reportes: contiene la realización de los casos de uso que permiten la visualización de los reportes, así como las interfaces y servicios que brinda para ser usados por terceros.

Una vez que el controlador de la capa de presentación, el layout_browser.js, recibe la petición de la creación de las interfaces de este módulo, carga la clase principal main.js que invoca a report.js, parameters.js y report_viewer.js encargados de construir las interfaces correspondientes y realizar las peticiones necesarias al controlador según las necesidades del usuario, que pueden ser crear el reporte, exportarlo a los diferentes formatos o filtrarlo. El controlador del negocio recibe dichas solicitudes, ejecuta las acciones representadas en la Figura 11 en la clase RViewerController, que para su realización requiere la interacción con las clases del modelo, los objetos de la lógica de negocio y funciones y librerías comunes que se encuentran en el paquete common.

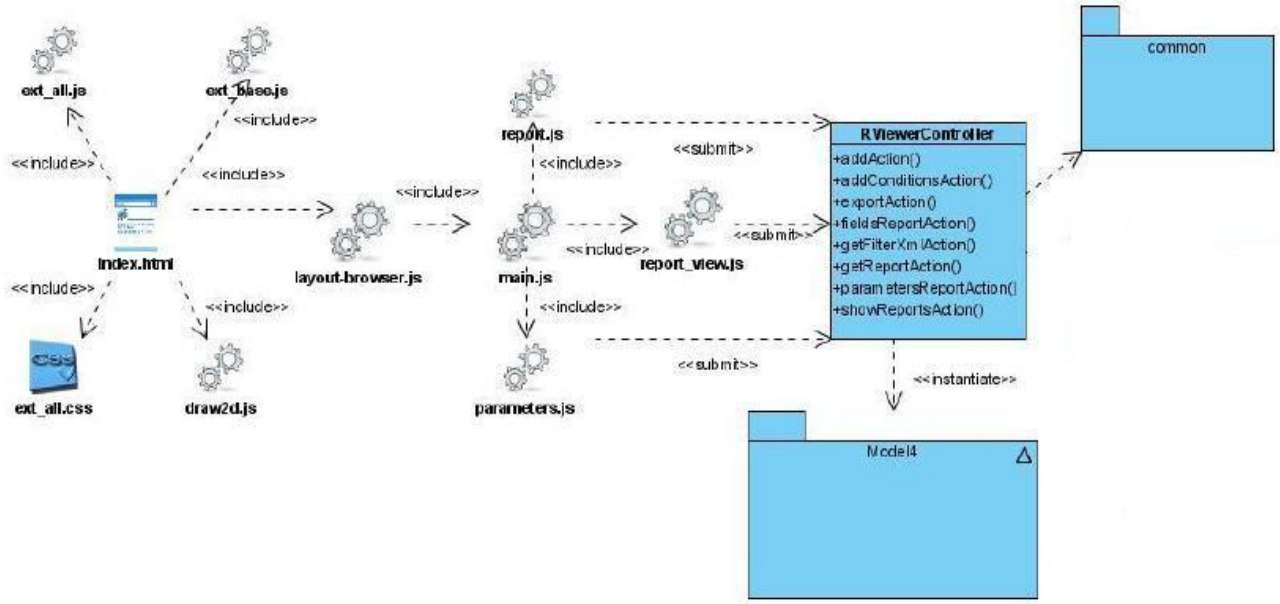


Figura 11. Diagrama de Clases Visor de Reportes

Administrador de reportes: permite la realización de los casos de uso pertenecientes a la gestión de usuarios, reportes, modelos, consultas y programaciones.

Las peticiones enviadas por la capa de presentación mediante solicitudes AJAX, principalmente generadas por report_manager.js son atendidas por el controlador el cual ejecuta una serie de acciones que le dan funcionalidad al módulo, dichas acciones están encaminadas principalmente a la creación y eliminación de los reportes, modelos y categorías, además de la programación de tareas en el servidor de reportes. Se extrae información del modelo específicamente de las entidades Category, Subscription y Schedule para la realización de los casos de uso.

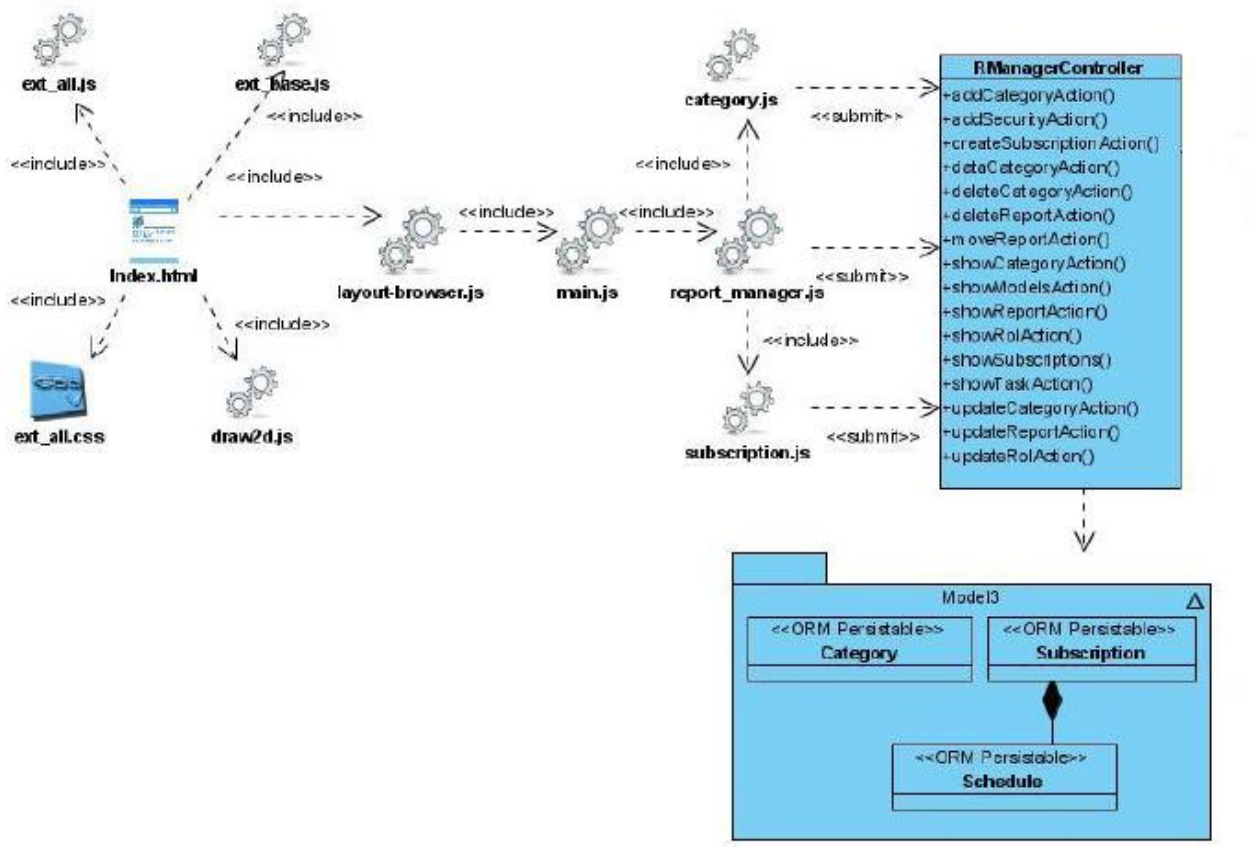


Figura 12. Diagrama de Clases Administrador de Reportes

2.4.3 **Vista de Despliegue**

El Sistema Generador Dinámico de Reportes está diseñado para que cada organización instale una instancia del sistema en sus servidores puesto que el mismo puede manejar información delicada y que lo utilice de acuerdo a sus necesidades teniendo en cuenta los niveles de integración previstos. El sistema puede ser instalado de forma independiente con acceso directo a través de un navegador, como sistema embebido dentro de otro, como aplicación independiente pero con acceso a los reportes desde otra aplicación o sistema, entre otros.

- **Instalación como sistema independiente**

En este caso se necesitan dos servidores web uno como entorno de ejecución con Apache2 con soporte para PHP 5.3.3 y sistema operativo basado en GNU/Linux en cualquiera de sus distribuciones (preferentemente Debian o Ubuntu), y el otro servidor como entorno de ejecución con Apache Tomcat para la instalación de Jasper Report 4.0, esto es debido a que la aplicación está desarrollada en PHP

pero la librería para la generación de reportes utilizada(Jasper Report) está desarrollada en Java por lo que se hace necesario vincular PHP/Java mediante servicios utilizando SOAP como protocolo de comunicación entre ambos servidores para el intercambio de información. La base de datos puede estar instalada en el mismo servidor Apache2 o en uno distinto en dependencia de las posibilidades de la organización, siendo necesario PostgreSQL, versión 9. Los usuarios podrán conectarse a la aplicación desde estaciones clientes e incluso desde el mismo servidor a través del navegador web Mozilla Firefox o cualquier otro que utilice el motor Gecko como es el caso de Epiphany Web Browser del escritorio Gnome (Ver Figura 13).

