Universidad de las Ciencias Informáticas

Facultad 3

Arquitectura de Software para el Sistema de Gestión de Reportes Dinámicos

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

Autores: Jenny Infante Frometa

Yasmany Hernández Hernández

Tutor: Ing. Yasmany Molina Díaz

Consultante: Lic. Ernesto Sarduy Alonso

Junio, 2009

DECLARACIÓN DE AUTORÍA

Declaramos que somos los únicos autores de este trabajo y autorizamos a la infraestructura Productiva de la Universidad de las Ciencias Informáticas (UCI) a hacer uso del mismo su beneficio y como estime conveniente.

Para que así conste firmamos la presente a los ___ días del mes de _____ de 2009.

_____	_____	_____
Jenny Infante Frometa	Yasmany Hernández Hernández	Ing. Yasmany Molina Díaz
Autor	Autor	Tutor

Dedicatoria

A mami, papi y yeya, por ser la razón de ser de mi vida.

Jenny

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

Yasmany

Agradecimientos

A mami por darme las fuerzas y el amor para seguir adelante; a papi, por ser fuente de mi inspiración y espejo de ejemplo; a mi hermanita del alma por siempre confiar en mí.

Por los 5 largos años de espera, el amor y la confianza que me brindas, gracias a ti, mi siempre nene, Roy.

A toda mi familia y vecinos por el apoyo y la confianza puesta en mí. Gracias a los que aquí me acogieron en su seno y me brindaron su mano cuando más la necesitaba: Andrea, Bienve, Jose, Virgen.

A mi mejor amigo y compañero de tesis, Yasmany, seguiremos siendo los inseparables.

A mi tutor y los compañeros del proyecto, sin ustedes no hubiera sido posible.

A mis amistades del grupo, los que son y los que fueron, por estar siempre ahí y hacer estos años más llevaderos.

Gracias a la universidad por hacer de mi más que una Ingeniera, toda una mujer.

A mi mamá por enseñarme que para conseguir algo basta proponérselo y esforzarse, a mi papá por darme la educación que me convirtió en un hombre de bien.

A Daimí, mi novia, compañera y amiga, por estar siempre a mi lado, apoyarme y preocuparse por mí en todo momento.

A mis abuelos, especialmente a mis abuelas Lucía y María del Carmen por brindarme siempre su cariño. A toda mi familia, porque cada uno de ellos puso un pedacito en mí hasta llegar a ser quien soy.

A mi inseparable amiga de 5 años, Jenny, quién mejor que ella para ser mi compañera de tesis.

Gracias a mi tutor y compañeros del proyecto, sin ustedes no lo hubiésemos logrado.

Al equipo Energía, en especial a Gandol y Alain, con ellos aprendí a aprender.

A todas mis amistades de la UCI, los de hace 5 años y los más nuevos, pero no menos importantes.

Resumen

Los generadores de reportes son herramientas complementarias de los sistemas de información. Proveen una forma transparente al usuario para realizar consultas a la base de datos y obtener información de ella en forma de reporte.

El presente trabajo de diploma describe la arquitectura de software de un sistema de Gestión de Reportes Dinámicos para la web desarrollado en PHP. El conjunto de herramientas implementado constituyen aplicaciones ricas, ya que explotan al máximo la experiencia del usuario haciendo uso de Javascript, AJAX y otras tecnologías subyacentes.

El resultado fundamental se centra en la obtención de una arquitectura de software para este sistema que potencie atributos de calidad, obteniendo como resultado 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 Claves: aplicaciones web, arquitectura de software, generadores de reportes, PHP.

“Programar sin una arquitectura en mente es como explorar una gruta solo con una linterna: no sabes dónde has estado ni hacia dónde vas”

Danny Thorpe

ÍNDICE DE CONTENIDO

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

Introducción

En el entorno competitivo empresarial actual, obtener información valiosa es esencial. Con el advenimiento de nuevas tecnologías, se incrementa la cantidad de datos que es necesario analizar para extraer información útil para los procesos de negocio de las empresas. La tecnología facilita el proceso de recopilación y análisis de los datos manejados, pero lo cierto es que las métricas del funcionamiento y los recursos de información más importantes siguen estando perdidos en un mar de números y de sistemas desconectados.

La mayoría de las organizaciones cuentan con varios sistemas dispersos, en el peor de los casos, cada uno posee sus propias fuentes de datos y mecanismos de representación. Esto provoca que el mantenimiento de la información actualizada a través de los departamentos y unidades de negocios sea extremadamente difícil; es por ello que se puede afirmar que las organizaciones tienen una abundancia de datos, pero una penuria de conocimiento.

Los procesos de descripción, análisis y representación de la información, así como las nuevas tecnologías asociadas a ellos, adquieren, en estas circunstancias, un sentido trascendente: más que simples medios para la obtención de resultados debe considerárseles como herramientas que contribuyen al desempeño, aprendizaje individual y colectivo, así como a la construcción positiva de la empresa, en función de obtener utilidades y crear los valores propios de la organización.

El principal reto de muchas empresas es proporcionar la información adecuada a las personas adecuadas en el momento adecuado. Los responsables de utilizar la información frecuentemente necesitan acceder, procesar y visualizar datos de negocio que pueden estar distribuidos por toda la organización y/o fuera de ella. Con este fin, existen en el mundo diferentes herramientas que viabilizan el proceso de generación de reportes, se pudieran citar algunas como: Active Reports, Jasper Reports y Crystal Reports, pero estas solo cubren parte del ciclo de vida de los reportes, sin embargo existen otras como Microsoft SQL Server 2005 Reporting Services que además de facilitar el diseño y generación de reportes, soporta los procesos de administración y entrega de los mismos, pero su costo es muy elevado y es una herramienta privativa.

El Ministerio de Energía y Petróleo de la República Bolivariana de Venezuela, siguiendo la línea trazada por la Revolución Bolivariana de producir un cambio en la gestión de los ministerios, como parte del proceso de informatización de la sociedad venezolana, ha realizado un grupo de transformaciones y

mejoras en las diferentes oficinas adscritas al mismo en busca de garantizar la estabilidad, confiabilidad, vitalidad, seguridad e inviolabilidad de las tecnologías minimizando dependencias tecnológicas.

Dentro del marco de cooperación del proyecto de integración latinoamericana ALBA, la empresa cubana ALBET.SA, que constituye la representación comercial de la Universidad de las Ciencias Informáticas, ha sido contratada para la automatización de las áreas y servicios del Ministerio.

En consecuencia, se desarrolla en el Centro de Tecnologías de Almacenamiento y Análisis de Datos de la Universidad de las Ciencias Informáticas un sistema de Gestión de Reportes Dinámicos, cuya finalidad se centra en obtener un grupo de aplicaciones capaces de crear, administrar y entregar reportes dinámicos en tiempo real o programado. Debe ser capaz de satisfacer los requisitos solicitados por la Oficina de Tecnología e Informática del Ministerio de Energía y Petróleo de la República Bolivariana de Venezuela, y además, ser lo suficientemente genérico como para ser utilizado en cualquier otra institución que requiera los servicios de un sistema de Gestión de Reportes Dinámicos, 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.

En 1972, David Parnas, precursor importante de la Arquitectura de Software e introductor de algunas de sus nociones más esenciales y permanentes, publicó un ensayo en el que discutía la forma en que la modularidad en el diseño de sistemas podía mejorar la flexibilidad y el control conceptual del mismo, acortando los tiempos de desarrollo. Decía Parnas que las decisiones tempranas de desarrollo serían las que probablemente permanecerían invariantes en el desarrollo ulterior de una solución. Esas “decisiones tempranas” constituyen de hecho lo que hoy se llamarían decisiones arquitectónicas. La estructura es primordial, y la elección de la estructura correcta ha de ser crítica para el éxito del desarrollo de una solución (Clements, y otros, 1996), pues ello sintetiza, como ninguna otra expresión, el programa y la razón de ser de la Arquitectura de Software.

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, ejemplo de esto es el conjunto de herramientas que se encuentran en explotación en el Ministerio para la generación de reportes.

La Oficina de Tecnología e Informática es la encargada de proveer las aplicaciones para la generación de reportes que son solicitados por el personal del Ministerio o cualquier ente adscrito. Para satisfacer dichas necesidades cuenta con reportes predeterminados. Al igual que en muchas otras entidades y organizaciones, al salir los sistemas a producción, aparecen nuevas necesidades de información, que ameritan incorporar nuevos tipos de reportes a las aplicaciones. Esta situación provoca una fuerte dependencia entre los usuarios y la entidad, pues se precisa modificar el código fuente de las aplicaciones con cada nueva solicitud, lo cual atrasa y limita la inclusión de los nuevos reportes en los sistemas. Para dar solución a dicha problemática se hace necesario desarrollar una herramienta que con una arquitectura robusta, flexible y escalable, sea capaz de gestionar eficientemente el proceso de diseño, generación y administración de reportes de forma dinámica y personalizada.

En atención a lo expuesto, se determinó el siguiente **Problema Científico:**

¿Cómo diseñar una arquitectura de software robusta, flexible y escalable para un Sistema de Gestión de Reportes que permita la generación y administración de informes de forma dinámica?

Objeto de Estudio

Proceso de desarrollo de software del sistema de Gestión de Reportes Dinámicos.

Objetivo

Obtener la arquitectura de software para el sistema de Gestión de Reportes Dinámicos que permita la generación y administración de informes de forma dinámica garantizando que esta sea robusta, flexible y escalable.

Campo de Acción

Arquitectura de software del sistema de Gestión de Reportes Dinámicos.

Hipótesis

Si se aplican correctamente los estilos y patrones arquitectónicos a los elementos estructurales del sistema de Gestión de Reportes Dinámicos entonces se obtendrá una arquitectura robusta, flexible y escalable que soporte la implementación de un conjunto de aplicaciones que cumplan con las funcionalidades requeridas.

Tareas de la Investigación

1. Sistematización del estado del arte sobre arquitecturas para sistemas generadores de reportes.
2. Selección de las herramientas y metodología a usar en el desarrollo del sistema.
3. Identificación y fundamentación de los estilos y patrones arquitectónicos a utilizar en la solución.
4. Describir la evolución de la arquitectura base para obtener un producto de calidad.
5. Evaluación de la arquitectura propuesta.

Para dar cumplimiento a las tareas se emplearon los siguientes métodos científicos en el proceso investigativo:

Métodos teóricos:

Analítico - Síntesis: Se utilizó para el procesamiento de la información permitiendo analizar los documentos y la extracción de los elementos más importantes acerca del proceso de gestión de reportes y para precisar características del modelo arquitectónico propuesto, así como para el arribo de las conclusiones de la investigación.

Histórico - Lógico: Para determinar las tendencias actuales de desarrollo de los modelos y enfoques arquitectónicos de los sistemas de gestión de reportes.

Inductivo - Deductivo: A partir del estudio de distintos estilos y modelos de arquitecturas arribar a proposiciones de estilos de arquitecturas específicas.

Método Sistémico: Para determinar los componentes arquitectónicos y definir las relaciones entre estos, para favorecer el logro de una arquitectura flexible, reusable y robusta.

Hipotético - Deductivo: Se utilizó para a partir del problema concreto y objetivo plantear la hipótesis que será puesta a contrastación en todo el proceso investigativo.

Empíricos

Observación: Se utilizó para realizar una valoración a partir de la percepción en la etapa exploratoria de la realidad estudiada para determinar cómo ocurre realmente el proceso de gestión de reportes en el Ministerio y cuáles son los principales elementos en los que se debe trabajar.

El modo eficiente de desarrollar el proceso de gestión de informes es un tema que cobra vital importancia en la actualidad, el diseño de sistemas de reportes que estén conformados por un conjunto de herramientas con una arquitectura flexible, modular y robusta en el contexto actual da al traste con el éxito de la organización donde se aplique, una vez que perfecciona los procesos de presentación y uso de la información que en esta se maneja.

El sistema de Gestión de Reportes Dinámicos ofrece una solución de reporte que potencia una redituable inteligencia de negocios, destinada a mejorar la rapidez y calidad de la adopción de decisiones en todos los niveles de una organización.

Las instituciones requieren una manera centralizada de crear, administrar y entregar reportes en tiempo real. El sistema permite a las organizaciones encargarse de este proceso y les proporciona a los desarrolladores herramientas para confeccionar reportes e implementar soluciones personalizadas de informes para usuarios individuales en toda una organización. Constituye una de las herramientas que posibilitan la inteligencia de negocios ya que genera vistas agregadas de datos para mantener a la gerencia informada sobre el estado de su negocio, además de que:

- Proporciona soporte a todo el ciclo de vida de los reportes.
- Provee accesibilidad a la información emitiendo reportes a través de un navegador Web u otros formatos estándares.
- Está orientada al usuario final, brindando herramientas como el Diseñador de Reportes que permite la confección de informes de forma interactiva en la web.

Es una solución abierta y extensible para los informes administrados. Su arquitectura flexible permite a los programadores de software y a las empresas integrar el conjunto de herramientas con sus sistemas heredados, portales empresariales o aplicaciones personalizadas.

El sistema de Gestión de Reportes Dinámicos constituye una aplicación RIA, acrónimo de Rich Internet Applications (Aplicaciones Ricas de Internet) puesto que combina las ventajas que ofrecen las aplicaciones Web y las aplicaciones tradicionales o de escritorio. Se evidencia en las características y potencialidades del Diseñador de reportes, que se extiende al máximo las posibilidades de diseño del usuario. Este diseñador permite personalizar al detalle la salida de los reportes y la forma en que la información será visualizada. Por otra parte, el servicio de entrega automatizada de informes es un elemento de valor agregado para los sistemas generadores de reportes, este servicio permitirá a los

usuarios hacer suscripciones a determinados reportes que posteriormente recibirán de forma automática y regular por la vía especificada y en el formato deseado.

Su arquitectura modular y las posibilidades de escalabilidad por medio de extensiones y complementos que se integrarán a los procesadores principales que componen el servidor de reportes, permite que ésta se integre a herramientas que soporten el proceso de ayuda de toma de decisiones en las empresas u organizaciones.

Estructura de la Tesis

La tesis quedó estructurada en tres capítulos.

Capítulo 1 Fundamentación teórica: Estudio del arte de la arquitectura de software y las herramientas de reportes que existen en el mundo. Se realiza un análisis crítico y valorativo del estado del arte en el tema.

Capítulo 2 Descripción arquitectónica: En este capítulo se hace una propuesta de solución al problema planteado en el diseño teórico de la investigación. Se realiza la descripción de la arquitectura del sistema utilizando las 4+1 vistas de la arquitectura propuestas por la metodologia utilizada más la vista de datos.

Capítulo 3 Evaluación de la arquitectura: Se procede a la evaluación de la solución propuesta utilizando los métodos basados en escenarios y simulación.

1 Fundamentación Teórica

En este capítulo se abordarán temas relacionados con el estado del arte de las diferentes herramientas que existen en el mundo para la generación de reportes, sus características y un análisis crítico de las mismas a través de una comparación teniendo en cuenta los aspectos más relevantes. Se analizará la Arquitectura de Software como disciplina dentro del proceso de desarrollo de software, sus tendencias actuales, así como una fundamentación de los estilos y patrones arquitectónicos utilizados en la solución. Se establecen las herramientas, metodología y lenguajes a utilizar, argumentando el por qué de su elección.

1.1 Estado del arte Generadores de Reportes

Actualmente, en diversos sistemas de generación de reportes a nivel mundial existen problemas con el dinamismo a la hora de brindar la información necesaria. Las herramientas que permiten este trabajo son elaboradas para generar un tipo de reporte determinado en tiempo de desarrollo lo cual obliga al usuario a redefinir los requisitos cada vez que necesite añadir uno nuevo. 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 con este fin existen en el mundo y en Cuba, particularmente en la Universidad de las Ciencias Informáticas (UCI).

1.1.1 Microsoft SQL Server 2005 Reporting Services

SQL Server 2005 Reporting Services 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.

Reporting Services contiene los componentes principales siguientes:

- Servidor de informes que aloja y procesa informes en diversos formatos: HTML, PDF, TIFF, Excel, CSV, etc.
- API que permite a los programadores integrar o extender procesamiento de datos e informes en aplicaciones personalizadas.
- Los informes incluyen características interactivas basadas en la Web: informes de varios niveles de detalle, que permiten la navegación por distintas capas de datos; los informes con parámetros, que admiten el filtro de contenido en tiempo de ejecución; o los informes de formato libre, con diseños verticales, anidados o adyacentes.
- 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, 2006).

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

Licencia bajo la que se libera: 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, 2006).

1.1.2 Active Reports

ActiveReports para .Net 3.0 toma las mundialmente premiadas capacidades de ActiveReports llevándolas al más alto nivel con mejoras significativas que proveen la habilidad de diseñar, crear y desplegar aplicaciones de reportes de forma fácil y rápida.

Características principales:

- Está diseñado teniendo en cuenta a los desarrolladores y sus diversas necesidades. Soporta el uso de componentes .NET en tiempo de diseño
- Escrito completamente en Visual C# y provee una completa integración con Visual Studio .NET.
- Asistente para la conversión de reportes importados desde Microsoft Access.
- Incluye filtros para exportar a los más populares formatos tales como Adobe PDF, Microsoft Excel, RTF, HTML, texto plano e imágenes TIFF, tanto para aplicaciones de escritorio como para las basadas en la web.
- Diseñador de reportes orientado a usuarios finales, permitiendo su inclusión en las aplicaciones con el fin de que los propios usuarios diseñen y modifiquen sus reportes (GrapeCity, inc.).

Sistema Operativo: En tiempo de diseño para utilizarlo se requiere tener instalado Microsoft Visual Studio .NET 2003, Visual Studio 2005, o Visual Studio 2008 sobre Windows NT 4.0, Windows 2000, Windows XP, Windows Vista o Windows 2003 Server así como Microsoft .NET Framework v1.1, v2.0, v3.0, o v3.5.

En tiempo de ejecución: Windows 98, Windows NT 4.0, Windows 2000, Windows XP, Windows Vista o Windows 2003 Server con Microsoft .NET Framework v1.1, v2.0, v3.0, or v3.5.

Licencia: Se distribuye bajo licencia privativa perteneciente a GrapeCity, inc. Es aplicable para cada desarrollador individual y puede llegar a costar hasta $ 9,799 dólares en su versión profesional. La versión estándar alcanza un precio de $ 4199 dólares.

1.1.3 Jasper Reports

Es el motor de reportes más usado actualmente en el mundo del Open Source (Código Abierto), puede ser embebido en todo tipo de aplicaciones, desde las que generan reportes a partir de una plantilla o modelo predeterminado hasta las que brindan más libertad al usuario para diseñar sus propios reportes y ejecutar otras operaciones complejas. Es una librería implementada completamente en Java brindando el máximo nivel de portabilidad y con un amplio y expandible grupo de posibles fuentes de datos, posee además una amplia variedad de formatos de salida y exportación así como una gran comunidad mundial que mantiene y desarrolla la librería.