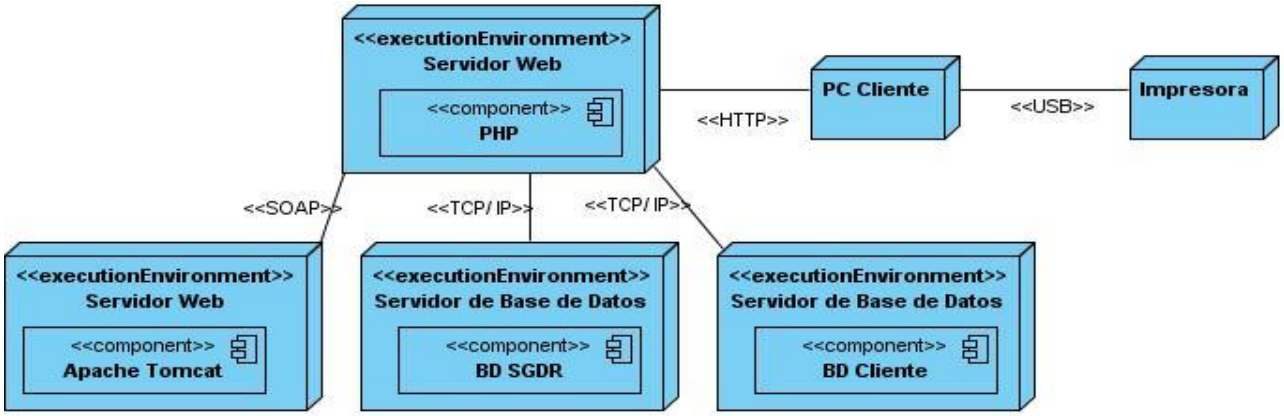


Figura 13. Diagrama de Despliegue del Sistema Independiente

- **Instalación como sistema embebido**

En este escenario se puede ejecutar la aplicación en el mismo servidor que el sistema externo o en uno diferente en dependencia de las posibilidades de la entidad ya que la integración se hace a nivel de URL. Se hace necesario en caso de ejecutarse ambas en el mismo servidor que se configurase un host virtual independiente para el SGDR, debido a restricciones establecidas por el funcionamiento de Symfony. (Ver Figura 14)

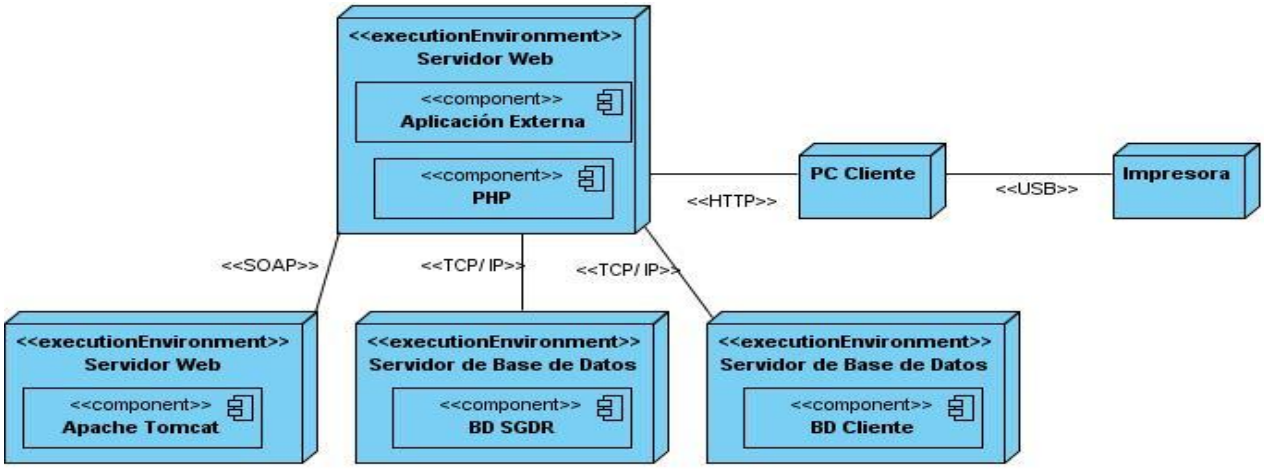


Figura 14. Diagrama de Despliegue del Sistema Embebido

- **Instalación como sistema independiente con acceso a los reportes desde otra aplicación o sistema**

Siguiendo este entorno es posible desde otra aplicación o sistema hacer una petición a través de una URI y obtener los datos necesarios para representar un reporte e incluso establecer filtros y parámetros a la obtención de los datos. (Ver Figura 15)

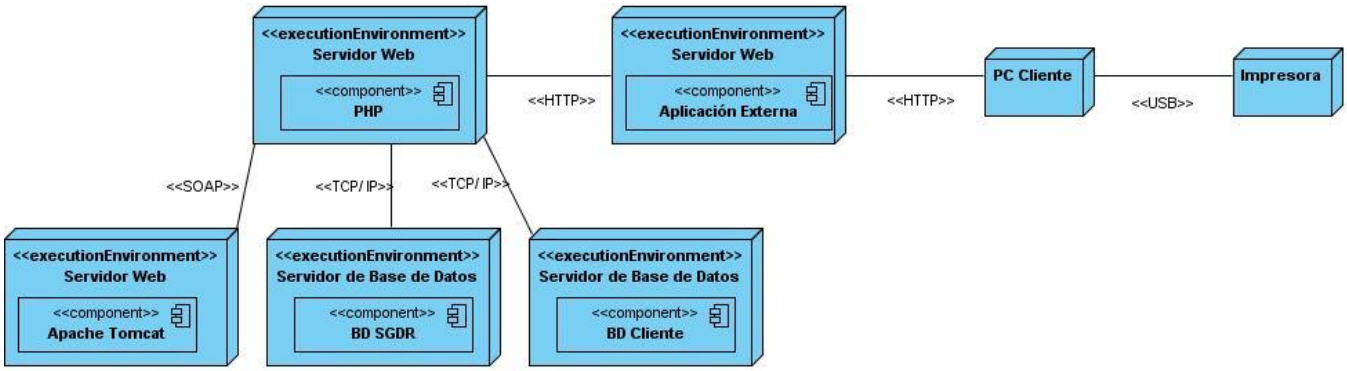


Figura 15. Vista de Despliegue del Sistema Independiente con Acceso Remoto a Reportes

2.4.4 **Vista de Implementación**

La vista de implementación se centra en la organización real de los módulos de software en el ambiente de desarrollo de software. El software se empaqueta en partes pequeñas (bibliotecas de programas o subsistemas) que pueden ser desarrollados por uno o un grupo pequeño de

desarrolladores. Los subsistemas se organizan en una jerarquía de capas, cada una de ellas brinda una interfaz estrecha y bien definida hacia las capas superiores.

Symfony como framework a utilizar organiza el código fuente en una estructura de tipo proyecto y almacena los archivos del proyecto en una estructura estandarizada de tipo árbol. Dentro de un proyecto, las operaciones se agrupan de forma lógica en aplicaciones. Cada aplicación está formada por uno o más módulos. Un módulo normalmente representa a una página web o a un grupo de páginas con un propósito relacionado. Los módulos almacenan las acciones, que representan cada una de las operaciones que se puede realizar en un módulo (18).En el Anexo 1 se muestra la estructura física del sistema, con la distribución por módulos y acciones significativas que propone Symfony.

La distribución de cada componente de implementación por cada capa de la arquitectura y las relaciones de dependencia que se establecen entre éstos se muestra el Anexo 2. A continuación se describen detalladamente los artefactos más importantes de los subsistemas desde el punto de vista de su implementación.

- **SGDR_Presentación**

El principal componente de esta capa es el framework ExtJS que junto a un conjunto de ficheros y librerías JavaScript conforman la presentación del sistema a los usuarios finales. En la Figura 16 se encuentra el paquete js que agrupa los componentes JavaScript y las relaciones de dependencia entre ellos. Los componentes common_components y common_functions agrupan aquellos ficheros que son reutilizados por los diferentes componentes representados por report_generator; a su vez, este último específicamente los componentes query_builderjs y report_designerjs utilizan el framework OpenJacob, no han sido incluidos en la figura para mayor claridad. Los paquetes images y css representan los recursos de imágenes y hojas de estilo respectivamente, necesarios en la presentación de los JavaScript.

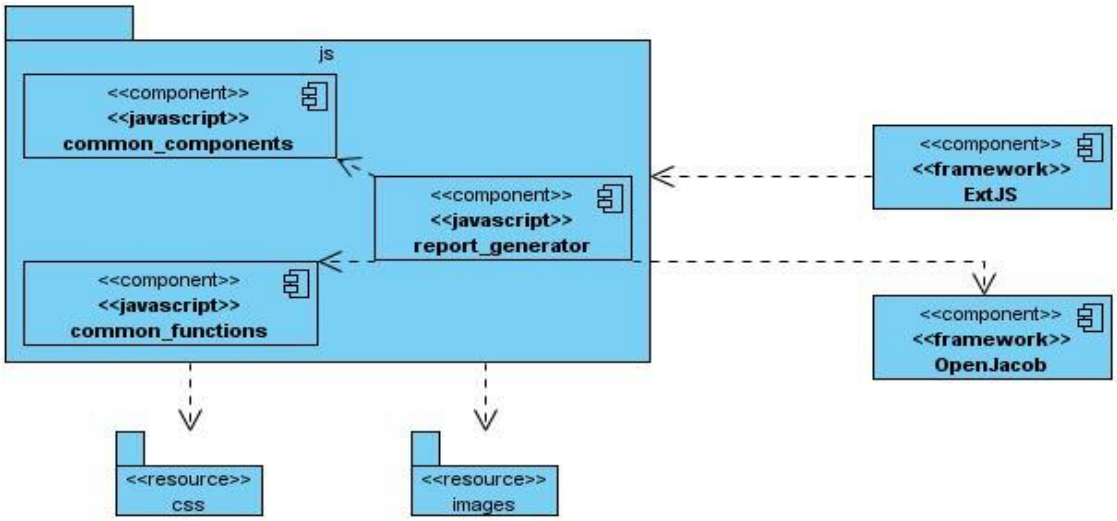


Figura 16. Diagrama de Componentes Capa de Presentación

- **SGDR_Negocio**

Los componentes de esta capa se muestran en la Figura 17, en la que se encuentra el paquete lib que recoge las librerías Jasper Report (motor para la generación de los reportes), JFreeChart (para la generación de gráficos), iText (para exportación a pdf) y el framework Symfony necesarios en la ejecución de las diferentes aplicaciones que representan los ficheros .php con las acciones y objetos del negocio que le dan funcionalidad al sistema.