Genera informes para formatos de impresión predeterminados existentes o para reportes continuos a ser visualizados en la web, los reportes pueden ser exportados a formatos como: PDF, XML, HTML, CSV, XLS, RTF, TXT.

A través de los subreportes es posible manipular y diseñar reportes con un diseño altamente complejo. Soporta a la misma vez varios orígenes de datos incluso aunque sean de tipos distintos.

Es posible pasar parámetros desde una aplicación a JasperReports, esto es muy simple de implementar y brinda una herramienta muy poderosa ya que permite clasificar, restringir o mejorar los datos que son enviados al usuario basándose en las condiciones de tiempo de ejecución (Jasperforge.org, 2008).

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 en caso que se utilicen fuentes de datos relacionales.

Licencia: Se distribuye a nivel mundial bajo los términos de la Licencia Pública para Librerías GNU (GNU Library Public License), por lo que es software libre y está respaldado por una gran comunidad internacional de desarrollo, el proyecto JasperForge.org y la empresa JasperSoft Corporation.

1.1.4 Crystal Reports

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

Las versiones actuales de Crystal Reports dan la posibilidad de:

- Crear y diseñar fácilmente reportes interactivos y conectarlos con prácticamente cualquier fuente de datos.
- Alto grado de precisión en su diseño y gráficos dinámicos que brindan una gran cantidad de información útil.
- Los informes pueden ser entregados vía web, por e-mail, en cualquier formato de Microsoft Office, Adobe PDF o embebidos en aplicaciones mayores de gestión.
- Las tres funciones que constituyen la base de la arquitectura de Crystal Reports son: Crystal Reports Designer incrustado, para el diseño de informes; Controles del visor de informes y Modelos de objetos (Microsoft Corporation).

Una de las desventajas es que sólo utiliza una jerarquía para agrupar los datos, o sea, clasifica los datos según la jerarquía otorgada y así define la estructura de todo el informe. Como consecuencia, si se desea generar reportes con estructura compleja, el usuario está obligado a crear varios subreportes. Además, elabora los reportes en tiempo de diseño ya que está dirigida más bien al programador y no al usuario.

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*), por lo que es software privativo y de uso restringido mediante el pago de patente (CrystalReports.com, 2009).

1.1.5 Generador de reportes Akademos

La aplicación Akademos, desarrollada en la UCI en el año 2006, es un sistema automatizado para la gestión académica que cuenta con un módulo para la generación de reportes. Utiliza Active Reports para la confección de los mismos pero se ve limitado ya que está adaptado solamente al negocio de la gestión académica en la UCI, restringiendo que los reportes sean elaborados con la información contenida en la base de datos específicamente utilizada por el software, lo cual imposibilita que esta herramienta sea de propósito general.

A continuación se ofrece una tabla comparativa resumiendo los aspectos fundamentales de estos sistemas.

Sistema	Arquitectura	Licencia	Libre	Sistema Operativo
SQL Server Reporting Services	Servidor	SQLS 2005	No	Windows 2003 Server/XP SP2
Active Reports	Librería	Individual	No	Windows 2000/2003/XP/Vista
JasperReports	Librería	GNU/GPL	Si	Multiplataforma
Crystal Reports	Librería	EULA	No	Windows 2003 Server/XP SP2

Tabla 1 Comparación de las herramientas de generación de reporte.

Variadas son las ofertas de herramientas con este fin que existen en el mercado. 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.

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 contrasta, precisamente, con los principales intereses del Gobierno Bolivariano de soberanía tecnológica, y 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.

Con respecto a JasperReports, a pesar de ser software libre, sus funcionalidades se ven limitadas desde el punto de vista que:

- El diseñador de reportes es una aplicación de escritorio, lo que implica la instalación de la misma en cada máquina cliente y además entra en contraste con uno de los principales requisitos solicitados por los clientes.
- El rendimiento se ve afectado, pues al ejecutarse sobre la Maquina Virtual de Java las aplicaciones suelen ser más lentas.

Es por ello que se hace necesario analizar y tomar lo mejor de cada una de estas herramientas con el objetivo de desarrollar un sistema de Gestión de Reportes Dinámicos (SGRD), que con un conjunto de herramientas web permita cubrir las necesidades de reportes de cualquier empresa o institución que lo requiera, siendo imprescindible el diseño de una arquitectura que posibilite y soporte la implementación del mismo en consecuencia a las solicitudes de los clientes.

1.2 La Arquitectura de Software

La arquitectura de software remonta sus antecedentes a la década de 1960 gracias a las tempranas inspiraciones del legendario Edsger Dijkstra, en 1968. Dijkstra, de la Universidad Tecnológica de Eindhoven en Holanda y Premio Turing 1972, propuso que se estableciera una estructuración correcta de los sistemas de software antes de lanzarse a programar, escribiendo código de cualquier manera (Reynoso, Marzo, 2004). Aunque no emplea el término arquitectura para describir el diseño conceptual del software, sus conceptos sientan las bases para lo que luego expresarían Niklaus Wirth (Wirth, Abril de 1971), DeRemer y Kron (DeRemer, et al., 1976).

Pensar la descomposición del programa antes de programar se comienza a proponer, investigar y aplicar a principios de la década del 70 cuando se comienza a distinguir entre pequeños y grandes programas o sistemas (DeRemer, et al., 1976). En la misma época, otro precursor importante, David Parnas, demostró que los criterios seleccionados en la descomposición de un sistema impactan en la estructura de los programas y propuso diversos principios de diseño que debían seguirse a fin de obtener una estructura adecuada (Parnas, 1972), enfatizando siempre en la búsqueda de calidad del software, medible en términos de economías en los procesos de desarrollo y mantenimiento (Designing software for ease of extension and contraction, Marzo 1979). El pensamiento de Parnas sobre familias de programas, en particular, anticipa ideas que luego habrían de desarrollarse a propósito de los estilos de arquitectura.

En el año 1992 aparece el primer estudio donde se define el término “arquitectura de software” en el sentido en que hoy se conoce; sus autores Perry y Wolf (Wolf, et al., Octubre de 1992) definen tres componentes: elementos, forma y razón, cada una de estas ideas ha permanecido viva desde entonces. Esta década de 1990 sintetizó la consolidación de la arquitectura de software protagonizada por la Universidad Carnegie Mellon (CMU SEI), surge también la programación basada en componentes y el surgimiento de los patrones.

Uno de los acontecimientos arquitectónicos más importantes del año 2000 fue la hoy célebre tesis de Roy Fielding que presentó el modelo REST, el cual establece definitivamente el tema de las tecnologías de Internet y los modelos orientados a servicios y recursos en el centro de las preocupaciones de la disciplina (Fielding, 2000).

1.2.1 Definición

En la actualidad, el número de definiciones de arquitectura de software existentes alcanza un orden de tres dígitos, amenazando llegar a cuatro. Lo cierto es que ninguna es respaldada totalmente por los arquitectos de hoy, lo que se resume en un gran número de alternativas o contrapuestas, ejemplo de ello es la colección que se encuentra en el Software Engineering Institute (SEI) donde se entremezcla el trabajo dinámico de estipulación de la arquitectura dentro del proceso de ingeniería o el diseño (su lugar en el ciclo de vida), la configuración o topología estática de sistemas de software contemplada desde un elevado nivel de abstracción y la caracterización de la disciplina que se ocupa de uno de esos dos asuntos, o de ambos.

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 Rational Unified Process, 1999 (Jacobson, et al., 2000):

“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]

Se define la arquitectura como parte integrante e indispensable de la Ingeniería de Software, con un enfoque orientado a objetos. Abarca conceptos importantes como la organización, composición y colaboración de elementos de un sistema de software.

Por otro lado, importantes investigadores en el tema plantean: (Clements, 1996):

"La arquitectura de software de un programa o sistema informático es la estructura o estructuras del sistema, que incluyen elementos de software, las propiedades externas visibles de esos elementos, y las relaciones entre ellos."[2]

En esta definición se plantea que la arquitectura es una abstracción del sistema como un todo, ocultando comportamiento específico de los elementos que va más allá de las relaciones entre ellos y que no impiden el uso y colaboración entre sí. Implica tres elementos importantes al decir de los autores, primero que la arquitectura define los elementos de software; en segundo lugar, que comprende más de una estructura de los sistemas; cada sistema informático posee una arquitectura de software; por último, el comportamiento de cada elemento forma parte de la misma.

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

"La Arquitectura de Software es la organización fundamental de un sistema encarnada en sus componentes, las relaciones entre ellos y el ambiente y los principios que orientan su diseño y evolución."

En esta definición el término componente se refiere a elemento, término más general para referirse a los elementos estructurales. Esta definición dada por la IEEE, define la arquitectura no como un flujo de trabajo dentro de una metodología, establece que no se inscribe en ninguna metodología de desarrollo, sino que es en sí, una disciplina que se desarrolla durante todo el ciclo de desarrollo de software y constituye la organización del sistema al nivel más alto de abstracción.

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

[2] En la primera edición del libro a los elementos básicos estructurales se le denominaba "componente", en esta edición se utiliza el término "elemento" para referirse a aspectos más generales.

Independientemente de las discrepancias entre las diversas definiciones, es común entre todos los autores el concepto de la arquitectura como un punto de vista que concierne a un alto nivel de abstracción. Existe en general, acuerdo de que ella se refiere a la estructura a grandes rasgos del sistema, estructura consistente en elementos y relaciones entre ellos. La noción clave de la arquitectura es la organización (un concepto cualitativo o estructural). (Reynoso, Marzo, 2004).