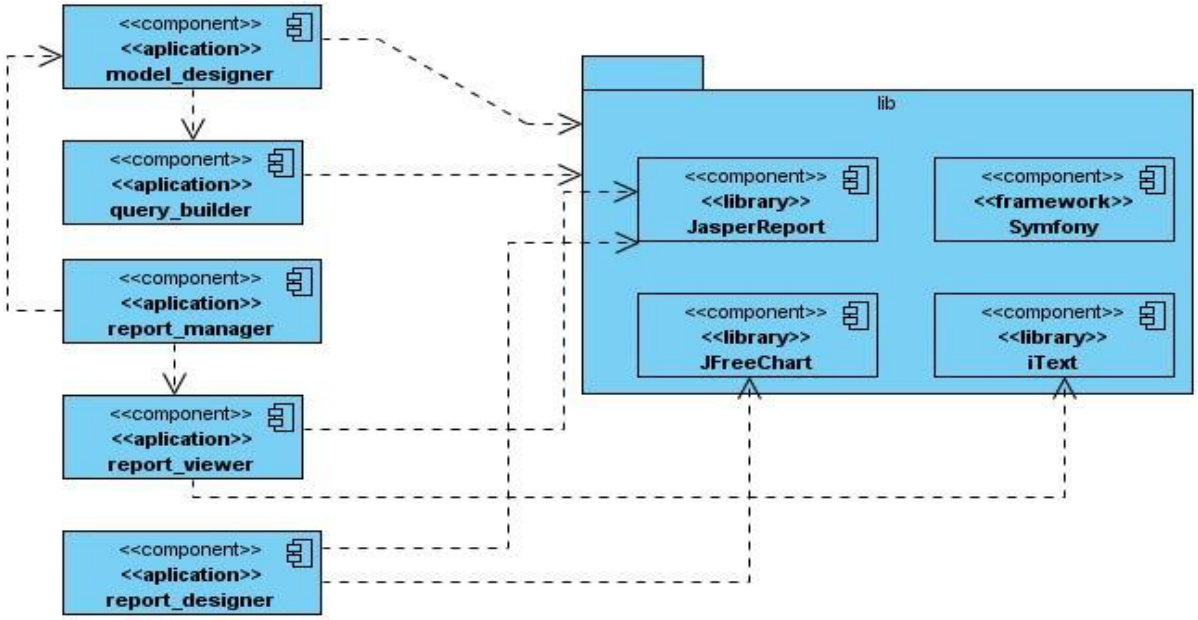


Figura 17. Diagrama de Componentes Capa de Negocio

- **SGDR_AD**

El componente om (*object model*) representado en la Figura 18 contiene las clases base del modelo, generadas una vez que se analiza el esquema de la base de datos. El componente model representa el conjunto de clases de objetos que se encargan del acceso a datos. Con respecto a map, contiene meta información relativa a las tablas que son necesarias para la ejecución de la aplicación.

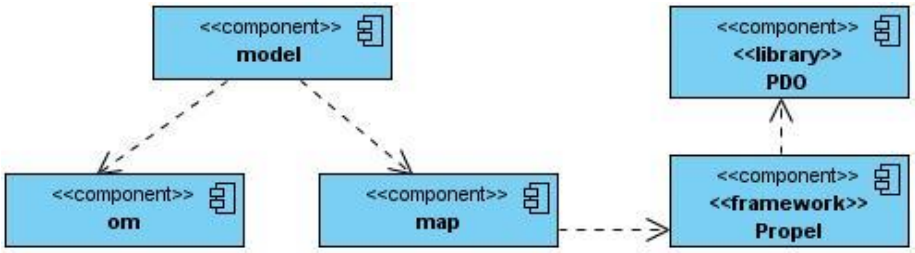


Figura 18. Diagrama de Componentes Capa de Acceso a Datos

2.4.5 **Vista de Datos**

Esta vista presenta el modelo de datos utilizado con la descripción de las tablas más importantes que componen la base de datos del SGDR. La siguiente figura muestra la estructura del modelo de datos empleado para la persistencia de las entidades del sistema, posteriormente se explica en detalles la descripción de cada tabla, agrupadas por módulos para su mejor entendimiento. (Ver Figura 19)

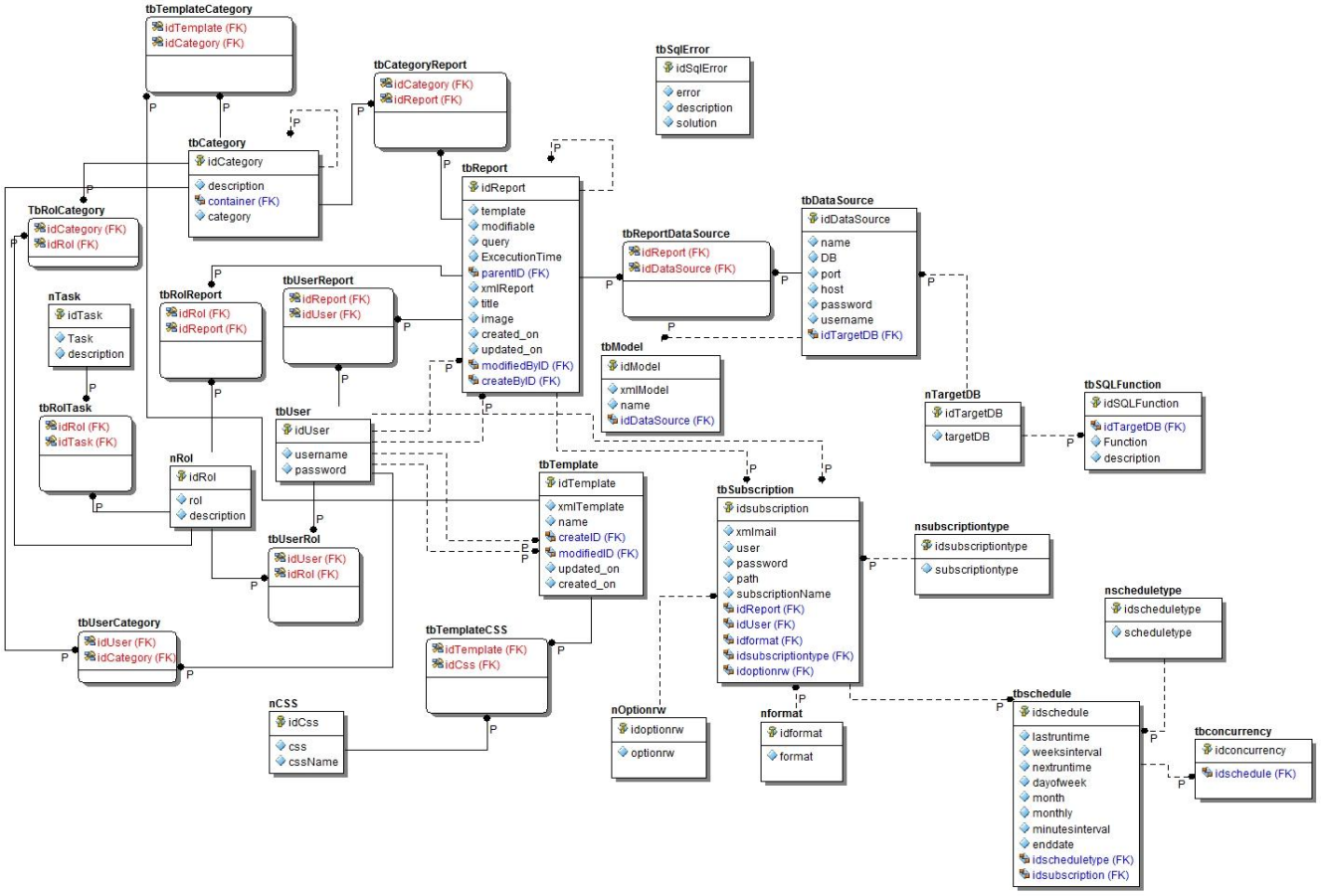


Figura 19. Modelo de Datos del Sistema

➢ Descripción de los Elementos del Modelo de Datos

Se identifican varios subsistemas de entidades que responden a funciones específicas para cada uno de los módulos del sistema:

Subsistema Diseñador de modelos: representa el conjunto de entidades que contienen la información que define la estructura de los modelos semánticos y la fuente de datos relacionadas a ellos así como la información almacenada a partir del Diseñador de Consultas.

idModel: almacena la definición de los modelos semánticos creados.

idDataSource: almacena la información de los orígenes de datos sobre los cuales están creados los modelos semánticos.

idTargetDB: tabla nomencladora que almacena el listado de los gestores de bases de datos soportados para crear orígenes de datos.

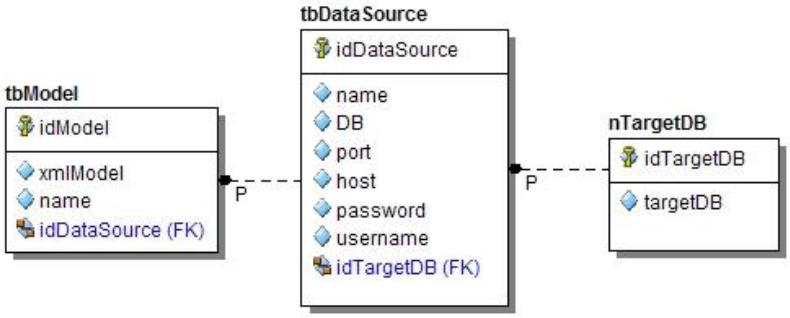


Figura 20. Modelo de Datos. Diseñador de Modelos

Subsistema Diseñador de Reportes: está compuesto por las entidades que guardan información sobre los reportes creados.

tbReport: almacena la definición de los reportes diseñados.

tbTemplate: contiene la definición de las plantillas diseñadas.

tbCategory: almacena las categorías creadas a las que se asociarán los reportes.

tbTemplateCSS: almacena las css que serán aplicadas a los reportes en su diseño.

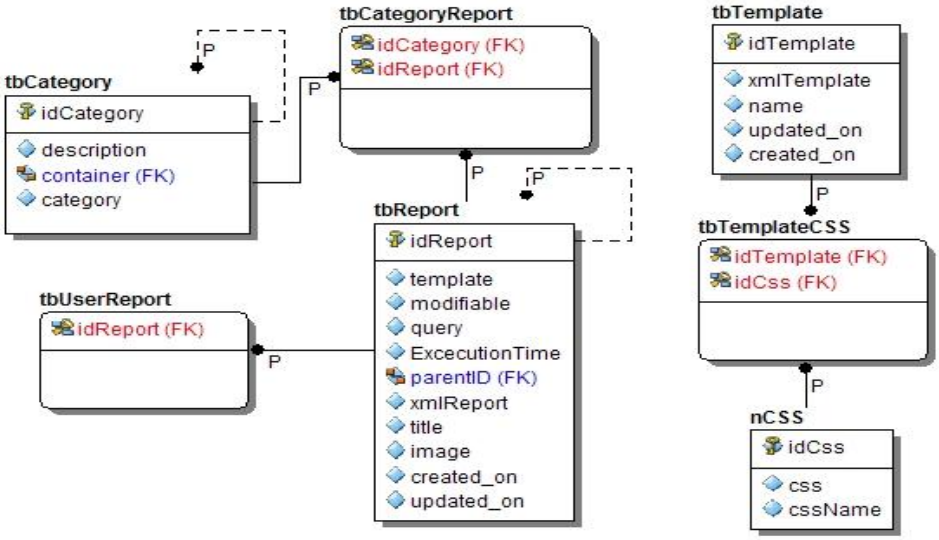


Figura 21. Modelo de Datos. Diseñador de Reportes

Subsistema Visor de Reportes: está compuesto por las entidades que guardan información sobre los reportes creados en este caso organizados por categoría.

idReport: almacena la definición de los reportes diseñados.

idCategory: almacena la información de los reportes organizándolos por categoría.

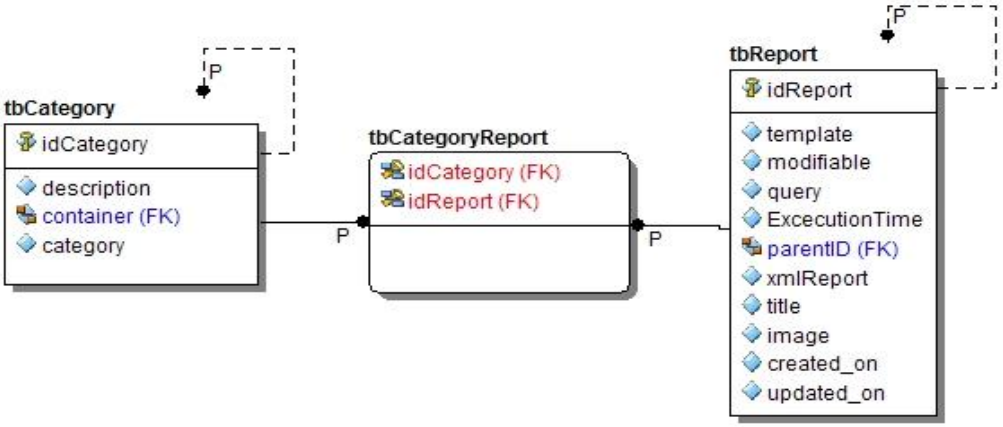


Figura 22. Modelo de Datos. Visor de Reportes

Subsistema Administrador de reportes: presenta las entidades relacionadas con las suscripciones y programación de entrega de los reportes, información sobre las categorías y los usuarios creados.

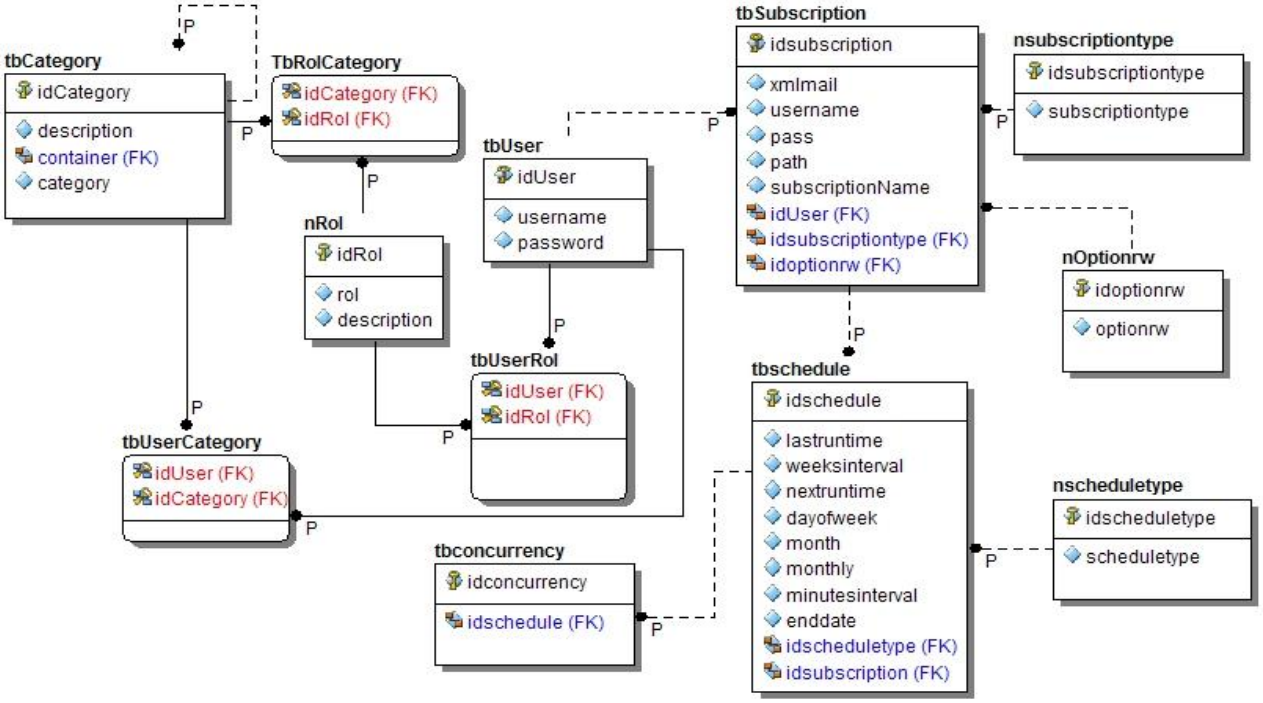


Figura 23. Modelo de Datos. Administrador de Reportes

tbSubscription: almacena la información de las suscripciones para entrega de reportes.

tbSchedule: almacena la información de las tareas programadas.

tbCategory: almacena las categorías creadas a las que se asociarán los reportes.

tbUser: almacena las cuentas de usuarios que accederán al sistema.

nRol: tabla nomencladora que almacena el listado de roles que serán asignados a los distintos usuarios.

2.5 Conclusiones del Capítulo.

En el presente capítulo fue descrita la arquitectura del sistema haciendo uso de la representación de las vistas arquitectónicas propuestas por Philippe Kruchten en su libro *"The 4+1 View Model o Software Architecture"* incluyendo además una Vista de Datos teniendo en cuenta los objetivos y restricciones propuestas, en resumen:

- Se identificaron los estilos y patrones arquitectónicos presentes en la solución.
- Se realizó una descripción del estilo multicapas como resultado de la aplicación en la arquitectura del sistema, estableciendo los componentes involucrados así como los conectores entre cada capa.
- Se fundamentaron los componentes que soportan el estilo Modelo-Vista-Controlador y se establecieron las librerías y frameworks a utilizar.

CAPÍTULO 3: Evaluación de la Arquitectura

Introducción

La arquitectura de los sistemas a ser construidos se convierte en un factor de suma importancia para lograr que éste tenga un alto nivel de calidad porque poseer una buena arquitectura de software es muy importante, debido a que ésta es el corazón de todo sistema informático y determina cuales serán los niveles de calidad asociados al sistema. (19)

No sirve de nada un sistema que no cumple con los atributos de calidad que se especificaron en los requerimientos no funcionales de los clientes. Por lo que diseñar una correcta arquitectura va a determinar el éxito o fracaso de un sistema de software, en la medida que esta cumpla o no con sus objetivos. (19) Debido a esto "Para reducir tales riesgos, y como buena práctica de ingeniería, es recomendable realizar evaluaciones a la arquitectura". (20)

El objetivo de evaluar una arquitectura es saber si puede habilitar los requerimientos, atributos de calidad y restricciones para asegurar que el sistema a ser construido cumple con las necesidades de los stakeholders. (21)

La evaluación de una arquitectura de software es una tarea no trivial, puesto que se pretende medir propiedades del sistema en base a especificaciones abstractas, como por ejemplo los diseños arquitectónicos. Por ello, la intención es más bien la evaluación del potencial de la arquitectura diseñada para alcanzar los atributos de calidad requeridos. (22)

Después de una evaluación de una arquitectura de software, se pueden tomar algunas decisiones como: si se puede seguir el proyecto con las áreas de debilidad dadas en la evaluación o si hay que reforzar la arquitectura de software o si hay que comenzar de nuevo toda la arquitectura de software.