De forma general se puede decir que la arquitectura de software abarca decisiones importantes sobre:

- La organización del sistema software.
- Los elementos estructurales que compondrán el sistema, sus interfaces, comportamiento y colaboración.
- Evolución de los elementos estructurales en subsistemas.
- El estilo de arquitectura.

Más allá de que hoy existan numerosos conceptos en el plano detallado de las técnicas y metodologías, esta disciplina se articula alrededor de algunos conceptos y principios esenciales los cuales se tratarán en los epígrafes siguientes.

1.3 Estilos Arquitectónicos

Los estilos arquitectónicos, caracterizan una familia de sistemas que están relacionados por compartir propiedades estructurales y funcionales. Generalmente proveen guía y análisis para crear una clase amplia de arquitecturas en un dominio específico donde los patrones se enfocan en solucionar los problemas más pequeños, más específicos dentro de un estilo dado. Los estilos, en cambio, expresan la arquitectura en el sentido más formal y teórico, constituyendo un tópico esencial, ellos se encuentran en el centro de la arquitectura y constituyen buena parte de su sustancia (Kicillof, et al., Marzo de 2004).

En octubre de 1992 se tiene referencia de la primera definición explícita de estilos, aparece propuesta por Dewayne Perry de AT&T Bell Laboratories de New Jersey y Alexander Wolf de la Universidad de Colorado; definen un estilo arquitectónico como una abstracción de tipos de elementos y aspectos formales a partir de diversas arquitecturas específicas. Un estilo arquitectónico encapsula decisiones esenciales sobre los elementos arquitectónicos y enfatiza restricciones importantes de los elementos y sus relaciones posibles (Wolf, et al., Octubre de 1992). En el texto fundacional de la arquitectura de software, Perry y Wolf establecen el razonamiento sobre estilos de arquitectura como uno de los aspectos

fundamentales de la disciplina. Un estilo es un concepto descriptivo que define una forma de articulación u organización arquitectónica. El conjunto de los estilos cataloga las formas básicas posibles de estructuras de software, mientras que las formas complejas se articulan mediante composición de los estilos fundamentales (Reynoso, Marzo, 2004).

Al hacer referencia a restricciones proporciona una visibilidad a ciertos aspectos de la arquitectura, evitando sean violados o ignorados. Un segundo énfasis, más circunstancial, concierne a la susceptibilidad de los estilos arquitectónicos a ser reutilizados. Llama la atención el hecho de que en estos inicios de los estilos como concepto no aparece todavía ningún atisbo de tipología estilística.

Por otro lado, Mark Klein y Rick Kazman en 1999 proponen una definición donde afirman que son artefactos de ingeniería importantes porque definen clases de diseño junto con las propiedades conocidas asociadas a ellos. Ofrecen evidencia basada en la experiencia sobre la forma en que se ha utilizado históricamente cada clase, junto con razonamiento cualitativo para explicar por qué cada clase tiene esas propiedades específicas (Mark Klein, Octubre de 1999).

Empleando retóricamente más palabras que las necesarias, en el siglo XXI, Roy Thomas Fielding sintetiza la definición diciendo que un estilo arquitectónico es un conjunto coordinado de restricciones arquitectónicas que restringe los roles/rasgos de los elementos arquitectónicos y las relaciones permitidas entre esos elementos dentro de la arquitectura que se conforma a ese estilo (Fielding, 2000).

Los estilos se ordenan en seis o siete clases fundamentales y unos veinte ejemplares, como máximo. A continuación la clasificación propuesta por Shaw y Garlan (Mary Shaw, 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)

Los estilos casi siempre se usan combinados; cada capa o componente puede ser internamente de un estilo diferente al de la totalidad; muchos estilos se encuentran ligados a dominios específicos, o a líneas de producto particulares.

A continuación se describen algunos de los estilos que se encuentran bajo la clasificación de Llamada y Retorno por su importancia en la elaboración de la solución.

1.3.1 Arquitectura en Capas

Ayuda a estructurar aplicaciones que pueden estar descompuestas en grupos de subtareas en las cuales cada grupo está en un nivel particular de abstracción. Este patrón describe el principio más difundido en la arquitectura. La calidad tan especial de la arquitectura de tres capas consiste en aislar la lógica de la aplicación y en convertirla en una capa intermedia bien definida y lógica del software.
En la capa de presentación se realiza relativamente poco procesamiento; las ventanas envían a la capa intermedia peticiones de trabajo y este se comunica con la capa de almacenamiento del extremo posterior.

Ventajas

- Modularidad del sistema.
- Facilita la localización de errores.
- Mejora soporte del sistema.
- Reutilización de capas.
- Contención de cambios a una o pocas capas.

Desventajas

- Puede ser difícil definir que componentes ubicar en cada una de las capas.
- A veces no se logra la contención del cambio y se requiere una cascada de cambios en varias capas.
- Pérdida de eficiencia.

- Dificultad de diseñar correctamente la granularidad de las capas.

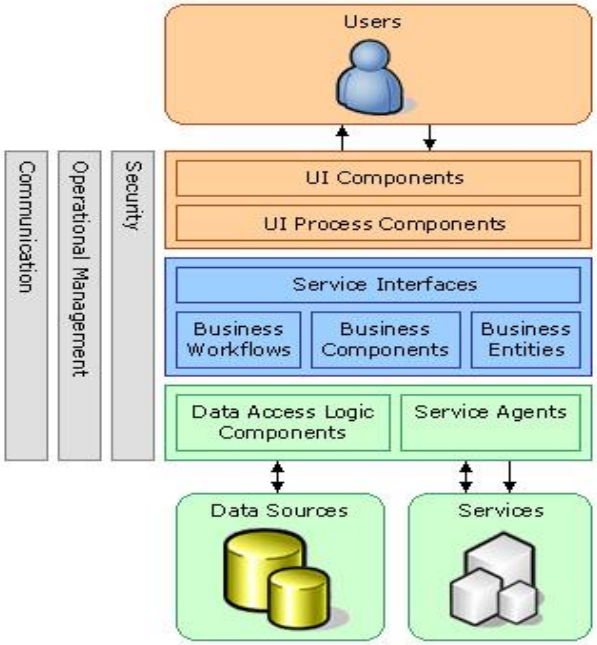


Figura 1 Arquitectura en capas.

1.3.2 Modelo-Vista-Controlador (MVC)

Separa los datos de una aplicación, la interfaz de usuario, y la lógica de control en tres componentes distintos. MVC se ve frecuentemente en aplicaciones web, donde la vista generalmente es la página HTML y el código que provee de datos dinámicos a la página; el modelo es el Sistema de Gestión de Base de Datos (SGBD) y la Lógica de negocio; y el controlador es el responsable de comunicar el modelo y la vista, respondiendo a peticiones en ambos sentidos.

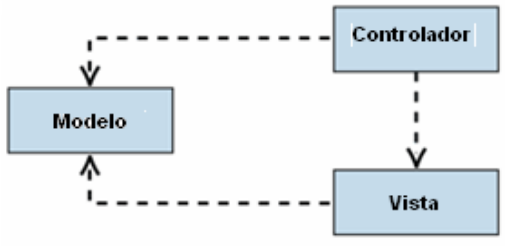


Figura 2 Patrón Modelo-Vista-Controlador

Modelo: Esta 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. Responde a requerimientos de información sobre su estado (usualmente formulados desde la vista y responde a instrucciones de cambiar el estado habitualmente desde el controlador). Es independiente de la vista y el controlador.

Vista: Presenta el modelo en un formato adecuado para interactuar, usualmente la interfaz de usuario.

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

Ventajas

- Separación entre interfaz, lógica de negocio y de presentación.
- Evita poner el código indebido en la capa impropia. Facilita despliegue en caso de modificaciones en el modelo de datos.
- Sencillez para crear distintas representaciones de los mismos datos.
- Reutilización de los componentes.
- Simplicidad en el mantenimiento de los sistemas.
- Los desarrollos suelen ser más escalables.

Desventajas

- Tener que ceñirse a una estructura predefinida puede aumentar la complejidad de la solución, hay problemas que son más difíciles de resolver respetando el patrón.
- Si hay demasiados cambios en el modelo puede provocar un constante refrescamiento de las vistas, a menos que se prevea programáticamente.
- La curva de aprendizaje para los nuevos desarrolladores se estima mayor que la de modelos más simples.
- La distribución de componentes obliga a crear y mantener un mayor número de ficheros.

1.4 Patrones

Los desarrolladores con experiencia acumulan un repertorio tanto de principios generales como de soluciones basadas en aplicar ciertos estilos que les guían en la creación de software. Estos principios y estilos, si se codifican con un formato estructurado que describa el problema y la solución, y se les da un nombre, podrían llamarse “Patrones”.

Buschmann define un patrón de software como aquel que describe un problema recurrente que ocurre en un contexto específico en el diseño de sistemas y presenta un esquema de solución genérica y probada. La especificación de la solución incluye la descripción de los componentes, sus responsabilidades y relaciones y la forma en que colaboran entre sí (Buschmann, 1996). Por tanto, un patrón no es más que la descripción detallada de una solución adecuada a un problema concreto. El establecimiento de estos patrones comunes es lo que posibilita el aprovechamiento de la experiencia acumulada en el diseño de aplicaciones.

La estructura de un patrón es la siguiente:

1. Nombre: Define un vocabulario de diseño. Facilita la abstracción.
2. Problema: Describe cuando aplicar el patrón. Conjunto de fuerzas: objetivas y restricciones. Prerrequisitos.
3. Solución: Elementos que constituyen el diseño (plantilla). Forma canónica para resolver fuerzas.
4. Consecuencias: Resultados. Extensiones. Consensos.