3.1 **Atributos de Calidad**

La calidad del software se define como el grado en el cual el software posee una combinación deseada de atributos. Los cuales constituyen requerimientos del sistema que responden a las características que el sistema debe satisfacer y es lo que se conoce como atributo de calidad.

Un atributo de calidad es una característica de calidad que afecta a un elemento. Donde el término "**característica**" se refiere a aspectos no funcionales y el termino "**elemento**" a componente (20). A grandes rasgos Bass establece la clasificación de los atributos de calidad en dos categorías: atributos observables vía ejecución y los no observables en vía de ejecución.

El modelo de calidad expuesto en la norma ISO/IEC 9126 categoriza los atributos de calidad en seis características (Funcionalidad, Fiabilidad, Usabilidad, Eficiencia, Mantenibilidad y Portabilidad), cada

una de estas se dividen a su vez en subcaracterísticas que podrán ser medidas a través de métricas internas y externas.

3.2 **Técnicas de Evaluación**

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

Tabla 2. Instrumentos Asociados a las Técnicas de Evaluación de Arquitectura de Software

Técnicas de Evaluación	Instrumentos de Evaluación
Basada en Escenarios	• Profiles • Utility Tree
Basada en Simulación	• ADLs • Modelos de colas
Basada en Modelos Matemáticos	• Cadenas de Markov • Reliability y Block Diagrams
Basada en Experiencia	• Intuición y experiencia • Tradición • Proyectos similares

A continuación se explica la técnica basada en escenarios por ser la empleada en el método escogido.

3.2.1 **Evaluación Basada en Escenarios**

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

Las técnicas basadas en escenarios cuentan con dos instrumentos de evaluación relevantes: el Utility Tree propuesto por Kazman y los Profiles, propuestos por Bosch. En este caso se explicará el Utility Tree por ser la empleada en la evaluación de la arquitectura.

Utility Tree

Es un esquema en forma de árbol que presenta los atributos de calidad de un sistema de software, refinados hasta el establecimiento de escenarios que especifican con suficiente detalle el nivel de prioridad de cada uno (23). La intención del uso del Utility Tree es la identificación de los atributos de calidad más importantes para un proyecto particular. El Utility Tree contiene como nodo raíz la utilidad general del sistema. Los atributos de calidad asociados al mismo conforman el segundo nivel del árbol los cuales se refinan hasta la obtención de un escenario lo suficientemente concreto para ser analizado y otorgarle prioridad a cada atributo considerado. Cada atributo de calidad perteneciente al árbol contiene una serie de escenarios relacionados, y una escala de importancia y dificultad asociada, que será útil para efectos de la evaluación de la arquitectura.

3.3 Método ATAM

ATAM (*Architecture Trade-off Analysis Method*): está inspirado en tres áreas distintas: los estilos arquitectónicos, el análisis de atributos de calidad y el método de evaluación SAAM (*Software Architecture Analysis Method*). El método se concentra en la identificación de los estilos arquitectónicos o enfoques arquitectónicos utilizados. Estos elementos representan los medios empleados por la arquitectura para alcanzar los atributos de calidad, así como también permiten describir la forma en la que el sistema puede crecer, responder a cambios, e integrarse con otros sistemas, entre otros (22).

Comprende nueve pasos, agrupados en cuatro fases. El Anexo 5 presenta una tabla con las fases y pasos enumerados, junto con su descripción.

3.4 Procedimiento de Evaluación Propuesto

Para la evaluación de la arquitectura se tomaron en cuentas los atributos de calidad relacionados directamente con la arquitectura del software: seguridad, rendimiento, funcionalidad, eficiencia, usabilidad, mantenibilidad, definidos por la ISO/IEC 9126 e incluso se incorporaron otros que también se consideraron importantes desde el punto de vista arquitectónico. Además se decidió de las técnicas estudiadas seguir la línea de la técnica basada en escenarios con el instrumento de evaluación el árbol de utilidad (*Utility Tree*), independientemente de que van implícitos en el método basado en arquitectura ATAM seleccionado para realizar la evaluación, el cual apoya a los involucrados con el proyecto a entender las consecuencias de las decisiones arquitectónicas respecto a los atributos de calidad del sistema.

La estrategia de evaluación implementada evalúa los atributos de calidad no observables vía

ejecución, determinando el impacto de las decisiones arquitectónicas tomadas en dichos atributos.

3.4.1 Evaluando la arquitectura

- **Escenarios**

Como resultado de las actividades de aplicar el método ATAM se obtiene el árbol de utilidades (Ver figura 24), el mismo se toma como un plan para el resto de la evaluación que realiza el método. Está encaminado fundamentalmente a 4 atributos de calidad: desempeño, modificabilidad, escalabilidad y configurabilidad. Como nodos hijos de éstos se encuentran atributos que constituyen una refinación de dichos atributos de calidad.

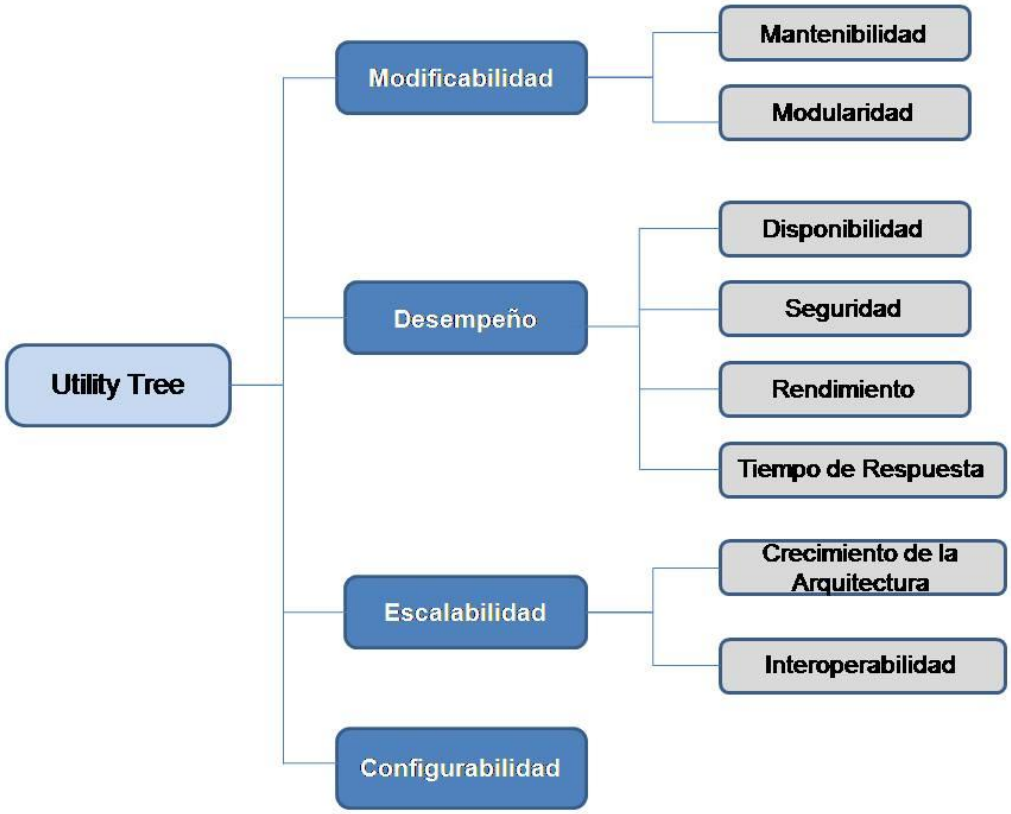


Figura 24. Árbol de Utilidad

A los atributos de calidad se les asignan escenarios que fueron determinados como resultado de una tormenta de ideas entre los involucrados (arquitecto, líder de proyecto, desarrolladores).

Los escenarios deben ser ordenados y priorizados para que los arquitectos puedan contar con más orientaciones para tomar decisiones arquitectónicas. Después se procede a incluir los escenarios más importantes en el árbol de utilidad asignándole un orden de acuerdo a dos criterios.

a) Evaluar el riesgo de no considerar el escenario.

b) Evaluar el esfuerzo necesario para lograr cumplir con el escenario.

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

Tabla 3. Escenarios Categorizados y Priorizados

Atributo de Calidad	Subcaracterísticas	Escenario	Prioridad
Desempeño	**Seguridad**	E1. Un usuario no autorizado intenta acceder a la aplicación.	(M,A)
	Disponibilidad	E2. Se están realizando tareas de mantenimiento o de configuración y el sistema se encuentra fuera de servicio, debe ser capaz de ponerse online y mantener en todo momento una cola de eventos a ejecutar para la generación y envío automático de reportes.	(M,B)
		E3. El usuario está diseñando un reporte y la base de datos fuente no está disponible en ese momento.	(A,M)
Escalabilidad	**Interoperabilidad**	E4. Se requiere que las aplicaciones de las empresas utilicen el generador de reportes para sacar información de los procesos de negocio.	(A,A)
	Crecimiento de la arquitectura	E5. Integración con la suite de herramientas para el Análisis Inteligente de Datos.	(M, M)
		E6. Se desea optimizar el rendimiento de la aplicación distribuyéndola en diferentes servidores cuando se despliegue la misma.	(A, M)
Modificabilidad	**Modularidad**	E7. Se desea cambiar la interfaz de usuario para que ésta sea más amigable para el usuario.	(A, M)

		E8. Incorporar nuevas funcionalidades a los módulos sin detener la explotación del sistema.	
	Mantenibilidad	E9. Se requiere agregar una nueva tabla al modelo de datos del sistema, se genera automáticamente un nuevo modelo para el acceso a datos.	(M, M)
		E10. Se requiere la instalación de una nueva versión de la base de datos en el mínimo tiempo posible.	(M, M)
Configurabilidad		E7. Se desea cambiar la interfaz de usuario para que ésta sea más amigable para el usuario.	(A, M)
Leyenda. Prioridad Alta (A), Media (M), Baja (B).			