1.4.1 Categorías de patrones

Cada una de las categorías de patrones define un mismo nivel de abstracción o escala de aplicabilidad (Buschmann, 1996).

Patrones de arquitectura: Constituyen plantillas para arquitecturas de software concretas. 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 construir sistemas 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.

Idiomas: Patrones de bajo nivel específicos para un lenguaje de programación o entorno concreto. Describe cómo implementar aspectos particulares de los componentes y sus relaciones usando características propias del lenguaje.

1.4.1.1 Patrones de arquitectura

Los patrones de arquitectura están claramente dentro de la disciplina arquitectónica, solapándose con los estilos (Kicillof, et al., Marzo de 2004), aún cuando se reconoce que los patrones se refieren más bien a prácticas de reutilización y los estilos conciernen a teorías sobre la estructuras de los sistemas a veces más formales que concretas. Algunas formulaciones que describen patrones pueden leerse como si se refirieran a estilos, y también viceversa. Puede apreciarse que toda vez que surge la pregunta de qué hacer con los estilos, de inmediato aparece una respuesta que apunta para el lado de las prácticas y los patrones.

Los patrones, aún los que se han caracterizado como arquitectónicos, se encuentran más ligados al uso y más cerca del plano físico, sin disponer todavía de un lenguaje de especificación y sin estar acomodados en una taxonomía que ordene razonablemente sus clases y en un mapa que los sitúe inequívocamente en la trama de los grandes marcos de referencia (Kicillof, et al., Marzo de 2004).

En cuanto a los patrones de arquitectura, su relación con los estilos arquitectónicos es perceptible, pero indirecta y variable incluso dentro de la obra de un mismo autor. Se asume el concepto y clasificación que se propone en (Buschmann, 4 de mayo 2007), donde los patrones de arquitectura “expresan un esquema de organización estructural para los sistemas de software. Proporcionan un conjunto de subsistemas predefinidos, especifica sus responsabilidades e incluye reglas y lineamientos para organizar la relación entre ellos”.

La selección de un patrón de arquitectura o la combinación de varios es solo el primer paso cuando se diseña la arquitectura de un sistema software.

1.4.1.2 Patrones de diseño

Un patrón de diseño es un conjunto de reglas que describen como afrontar tareas y solucionar problemas que surgen durante el desarrollo de software. Se pueden agrupar en tres conjuntos de patrones según su finalidad (Ganma, et al., 1995):

Patrones de creación: Ayudan a que el sistema sea independiente de cómo sus objetos son creados, compuestos o representados. El objetivo de estos patrones es abstraer el proceso de instanciación y ocultar los detalles de cómo los objetos son creados o inicializados. Proporcionan a los programas una mayor flexibilidad para decidir que objetos usar.

Patrones estructurales: Los patrones estructurales describen como las clases y objetos pueden ser combinados para formar grandes estructuras y proporcionar nuevas funcionalidades. Estos objetos adicionados pueden ser incluso objetos simples u objetos compuestos.

Patrones de comportamiento: Los patrones de comportamiento nos ayudan a definir la comunicación e iteración entre los objetos de un sistema. El propósito de este patrón es reducir el acoplamiento entre los objetos.

Estos patrones se pueden clasificar teniendo en cuenta el propósito y el ámbito donde se aplique, en (Ganma, et al., 1995) se propone la clasificación que muestra el Anexo 1. Anexo 1 Clasificación de los patrones de diseño.Existen otras formas de clasificar los patrones, por ejemplo en dependencia de cómo se referencian entre ellos, debido a que algunos son utilizados juntos.

Los patrones de diseño no solo ofrecen una vía útil para desarrollar software robusto de forma rápida, sino que también proveen una forma de encapsular muchas ideas de forma rápida y amigable.

A continuación se especifican algunos de estos patrones, puesto que constituyen los más empleados en aplicaciones web (Herrington, 2006).

Factory Method (Método de fabricación): Este tipo de patrón es usado frecuentemente debido a su utilidad. Su objetivo es devolver una instancia de múltiples tipos de objetos, normalmente todos estos objetos provienen de una misma clase padre mientras que se diferencian entre ellos por algún aspecto de comportamiento, ocultando al usuario la casuística para elegir el subtipo a crear.

Singleton (Instancia única): Garantiza la existencia de una única instancia para una clase y la creación de un mecanismo de acceso global a dicha instancia.

Observer (Observador): Define una dependencia entre objetos, de forma que cuando un objeto cambie de estado se notifique y actualicen automáticamente todos los objetos que dependen de él. Asume que el objeto que contiene los datos es independiente de los objetos que muestran los datos de manera que son estos los que “observan” los cambios en dichos datos.

Command (Orden): Especifica una forma simple de separar la ejecución de un comando del entorno que generó dicho comando. Encapsula una operación en un objeto, permitiendo ejecutar dicha operación sin necesidad de conocer el contenido de la misma.

Front Controller (Controlador frontal): Es un patrón muy utilizado por las aplicaciones web, que describe básicamente cómo construir un sólo punto de acceso para todas las peticiones de una aplicación web. Escucha las peticiones que vienen desde una URL, y se encarga de llamar al controlador específico, posteriormente llama a la acción deseada del controlador.

1.5 Lenguajes y Tecnologías

Para el desarrollo del sistema propuesto se pudiera haber valorado lenguajes de programación web como: Java, Python, Ruby o Perl, pero uno de los requisitos no funcionales del cliente especificaba que se implementara la solución sobre PHP, con el objetivo de no entrar en contraste con el resto de las aplicaciones y servicios que se ejecutan actualmente en los servidores del Ministerio del Poder Popular para la Energía y Petróleo de la República Bolivariana de Venezuela (MENPET). En aras de lograr interactividad, velocidad y usabilidad en la aplicación se utiliza una técnica de desarrollo web para crear aplicaciones RIA, es el caso de AJAX, tecnología explicada más adelante.

1.5.1 Lenguaje de Programación: PHP

PHP es un lenguaje de programación interpretado, es un acrónimo recursivo que significa PHP Hypertext Pre-processor. Diseñado originalmente para la creación de páginas web dinámicas, de propósito general ampliamente difundido, principalmente en interpretación del lado del servidor (server-side scripting) aunque puede ser incrustado dentro de código HTML. Publicado bajo la PHP License, considerada como software libre.

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

- Capacidad de expandir su potencial utilizando la enorme cantidad de módulos (llamados extensiones).

- Posee una amplia documentación.
- Es libre, por lo que se presenta como una alternativa de fácil acceso para todos.
- Permite las técnicas de Programación Orientada a Objetos.
- Biblioteca nativa de funciones sumamente amplia e incluida.
- No requiere definición de tipos de variables.
- Tiene manejo de excepciones (desde PHP5).

1.5.2 Tecnología AJAX

AJAX, acrónimo de **A**synchronous **J**avaScript **A**nd **X**ML (JavaScript asíncrono y XML), es una técnica de desarrollo web para crear aplicaciones interactivas. 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.

Ajax es una tecnología asíncrona, en el sentido de que los datos adicionales se requieren al servidor y se cargan en segundo plano sin interferir con la visualización ni el comportamiento de la página. JavaScript es el lenguaje interpretado (scripting language) en el que normalmente se efectúan las funciones de llamada de Ajax mientras que el acceso a los datos se realiza mediante XMLHttpRequest, objeto disponible en los navegadores actuales. En cualquier caso, no es necesario que el contenido asíncrono esté formateado en XML.

Es una técnica válida para múltiples plataformas y utilizable en muchos sistemas operativos y navegadores, puesto que está basado en estándares abiertos como JavaScript y Document Object Model (DOM).

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 para mostrar e interactuar dinámicamente con la información presentada.
- XMLHttpRequest o el objeto iframe: para intercambiar datos de forma asíncrona con el servidor web.
- XML y JSON: Formato usado generalmente para la transferencia de datos solicitados al servidor.

1.5.3 Lenguaje de Modelado

Para expresar las características del sistema, o en otras palabras, modelarlo, se aplica una convención gráfica o algún lenguaje avanzado de alto nivel de abstracción.

1.5.3.1 Lenguajes de Descripción Arquitectónica

Los lenguajes de descripción de arquitecturas, o ADLs (Architecture Description Languages), 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 nos proveen una forma de construir una estructura de alto nivel.

La definición más simple es la de Tracz (Wolf, Enero de 1997) que define un ADL como una entidad consistente en cuatro “Cs”: componentes, conectores, configuraciones y restricciones (constraints).

Los ADLs que existen actualmente en la industria rondan la cifra de veinticinco; cuatro o cinco de ellos han experimentado dificultades en su desarrollo o no se han impuesto en el mercado de las herramientas de arquitectura. Pese a que no existe hasta hoy una definición consensuada y unívoca de ADL, se acepta comúnmente aquella que expresa que un ADL debe proporcionar un modelo explícito de componentes, conectores y sus respectivas configuraciones (Kicillof, et al., Marzo de 2004).

1.5.3.2 Lenguaje Unificado de Modelado

El Lenguaje Unificado de Modelado (UML) es un lenguaje de modelado visual que se usa para especificar, visualizar, construir y documentar artefactos de un sistema de software (Rumbaugh, et al., 1999). Su objetivo es representar el conocimiento acerca de los sistemas que se pretenden construir y las decisiones tomadas durante su desarrollo, tanto los representados por diagramas estáticos (Casos de Uso, diagrama de clases, etc.) como los dinámicos (Diagramas de actividades, interacción, etc.).

La representación de un software está formado por 4+1 vistas o modelos parciales separados, relacionados entre sí, las cuales son:

- Vista de casos de uso: Como la perciben los usuarios, analistas y encargados de las pruebas.
- Vista lógica: Comprende las clases, interfaces y colaboraciones que forman el vocabulario del problema y su solución.
- Vista de implementación: Incluye los componentes y archivos sobre el sistema físico.

- Vista de procesos: Conforman los hilos y procesos que forman los mecanismos de sincronización y concurrencia.
- Vista de despliegue: Comprende los nodos que forman la topología de hardware sobre la que se ejecuta el sistema.

Aunque no existe de forma explícita una vista arquitectónica, estas cinco vistas pretenden describir, en su conjunto, la arquitectura del sistema (Kruchten, 1995).

A manera de resumen se pudiera decir que en la comunidad de arquitectos existen dos facciones claramente definidas; la primera, vinculada con el mundo de Rational y UML, impulsa el uso de UML como si fuera un ADL; la segunda ha señalado las limitaciones de UML como lenguaje para definir arquitecturas por la forma de expresar ciertas características, sobre todo dinámicas de las estructuras que no es suficiente para los arquitectos (Kicillof, et al., Marzo de 2004). Se asume la primera variante en principio, por ir en consecuencia con la estandarización del proceso de desarrollo de software en el contexto donde se inscribe el proyecto.

1.6 Frameworks

En el mundo de las aplicaciones web existen varios frameworks o marcos de trabajo que facilitan y automatizan muchas funcionalidades comunes en el desarrollo de este tipo de software. Las principales funcionalidades que proveen son, entre otras, el acceso a los datos mediante ORM (*Mapeo objeto-relacional, según siglas en inglés*), la implementación de un patrón arquitectónico como Modelo-Vista-Controlador, así como algunos asistentes para el diseño de interfaces de usuario de mayor calidad en menos tiempo. Existen varios marcos de trabajo para PHP como CakePHP, Zend Framework, Symfony, CodeIgniter, Prado, P4A, entre otros. Los mejores y más utilizados por los desarrolladores a nivel mundial coinciden con los siguientes: CakePHP, Zend Framework y Symfony, por lo que fueron objeto de análisis para su evaluación como posibles a utilizar.

1.6.1 CakePHP

CakePHP es un marco de desarrollo rápido que provee una arquitectura extensible para desarrollar, mantener y desplegar aplicaciones web. Utilizando conocidos patrones de diseño como MVC e incluye un

ORM. Dentro de la convención del paradigma de configuración, CakePHP reduce los costos de desarrollo y ayuda a los desarrolladores a escribir menos código, sus principales características son (cakephp.org):

- Compatible con PHP4 y PHP5.
- CRUD (*Create, Read, Update and Delete*) de la base de datos integrado.
- URLs amigables.
- Sistema de plantillas rápido y flexible.
- Helpers para AJAX, Javascript, HTML, formularios y más.
- Trabaja en cualquier subdirectorio del sitio.
- Validación integrada.
- Scaffolding de las aplicaciones.
- Listas de control de acceso.
- Componentes de seguridad y sesión.
- Se distribuye bajo licencia MIT (*Instituto Tecnológico de Massachusetts*), es desarrollado por este instituto y es multiplataforma.

1.6.2 Zend Framework

Framework para desarrollo de aplicaciones y servicios web, brinda soluciones para construir sitios modernos, robustos y seguros. Además es “Open Source” y trabaja con PHP 5, a diferencia de CakePHP que trabaja con PHP 4 y PHP 5. La principal ventaja de Zend Framework es que es desarrollado por Zend (empresa que respalda comercialmente a PHP). Las principales características de dicho framework son (Zend Company, 2009):

- Implementa MVC.
- Cuenta con módulos para manejar archivos PDF, canales RSS, Web Services (Amazon, Flickr, Yahoo), entre otros.
- Para el acceso a base de datos, balancea el ORM con eficiencia y simplicidad.
- Completa documentación y pruebas de alta calidad.
- Soporte avanzado para i18n (internacionalización).
- Robustas clases para autenticación y filtrado de entrada.
- Zend Framework es multiplataforma, desarrollado y mantenido por Zend Technologies y se puede utilizar bajo licencia “New BSD License”.

1.6.3 Symfony

Symfony es un completo framework diseñado para optimizar el desarrollo de las aplicaciones web mediante algunas de sus principales características. Separa la lógica de negocio, la lógica de servidor y la presentación de la aplicación web. Proporciona varias herramientas y clases encaminadas a reducir el tiempo de desarrollo de una aplicación compleja. Además, automatiza las tareas más comunes, permitiendo al desarrollador dedicarse por completo a los aspectos específicos de cada aplicación. El resultado de todas estas ventajas es no "reinventar la rueda" cada vez que se crea una nueva aplicación web (Potencier, 2009).

Es compatible con la mayoría de gestores de bases de datos, como MySQL, PostgreSQL, Oracle y Microsoft SQL Server, sus principales características son:

- 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.
- Implementa Front Controller como patrón derivado del MVC.
- Acceso a datos a través de ORM.
- Incluye soporte para AJAX.
- Provee varios "helpers" para la creación de interfaces.
- Soporta la instalación de extensiones o "plugins" para añadir nuevas funcionalidades.
- URLs amigables y configurables a través de un poderoso sistema de enrutamiento.
- Sencillo de usar en la mayoría de casos, pero lo suficientemente flexible como para adaptarse a los casos más complejos.
- Basado en la premisa de "convenir en vez de configurar", en la que el desarrollador solo debe configurar aquello que no es convencional.
- Sigue la mayoría de las mejores prácticas y patrones de diseño para la web.
- Preparado para aplicaciones empresariales y se adapta a las políticas y arquitecturas propias de cada empresa, además de ser lo suficientemente estable como para desarrollar aplicaciones a largo plazo.
- Código fácil de leer que incluye comentarios de phpDocumentor y que permite un mantenimiento muy sencillo.
- Fácil de extender, lo que permite su integración con las librerías de otros fabricantes.
- Symfony es multiplataforma, desarrollado por Sensio Labs, Inc y liberado por una licencia de tipo MIT para software libre compatible con la GPL (*GNU General Public License*).

Se utilizará en la solución Symfony por las características y facilidades de uso que presenta, además de contar con una comunidad de desarrollo activa y eficiente para la elaboración de nuevos plugins.

1.7 Metodología de desarrollo

Un Proceso de Desarrollo de Software es la definición del conjunto de actividades que guían los esfuerzos de las personas implicadas en el proyecto, a modo de plantilla que explica los pasos necesarios para terminar el proyecto (Jacobson, et al., 2000).

1.7.1 Proceso Unificado de Desarrollo (RUP)

RUP es un proceso de desarrollo de software dirigido por casos de uso, centrado en la arquitectura, iterativo e incremental. Es una metodología orientada a objetos que utiliza a UML como lenguaje de modelado.

Según RUP, la arquitectura involucra los elementos más significativos del sistema y está influenciada entre otros por plataformas software, sistemas operativos, manejadores de bases de datos, protocolos, consideraciones de desarrollo como sistemas heredados y requerimientos no funcionales. Se representa mediante varias vistas que se centran en aspectos concretos.

1.7.1.1 Fases de RUP

El ciclo de vida de un software se centra en cuatro fases fundamentales:

- Inicio o Conceptualización.
- Elaboración.
- Construcción.
- Transición.

En cada una de ellas la arquitectura juega un papel importante. Durante la fase de inicio se presenta una arquitectura candidata, se determinan algunos de los casos de uso arquitectónicamente significativos que incluyen los más necesarios para el cliente en esta versión. En la fase de elaboración el objetivo fundamental es la elaboración de la línea base de la arquitectura del sistema, la cual se va elaborando en sucesivas iteraciones de esta fase, hasta construir y validar el “esqueleto” del sistema. Una vez en la fase de construcción se construye el modelo de implementación y se lleva a cabo el proceso de integración.

1.7.1.2 El rol de Arquitecto de Software

El arquitecto crea la arquitectura junto con otros desarrolladores. El arquitecto posee la responsabilidad técnica más importante en estos aspectos y selecciona patrones de arquitectura y entre productos para establecer las dependencias entre subsistemas para cada uno de esos distintos intereses (Jacobson, et al., 2000).

El rol de arquitecto existe en otras metodologías de desarrollo, con actividades diferentes pero con el mismo objetivo dentro del equipo de desarrollo.

En resumen, el arquitecto de software debe disponer de formación, madurez, visión y una amplia experiencia que permita recuperar cuestiones rápidamente y realizar valoraciones educadas y críticas en ausencia de la información completa. Según la Ayuda de Rational 2003 el arquitecto de software, o los miembros del equipo de arquitectura, deben combinar habilidades como: experiencia en el dominio del problema, liderazgo para dirigir y tomar decisiones críticas bajo presión, proactividad. El arquitecto es la fuerza técnica dirigente detrás del proyecto, no un visionario o un soñador.

1.7.1.3 Actividades y Artefactos

Como se analizó anteriormente, durante todas las fases de vida del desarrollo de software se realizan actividades concernientes a la arquitectura, de las cuales se pudieran citar:

- Priorizar los Casos de Uso.
- Análisis de la arquitectura.
- Identificar mecanismos de diseño.
- Estructurar el modelo de implementación.
- Reutilización de elementos de diseño existentes.
- Identificar los elementos de diseño.
- Describir la arquitectura en tiempo de ejecución.

El arquitecto es responsable de artefactos como:

- Documento Descripción de la Arquitectura (4 Vistas).
- Modelo de Despliegue.
- Modelo de Implementación.
- Protocolos.
- Interfaces.