Se hace necesaria la explicación detallada de cada escenario para llegar a conclusiones específicas en cuanto a la arquitectura propuesta para poder proceder a la implementación del sistema.

- **Análisis de los escenarios**

Se analizan los escenarios de mayor prioridad arquitectónica, teniendo en cuenta como reacciona la arquitectura del sistema. Se obtiene como resultado un listado con los puntos sensibles, las relaciones entre atributos, los riesgos y no riesgos.

A continuación se muestra el análisis detallado realizado a los principales escenarios involucrados (Ver los restantes escenarios en el documento “Tratamiento de los escenarios”). Se describe para cada uno el estímulo, el ambiente en el que se encuentra el sistema, la respuesta y una explicación de cómo responde la arquitectura a este evento.

El escenario 3 (ver Tabla 4) demuestra la flexibilidad de la arquitectura, ya que al mapear la base de datos de donde se extraerá la información en un XML, centraliza una forma estándar de representar las bases de datos con independencia del gestor que se utilice. Además, es posible el uso de la aplicación Diseñador de modelos en terceros sistemas que requieran de sus servicios. Asociado con este escenario se identificó el riesgo 2 explicado en la próxima sección.

Tabla 4. Escenario # 3 Modelo Semántico

Escenario # 3	El usuario está diseñando un reporte y la base de datos fuente no está disponible en ese momento.			
Atributo(s)	Desempeño-Disponibilidad.			
Ambiente	Operación normal.			
Estímulo	Creación de un reporte sin la base de datos disponible.			
Respuesta	Creación del reporte mediante modelo semántico.			
Decisiones Arquitectónicas	**Punto de Sensibilidad**	**Punto de Intercambio**	**Riesgo**	**No Riesgo**
			R2	
Explicación	El modelo semántico ofrece una abstracción de la base de datos fuente, extrayendo los metadatos necesarios para representar dicha base de datos en un fichero XML, que describe su estructura de una forma flexible y extensible, posibilitando: El trabajo desconectado de la base de datos mejorando el rendimiento y la seguridad. Abstracción de la complejidad de los datos al facilitar la asignación de alias a las entidades de la BD. Flexibilidad al poder ser usado por otras aplicaciones, ya que no tiene dependencia del negocio específico de cada organización.			

El escenario 4 (ver Tabla 5) se refiere a uno de los métodos de integración previstos. El acceso por URL es uno de los métodos más fáciles disponibles para los desarrolladores que necesitan incorporar funcionalidades del SGDR en aplicaciones propias. Está diseñada para brindar un máximo nivel de rendimiento para el acceso remoto a los reportes comunicándose directamente con el Servidor de Reportes.

Tabla 5. Escenario # 4 Integración con Otros Sistemas

Escenario # 4	Se requiere que las aplicaciones de las empresas utilicen el generador de reportes para extraer información de los procesos de negocio.

Atributo(s)	Escalabilidad – Interoperabilidad, Desempeño – Disponibilidad, Configurabilidad.			
Ambiente	Integración con aplicaciones de terceros.			
Estímulo	Utilizar el sistema embebido en otras aplicaciones.			
Respuesta	Se integra el sistema según el nivel de integración escogido.			
Decisiones Arquitectónicas	**Punto de Sensibilidad**	**Punto de Intercambio**	**Riesgo**	**No Riesgo**
		I1		
Explicación	La integración por URL es uno de los métodos de integración. Está diseñada para brindar un máximo nivel de rendimiento para el acceso remoto a los reportes comunicándose directamente con el Servidor de reportes. SGDR chequea si tiene permisos y una cookie de autenticación válida antes de acceder al reporte. El acceso por URL contempla una serie de parámetros que especifican el reporte al cual se desea acceder, el formato, si desea visualizarlo en el Visor de reportes o no, entre otras. La respuesta es codificada en el formato estándar JSON.			

El escenario 9 (Ver Tabla 6) refleja decisiones arquitectónicas que repercuten en el grado de modificabilidad, específicamente mantenibilidad que posee la arquitectura del sistema, relacionado fundamentalmente con el uso de Propel para el acceso a datos. La selección de los estilos arquitectónicos MVC y Capas incide directamente sobre estos atributos. La ventaja principal de este estilo arquitectónico es que en caso de que sobrevenga algún cambio, sólo se ataca al nivel requerido. Además al generar un nuevo modelo de datos con Propel las clases de objetos propias que poseen métodos y propiedades personalizadas no se modifican. Por otro lado las características del estilo MVC, potencia en gran medida la modificabilidad y por tanto el mantenimiento del sistema.

Tabla 6. Escenario # 9 Modelo de Datos

Escenario # 9	Se requiere agregar una nueva tabla al modelo de datos, se genera automáticamente un nuevo modelo para el acceso a datos.
Atributo(s)	Modificabilidad- Mantenibilidad.
Ambiente	Sistema en explotación.

Estímulo	Agregar nueva tabla al modelo de datos.			
Respuesta	Generación automática del modelo y puesta a punto de la aplicación en menos de 1 hora.			
Decisiones Arquitectónicas	**Punto de Sensibilidad**	**Punto de Intercambio**	**Riesgo**	**No Riesgo**
		I4		
Explicación	El framework de acceso a datos Propel posee un grupo de funcionalidades por línea de comandos para la generación automática de la capa acceso a datos, para este escenario los pasos son: Construir esquema de la base de datos. Propel se conecta a la base de datos y crea una descripción de la misma en .yml. Construir modelo de datos: a partir del esquema se generan las clases necesarias que mapean las entidades de la base de datos como objetos. El estilo en capas posibilita realizar modificaciones a la capa de datos y acceso a datos sin afectar las superiores, puesto que los objetos utilizados en capas superiores no se ven afectados. Cambios en el modelo originados por la adición de una nueva tabla, no afectan ni la Vista ni al Controlador, puesto que con el MVC hay un bajo acoplamiento entre ellos. Esto posibilita la modificación de la base de datos del sistema tanto en desarrollo como en producción, sin afectar el funcionamiento de la misma en un mínimo de tiempo.			

Como resultado del análisis de este escenario se detectó un punto de relación entre atributos, que establece que mientras que se favorece la mantenibilidad del sistema en este punto se afecta la disponibilidad, debido a que la aplicación dejará de estar disponible en ese momento.

➢ Resultados Obtenidos

Se analizaron un total de 10 escenarios, identificándose 4 riesgos, 3 puntos sensibles y 4 puntos de intercambio entre atributos de calidad.

- ✓ **Riesgos**
  - Configuración o mantenimiento con el sistema fuera de servicio.
  - La base de datos del cliente no disponible.
  - Cambio de la interfaz de usuario.
  - Instalación de una nueva base de datos.

- ✓ **Puntos Sensibles**
  - Acceso a la aplicación por usuarios no autorizados.
  - Integración con la suite de herramientas para el Análisis Inteligente de Datos.
  - Incorporación de nuevas funcionalidades.
- ✓ **Puntos de Intercambio**
  - Utilización del sistema para extraer información de los procesos de negocio.
  - Distribución del sistema en diferentes servidores durante el despliegue del mismo.
  - Cambio de la interfaz de usuario.
  - Modificaciones y actualizaciones a la base de datos asegura la mantenibilidad a la vez que afecta la disponibilidad.

Una vez identificados todos estos aspectos presentes en la arquitectura resulta necesario efectuar la toma de decisiones por parte de los stakeholders y el equipo que intervino en el proceso de ejecución de las pruebas de concepto de la arquitectura: arquitecto de software, líder de proyecto y algunos desarrolladores.

La toma de decisiones se ejecuta con el objetivo de mitigar los riesgos en la arquitectura desde el punto de vista de los atributos de calidad analizados y dar paso a la construcción del sistema. Como principales decisiones a tomar se aprobaron: definir políticas para la integración con la suite de herramientas para el Análisis Inteligente de Datos, definir una estrategia de manejo y recuperación de errores para facilitar las tareas de mantenimiento y configuración del sistema, creación del modelo semántico para el diseño de reportes cuando la base de datos del cliente esta desconectada mejorando el rendimiento del sistema, así como analizar las consecuencias que pueden arrojar cambios significativos en las decisiones arquitectónicas definidas.

3.5 Conclusiones del Capítulo

En este capítulo se aplicó el método de evaluación de arquitecturas ATAM como parte del proceso de evaluación temprana y la técnica basada en escenarios con el instrumento Árbol de Utilidad, en aras de determinar cómo la propuesta arquitectónica cumple con los atributos de calidad requeridos.

- Se identificaron 4 riesgos, 3 puntos sensibles y 4 puntos de intercambio entre los atributos de calidad presentes en el sistema.
- Se determinó que el sistema es funcional por lo que decidió dar paso a la implementación.

CONCLUSIONES

La definición de una arquitectura sólida y adaptable a las necesidades del sistema constituye uno de los aspectos más importantes para que los desarrolladores tengan una visión y un lenguaje común, posibilitando un trabajo organizado y guiado por buenas prácticas de patrones, metodologías, tendencias y tecnologías actuales.