En el análisis de metodologías ágiles como Scrum y XP se hace menos énfasis en la arquitectura del software mientras que en las no ágiles la arquitectura del software es esencial y se expresa mediante modelos. Se utiliza en el proyecto como metodología RUP, primeramente por la vasta documentación y soporte por herramientas CASE que posee, además de ser la estandarizada en la universidad para el desarrollo de proyectos de este tipo. Sin embargo, se han introducido principios de Scrum en la forma de trabajo, como la reunión de seguimiento diario, que se realiza todos los días a primera hora de la mañana y donde participan todos los miembros del proyecto. Durante esta reunión, el equipo sincroniza su trabajo, progreso e informa cualquier impedimento.

1.8 Calidad en la Arquitectura de Software

La arquitectura del software ha pasado a ser un artefacto esencial dentro del proceso de desarrollo de software moderno debido a sus múltiples usos, pero llegar a construir una arquitectura para el software que sea a la vez apropiada para dar cabida a las exigencias de las distintas partes interesadas y buena en términos absolutos es una tarea que dista muchísimo de ser sencilla.

Una de las definiciones de calidad de software plantea que es:

“la concordancia con los requisitos funcionales y de rendimiento establecidos con los estándares de desarrollo explícitamente documentados y con las características implícitas que se espera de todo software desarrollado de forma profesional” (Pressman, 2002)**.**

La definición de la ISO sentencia que es:

“la totalidad de rasgos y atributos de un producto de software que le apoyan en su capacidad de satisfacer sus necesidades explícitas o implícitas” (ISO/IEC, 1998).

La calidad arquitectónica es un elemento determinante en la calidad de software. Barbacci establece que el desarrollo de formas sistemáticas para relacionar atributos de calidad de un sistema a su arquitectura provee una base para la toma de decisiones objetivas sobre acuerdos de diseño y permite realizar predicciones razonablemente exactas sobre los atributos del sistema que son libres de prejuicios y asunciones no triviales.

Los atributos de calidad son propiedades o características del sistema, que pueden afectar el grado de satisfacción de los interesados, se caracterizan por ser propiedades medibles y se capturan generalmente como requerimientos no funcionales (Bastarrica, 2003).

En (Bass, et al., 2003) se establece una clasificación de los atributos de calidad en dos categorías:

Observables vía ejecución (Externos): aquellos atributos que se determinan del comportamiento del sistema en tiempo de ejecución.

No observables vía ejecución (Internos): aquellos atributos que se establecen durante el desarrollo del sistema.

Dentro de los atributos observables se encuentra:

Disponibilidad (Availability): Es la medida de disponibilidad del sistema para el uso.

Funcionalidad (Functionality): Habilidad del sistema para realizar el trabajo para el cual fue concebido

Desempeño (Performance): Grado en el cual un sistema o componente cumple con su funcionalidad, dentro de ciertas restricciones dadas, como velocidad, exactitud o uso de memoria.

Para más detalle ver Anexo 2.

Dentro de la categoría de los no observables están:

Integrabilidad (Integrability): Es la medida en que trabajan correctamente componentes del sistema que fueron desarrollados separadamente al ser integrados.

Interoperabilidad (Interoperability): Es la medida de la habilidad de que un grupo de partes del sistema trabajen con otro sistema.

Portabilidad (Portability): Es la habilidad del sistema para ser ejecutado en diferentes ambientes de cómputo.

Escalabilidad (Scalability): Es el grado con el que se pueden ampliar el diseño arquitectónico, de datos o procedimental.

Para más detalle ver Anexo 3.

La organización y descomposición de los atributos de calidad ha permitido el establecimiento de modelos específicos para efectos de la evaluación de la calidad arquitectónica. Uno de estos modelos es la ISO/IEC 9126 que fue adaptada para arquitecturas de software en el 2003. El modelo se basa en los

atributos de calidad que se relacionan directamente con la arquitectura: funcionalidad, confiabilidad, eficiencia, mantenibilidad y portabilidad. El Anexo 4 muestra una tabla con la correspondencia de estos atributos y los elementos arquitectónicos.

Hacer un adecuado balance entre los distintos atributos de calidad es esencial para obtener un sistema que cumpla las expectativas de las distintas partes interesadas, pero el diseño de una arquitectura que tenga en cuenta los atributos deseados, sólo permitirá que el sistema resultante tenga estas características, pero no lo garantiza.

Si las decisiones arquitectónicas determinan los atributos de calidad del sistema, entonces es posible evaluar las decisiones arquitectónicas con respecto a su impacto sobre dichos atributos. Se relacionan a continuación algunos métodos para la evaluación de la arquitectura.

1.8.1 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.

1.8.2 Método de Análisis de Acuerdos de Arquitectura (ATAM)

Este método de evaluación obtiene su nombre no solo porque nos dice cuán bien una arquitectura particular satisface las metas de calidad, sino que provee ideas de cómo esas metas de calidad interactúan entre ellas, como realizan concesiones mutuas (tradeoff).

1.8.3 Método de evaluación para Arquitecturas Parciales (ARID)

Método conveniente para realizar la evaluación de diseños parciales en las etapas tempranas del desarrollo. Utilizado para la evaluación de diseños detallados de unidades del software como los componentes o módulos. Gira en torno a la calidad y completitud de la documentación y la suficiencia, el ajuste y la conveniencia de los servicios que provee el diseño propuesto.

El método SAAM se sugiere cuando el atributo de calidad modificabilidad es el de mayor interés. Mientras que el método ATAM es más profundo para evaluar aspectos relacionados con la arquitectura, como el desempeño o la confiabilidad. 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.

1.9 Herramientas Case

El nombre de “Herramienta CASE” viene de Computer Aided Software Engineering (CASE) o Ingeniería de Software Asistida por Computación y consiste en una o varias herramientas que permiten organizar y manejar cierta información de un proyecto informático. Se escogió 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.

1.9.1 Visual Paradigm

Soporta notación UML 2.x, capacidades de ingeniería inversa y directa, generación de código, importación desde Rational Rose, generación de código e ingeniería inversa a la vez de los lenguajes: Java, C++, CORBA IDL, PHP, XML Schema, Ada y Python. Adicional, soporta la generación de código en: C#, VB .NET, Object Definition Language (ODL), Flash ActionScript, Delphi, Perl, Objective-C y Ruby. Además del soporte de ingeniería inversa de: Java class, .NET (dll, exe), JDBC y ficheros mapeados de Hibernate .

Principales características:

- Licencia: Gratuita y Comercial.
- Producto de calidad.
- Soporta aplicaciones Web.
- Varios idiomas.
- Fácil de instalar y actualizar.

1.10 Gestores de Bases de Datos

Es un tipo de software muy específico, dedicado a servir de interfaz entre la base de datos, el usuario y las aplicaciones que la utilizan. El propósito general de los sistemas de gestión de base de datos es el de manejar de manera clara, sencilla y ordenada un conjunto de datos que posteriormente se convertirán en información relevante para una organización.

1.10.1 MySQL

MySQL es un SGBD relacional, multihilo y multiusuario con más de seis millones de instalaciones (Schumacher, et al., 2008). MySQL AB —desde enero de 2008 una subsidiaria de Sun Microsystems— desarrolla MySQL como software libre en un esquema de licenciamiento dual.

El software MySQL proporciona un servidor de base de datos SQL *(Structured Query Language)* muy rápido. Dentro de las principales características de MySQL se encuentran (Sun Microsystems, 2008):

- Escalabilidad: es posible manipular bases de datos enormes.
- Escrito en C y C++ y probado con multitud de compiladores y dispone de APIs para plataformas diferentes.
- Conectividad: es decir, permite conexiones entre diferentes máquinas con distintos sistemas operativos.
- Proporciona sistemas de almacenamiento transaccional y no transaccional.
- Velocidad en la lectura de datos cuando utiliza el motor no transaccional MyISAM, pero puede provocar problemas de integridad en entornos de alta concurrencia en la modificación.

1.10.2 Postgre SQL

PostgreSQL es un SGBD relacional orientada a objetos de software libre, publicado bajo la licencia BSD. A continuación se enumeran las principales características:

- Implementación del estándar SQL92/SQL99.
- Soporta distintos tipos de datos.
- Permite la creación de tipos propios.
- Incorpora una estructura de datos array.
- Incorpora funciones de diversa índole: manejo de fechas, geométricas, orientadas a operaciones con redes, entre otras.

- Permite la declaración de funciones propias, así como la definición de disparadores.
- Soporta el uso de índices, reglas y vistas.
- Incluye herencia entre tablas.
- Permite la gestión de diferentes usuarios, como también los permisos asignados a cada uno de ellos.

De acuerdo al tipo de sistema que se va a implementar y las características del mismo de poder ser utilizado a nivel empresarial, se escogió PostgreSQL como gestor de base de datos, además de brindar soporte por una comunidad de desarrollo internacional, elemento este que no posee MySQL pues la Sun Microsystem no provee servicios de soporte a las versiones libres de este gestor.

1.11 Ambiente de Desarrollo

Un entorno de desarrollo o IDE (*Integrated Development Environment*), es una aplicación o conjunto de estas en las que se combinan las tecnologías a utilizar para el desarrollo del software, permiten entre otras tareas: escribir el código, compilarlo o ejecutarlo, detectar errores, gestionar versiones y crear instaladores.