- ✓ Se realizó un estudio del marco conceptual referencial sobre sistemas generadores de reportes para dar soporte a la propuesta arquitectónica.
- ✓ Quedó definida la plataforma tecnológica para el desarrollo de la nueva versión del SGDR con la selección de herramientas informáticas capaces de agilizar y facilitar la construcción del mismo.
- ✓ Fue descrita la arquitectura de software propuesta para el SGDR en su versión 2.0 destacando los elementos esenciales del Modelo de Casos de Uso, Modelo de Diseño, Modelo de Despliegue, Modelo de Implementación y Modelo de Datos, haciendo uso de estilos y patrones que potencian atributos de calidad.
- ✓ Se aplicó el estilo multicapas que aporta una lógica en cuanto a organización conservando la alta cohesión y el bajo acoplamiento entre los componentes del sistema.
- ✓ La aplicación del patrón Modelo-Vista-Controlador potencia la mantenibilidad del sistema propuesto al desacoplar los componentes.
- ✓ El uso de los frameworks en cada una de las capas fomenta la reutilización, acelera el tiempo de desarrollo y provee una estructura mantenible al código fuente.
- ✓ Los diferentes entornos de despliegue del sistema le aportan flexibilidad y escalabilidad a la arquitectura propuesta, una vez que puede ser adaptado a las necesidades particulares de una empresa u organización que requiera los servicios del generador de reportes.
- ✓ Se demostró que la arquitectura de la aplicación potencia los atributos de calidad requeridos con la aplicación del método ATAM y la técnica basada en simulación incluyendo el artefacto árbol de utilidad.

RECOMENDACIONES

Luego de haber analizado los resultados del presente trabajo de diploma, se puede llegar a la siguiente recomendación:

- La aplicación de otras técnicas de evaluación de la arquitectura en la etapa tardía como Simulación y Prototipos cuando este implementado del sistema.

BIBLIOGRAFÍA

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

Bass, Len, Clements, Paul y Kazman, Rick. 2003. *Software Architecture in Practice. s.l.* 2003. 2da edición.

Bengtsson, P. 2002. Design and Evaluation of Software Architecture. University of Karlskrona/Ronneby, Deparment of Software Engineering and Computer Science, Karlskrona. [En línea] 13 de mayo de 2002. http://www.ipd.hkr.se/pob/archive/thesis.pdf.

Booch, G.,Rumbaugh, J., Jacobson, I. 2000. *El Lenguaje Unificado de Modelado.* Addison-Wesley : s.n., 2000.

BuenMaster.com. 2007. BuenMaster.com. [En línea] 14 de agosto de 2007. http://buenmaster.com/?a=536.

Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., & Stal, M. 1996. *Pattern- Oriented Software Architecture. A Sustem of Patterns. John Wiley & Sons.* Inglaterra. : s.n., 1996. s.n.

Camacho, Erika, Cardeso, Fabio y Nuñez, Gabriel. 2004. *Arquitectura de Software. Guía de estudio.* 2004.

Clements, Paul. 1996. *A Survey of Architecture Description Languages . Proceedings of the International Workshopon Software Specification and Design.* Alemania. : s.n., 1996.

Comunity, OpenUp. Open UP. [En línea] EPF Copyrigth. http://epf.eclipse.org/wikis/opneup.

Danciu, Teodor. 2002. *The Jasper Reports Ultimade Guide.* 2002.

Diseño, Patrones de. 2001. Patrones de Diseño. [En línea] 2001. http://mit.ocw.universia.net/6.170/6.170/f01/pdf/lecture-12.pdf.

Erika Camacho, Fabio Cardeso, Gabriel Nuñez. 2004. *Arquiecturas de Software.* 2004.

ExtJS. 2009. API Documentation. [En línea] 2009. http://extjs.com/deploy/dev/docs.

Ganma, E., y otros. 1995. *Design Patterns: Elements of Reusable Object-Oriented Software.* 1995.

Garlan, David. *Software Architecture, the future of software engineering.* A Roadmap s.l. : En Anthony Finkelstein(ed.). ACM Press, 200.

Gómez, Omar Salvador. 2007. *Evaluando Arquitecturas de Software. Parte 1.Panorama General .* México : Brainwork S.A., 2007. 1870-0888.

Gustavo Andrés Brey, Gastón Escobar, Nicolas Passerini y Juan Arias. 2005. *Arquitectura de Proyectos de IT.Evaluación de Arquitecturas.* Buenos Aires. Universidad Tecnolóica Nacional.Facultad Regional de Buenos Aires-Departamento de Sistemas : s.n., 2005. s.n.

Heffelfinger, David R. Agosto 2009. *Jasper Reports for Java Developers.* Agosto 2009.

Herz, Andreas. 2009. Open-jACOB Draw2D . *Open-jACOB Draw2D.* [En línea] 2009. http://draw2d.org/draw2d.

In, Hoh,Kazman,Rick y Olson,David. 2001. *From Requeriments Negotiation to Software Architectural Decisions. Carnegie.* Software Engineering Institute : s.n., 2001.

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

Jaime Rojas, Nolberto. 2008. *Conviviendo con sistemas legados.* Bogotá, Colombia : Universidad de

los Andes, 2008.

Kicillof, Nicolás y Reynoso, Carlos. marzo de 2004. *Lenguaje de descripción arquitectónica de Software (ADL). s.l.* s.l. : Universidad de Buenos Aires: s.n., marzo de 2004.

Kruchten, P. 1995. Architectural Blueprints - The 4 + 1 vView Model of Software Architecture. [En línea] 1995. http://www.cs.ubc.ca/~gregor/teaching/papers/4+1view-architecture.pdf.

Larman, Craig. 1999. *UML y Patrones.* México : Dawn Speth White, 1999.

Potencier, Fabien y Zaninotto, Francois. 2009. Symfony. La guia definitiva. [En línea] 2009. http://www.symfony-project.org.

Pressman, Roger S. 2002. *Ingeniería de Software. Un enfoque práctico.* Madrid : 5ta edición.s.n., 2002.

Propel. 2009. Propel project. [En línea] 2009. http://propel.phpdb.org/trac.

Reynoso, Carlos and Kicillof, Nicolás. 2004. *Estilos y Patrones en la Estrategia de Arquitectura de Microsoft.* Buenos Aires, Argentina : s.n., 2004. s.n.

Reynoso, Carlos Billy. 2004. Introduccion a la Arquitectura de Software. [En línea] marzo de 2004. http://www.willydev.net/descargas/prev/IntroArq.pdf.

Rick Kazman, Mark Klein y Paul Clements. Agosto 2000. *ATAM: Method for Architecture Evaluation.* Agosto 2000. CMU/SEI-2000-TR-004 ESC-TR-2000-004.

Stig Seather Bakken, Egon Schmid y Jim Winstead. 2001. *Manual de PHP.* 2001.

Wolf, Alexander. enero de 1997. *Succeedings of the Second International Software Architecture Workshop(ISAW-2). s.l. : ACM SIG SOFT Software Engineering Notes, pp,.* enero de 1997.

REFERENCIAS BIBLIOGRÁFICAS

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

2. **Clements, Paul.** *A Survey of Architecture Description Languages .* Alemania : Proceedings of the International Workshopon Software Specification and Design, 1996.

3. **Garlan, David.** *Software Architecture: A Roadmap.* s.l. : En Anthony Finkelstein(ed.), the future of software engineering, ACM Press, 200.

4. **Pressman, Roger S.** *Ingeniería de Software. Un enfoque práctico.* Madrid : s.n., 2002. 5ta edición.

5. BuenMaster.com. *BuenMaster.com.* [Online] agosto 14, 2007. http://buenmaster.com/?a=536.

6. **Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., & Stal, M.** *Pattern- Oriented Software Architecture. A Sustem of Patterns. John Wiley & Sons.* Inglaterra. : s.n., 1996.

7. **Reynoso, Carlos Billy.** Introduccion a la Arquitectura de Software. [Online] marzo 2004. http://www.willydev.net/descargas/prev/IntroArq.pdf.

8. **Larman, Craig.** *UML y Patrones.* México : Dawn Speth White, 1999.

9. **Bengtsson, P.** Design and Evaluation of Software Architecture. University of Karlskrona/Ronneby, Deparment of Software Engineering and Computer Science, Karlskrona. [Online] mayo 13, 2002. http://www.ipd.hkr.se/pob/archive/thesis.pdf.

10. **Wolf, Alexander.** *Succeedings of the Second International Software Architecture Workshop(ISAW-2).* s.l. : ACM SIG SOFT Software Engineering Notes, pp., enero de 1997.

11. **Kicillof, Nicolás y Reynoso, Carlos.** *Lenguaje de descripción arquitectónica de Software (ADL).* s.l. : Universidad de Buenos Aires: s.n., marzo de 2004.

12. **Jaime Rojas, Nolberto.** *Conviviendo con sistemas legados.* Bogotá, Colombia : Universidad de los Andes, 2008.

13. **ExtJS.2009** - API Documentation. [Online] [En línea] 2009. http://extjs.com/deploy/dev/docs.

14. **Herz, Andreas. 2009.** Open-jACOB Draw2D. *Open-jACOB Draw2D.* [En línea] 2009. [Citado el: 2009 de Mayo de 6.] http://draw2d.org/draw2d.

15. **Heffelfinger, David R.** *Jasper Reports for Java Developers.* Agosto 2009.

16. **Danciu, Teodor.** *The Jasper Reports Ultimade Guide.* 2002.

17. **Propel project. 2009.** Propel. [En línea] 2009. [Citado el: 6 de Mayo de 2009.] http://propel.phpdb.org/trac.

18. **Potencier, Fabien y Zaninotto, Francois.** Symfony. La guia definitiva. [En línea] 2009. http://www.symfony-project.org/.

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

20. **Gómez, Omar Salvador.** *Evaluando Arquitecturas de Software. Parte 1.Panorama General.* México : Brainwork S.A., 2007. 1870-0888.

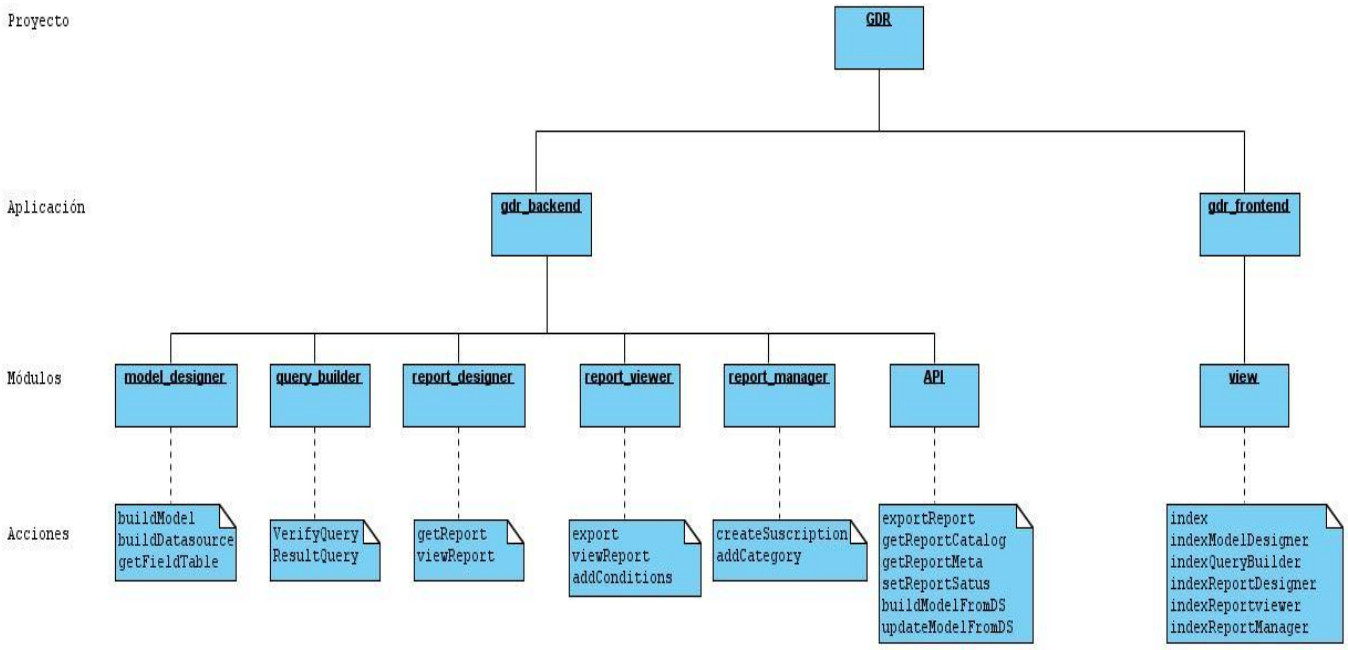
21. **Gustavo Andrés Brey, Gastón Escobar, Nicolas Passerini y Juan Arias.** *Arquitectura de Proyectos de IT.Evaluación de Arquitecturas.* Buenos Aires: Universidad Tecnolóica Nacional.Facultad Regional de Buenos Aires-Departamento de Sistemas : s.n., 2005.

22. **Erika Camacho, Fabio Cardeso, Gabriel Nuñez.** *Arquiecturas de Software.* 2004.

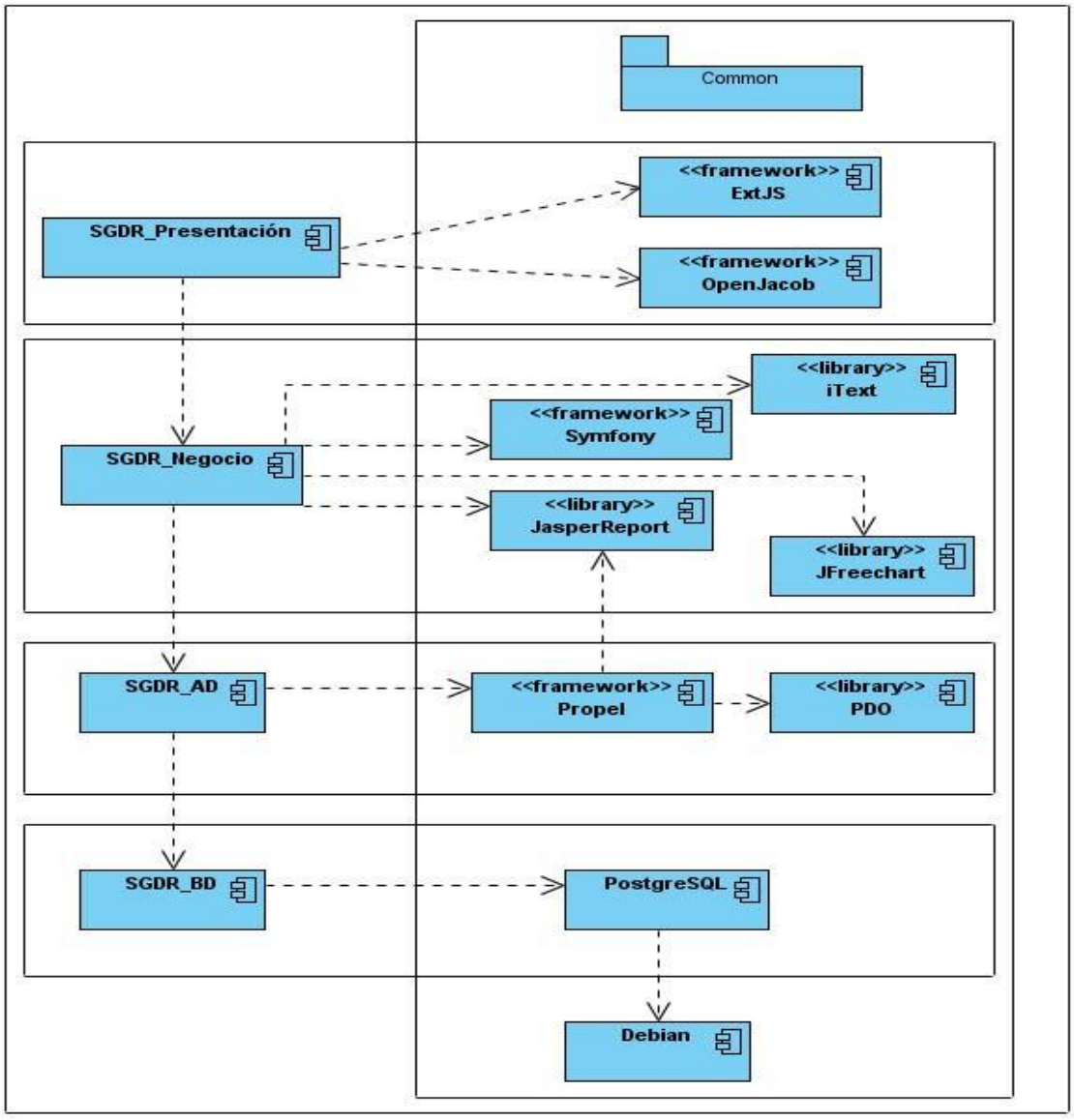
23. **In, Hoh,Kazman,Rick y Olson,David.** *From Requeriments Negotiation to Software Architectural Decisions.* Carnegie : Software Engineering Institute, 2001.

ANEXOS

Anexo1. Estructura del Proyecto Symfony del Sistema



Anexo2. Vista de Implementación del SGDR Distribuido en la Arquitectura Multicapa



Anexo 3. Pasos del Método de Evaluación ATAM

Fase 1: Presentación	
Paso 1. Presentación del ATAM	El equipo de evaluación presenta un panorama general de los pasos de ATAM trata de establecer las expectativas, las técnicas utilizadas y de los resultados del proceso.

Paso 2. Presentación de las metas del negocio	Se realiza la descripción de las metas del negocio que motivan el esfuerzo, y aclara que se persiguen objetivos de tipo arquitectónico. Se presenta brevemente el negocio y el contexto de la arquitectura.
Paso 3. Presentación de la arquitectura	El arquitecto presenta un panorama de la arquitectura, describe la arquitectura, enfocándose en cómo ésta cumple con los objetivos del negocio.
Fase 2: Investigación y análisis	
Paso 4. Identificación de los enfoques arquitectónicos	El equipo de evaluación y el arquitecto deben detallar los planteamientos arquitectónicos descubiertos en el paso anterior. Estos elementos son detectados, pero no analizados.
Paso 5. Generación del Utility Tree	Se identifican, priorizan, definen y refinan los atributos de calidad más importantes en un formato de árbol de utilidad y que engloban la utilidad del sistema, especificados en forma de escenarios. Se anotan los estímulos y respuestas, así como se establece la prioridad entre ellos.
Paso 6. Análisis de los enfoques arquitectónicos	Con base en los resultados del establecimiento de prioridades del paso anterior, se analizan los elementos del paso 4. Se identifican riesgos arquitectónicos, puntos de sensibilidad y puntos de balance. Se utilizan las preguntas similares a las presentadas en el paso 5.
Fase 3: Pruebas	
Paso 7. Lluvia de ideas y establecimiento de prioridad de escenarios.	Con la colaboración de todos los involucrados, se complementa el conjunto de escenarios y se le da prioridad a los escenarios sobre la base de su importancia relativa.
Paso 8. Análisis de los enfoques arquitectónicos	Este paso repite las actividades del paso 6, haciendo uso de los resultados del paso 7. Los escenarios son considerados como casos de prueba para confirmar el análisis realizado hasta el momento.
Fase 4: Reporte	
Paso 9. Presentación de los resultados	Basado en la información recolectada a lo largo de la evaluación del ATAM, se presentan los hallazgos a los participantes. El equipo de evaluación recapitula la ATAM pasos, los resultados, y recomendaciones.