1.11.1 Eclipse

Eclipse es un IDE multiplataforma y multilenguaje. El punto fuerte de Eclipse es su modularidad ya que este IDE, a pesar de haber sido desarrollado inicialmente para el lenguaje Java, actualmente gracias a las facilidades que brinda para la inclusión de plugins así como para el desarrollo de otros nuevos, posee soporte para casi todos los lenguajes de programación existentes, ya sean compilados o interpretados. Se integra fácilmente con herramientas CASE como Visual Paradigm, con sistemas de control de versiones como Concurrent Versions System (CVS) y Subversion (SVN). Es liberado bajo la “Eclipse Public License”, un tipo de licencia compatible con GNU/GPL por lo que es Software Libre (The Eclipse Foundation, 2008).

1.11.2 Zend Studio

Zend Studio o Zend Development Environment es un completo entorno integrado de desarrolllo para el lenguaje de programación PHP. Está escrito en Java, y está disponible para las plataformas Microsoft

Windows, Mac OS X y GNU/Linux. Junto con su contraparte Zend Platform, son la propuesta de Zend Technologies para el desarrollo de aplicaciones Web utilizando PHP, actuando Zend Studio como la parte cliente y Zend Platform como la parte servidora. Se trata en ambos casos de software comercial, lo cual contrasta con el hecho de que PHP es software libre (Zend Technologies Ltd, 2008).

1.11.3 Zend Studio for Eclipse

Fue creado por Zend Technologies Inc. aprovechando las bondades que brinda Eclipse e integrando Zend Studio dentro de éste como un plugin más. Se nutre de todas las características presentes en los dos IDEs y es distribuido comercialmente como software privativo. Se utiliza como IDE en el de desarrollo del sistema debido a sus potencialidades.

1.12 Conclusiones Parciales

A partir de los objetivos propuestos para este capítulo se puede concluir lo siguiente:

- Se realizó el estudio del arte de las herramientas para la generación de reportes en el mundo, analizando las ventajas y desventajas de cada una de ellas.
- Se definieron los principales conceptos relacionados con la arquitectura de software y se fundamentaron los principales estilos y patrones.
- 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.

2 Representación Arquitectónica

El sistema de Gestión de Reportes Dinámicos (SGRD) está diseñado con una arquitectura modular y distribuida que ayuda a obtener tanto escalabilidad como flexibilidad. Los procesos se distribuyen entre varios componentes que se pueden ampliar e integrar en soluciones personalizadas. Una típica aplicación para la generación de informes atraviesa por las tres etapas del ciclo de vida de los reportes: creación, administración y entrega; SGRD ofrece todas las herramientas necesarias para soportar estos procesos, la Figura 3 muestra la integración de las diferentes aplicaciones del sistema en el ciclo de vida.

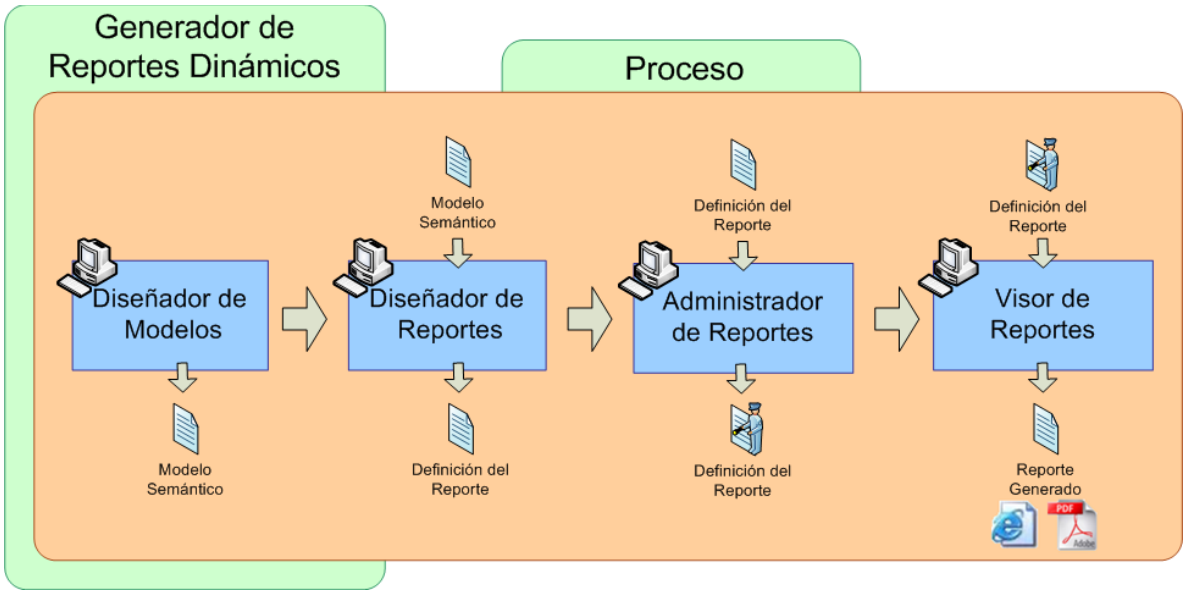


Figura 3 Ciclo de vida de los reportes

Para soportar los procesos de creación de informes se utilizan las aplicaciones Diseñador de modelos y Diseñador de reportes; la primera comienza el ciclo creando una abstracción de la fuente de datos denominada modelo semántico, el cual describe la estructura de los datos; el Diseñador de reportes se

encarga de la conformación del informe, teniendo como entrada el modelo semántico y generando como salida la definición del reporte. En la fase de administración, el Administrador de reportes permite ejecutar tareas para la gestión de los modelos semánticos y las definiciones de los reportes. La última etapa en el ciclo de vida de los reportes es la entrega, SGRD soporta este proceso mediante el Administrador de reportes, específicamente con el procesador de programación y entrega y el Visor de reportes, el cual posibilta la visualización y exportación del informe a diferentes formatos.

El proyecto consta de 3 grandes subsistemas, los cuales conforman los principales elementos del sistema (ver Figura 4):

Un Servidor de Reportes que administra y procesa los reportes en diferentes formatos. Las salidas incluidas pueden ser HTML, PDF, XLS, entre otras.

Un Conjunto de Aplicaciones que se utilizan para diseñar, administrar y visualizar reportes.

Una Interfaz de Comunicación que permita a terceros programas generar reportes y utilizarlos.

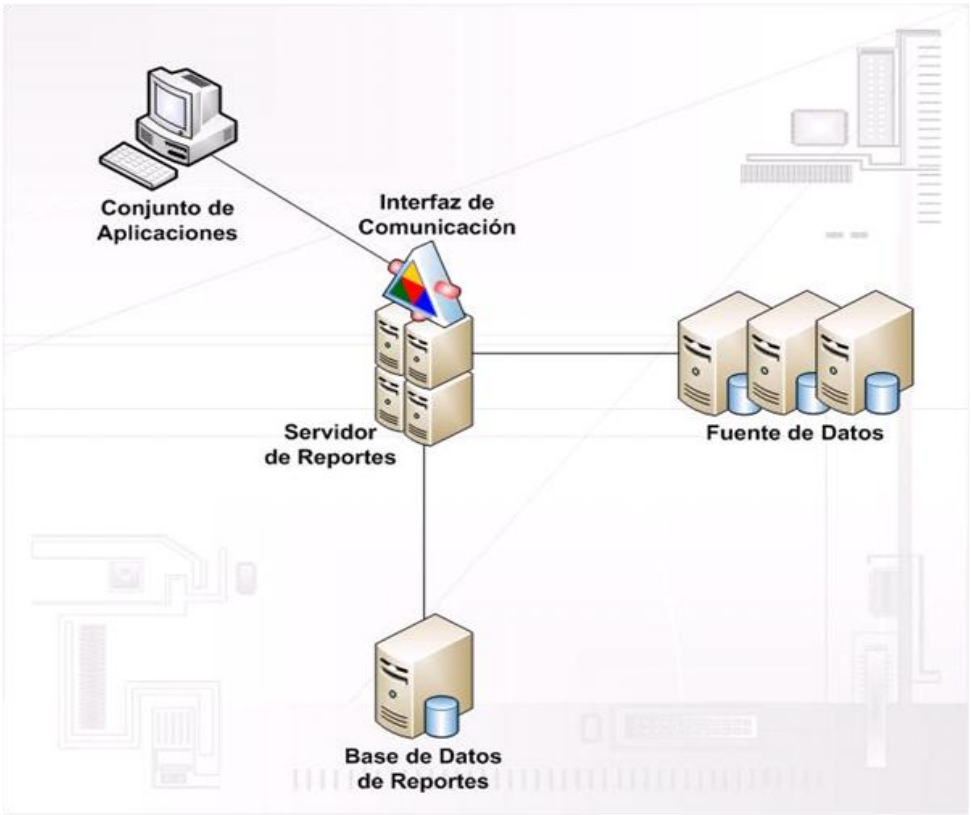


Figura 4 Representación de los subsistemas

Servidor de Reportes

La Figura 5 muestra la estructura de este subsistema, se implementa como un servicio del sistema operativo (SO) y como un servicio Web. Contiene servicios que constituyen un conjunto de interfaces de programación que las aplicaciones clientes pueden utilizar para obtener acceso al servidor de reportes. Este subsistema es un servidor sin estado que almacena todas las propiedades, los objetos y los metadatos en una base de datos de PostgreSQL. Los datos almacenados incluyen reportes publicados, definiciones de reporte y la jerarquía de carpetas que proporciona el direccionamiento de todos los elementos que administra el servidor de reportes.

A través de sus subcomponentes, el servidor de reportes procesa solicitudes de reportes y permite que estén disponibles para el acceso a petición o la distribución programada. Los subcomponentes del servidor de reportes incluyen procesadores y extensiones. Los procesadores son el núcleo del servidor de reportes. Las extensiones también son procesadores, pero realizan funciones muy específicas.

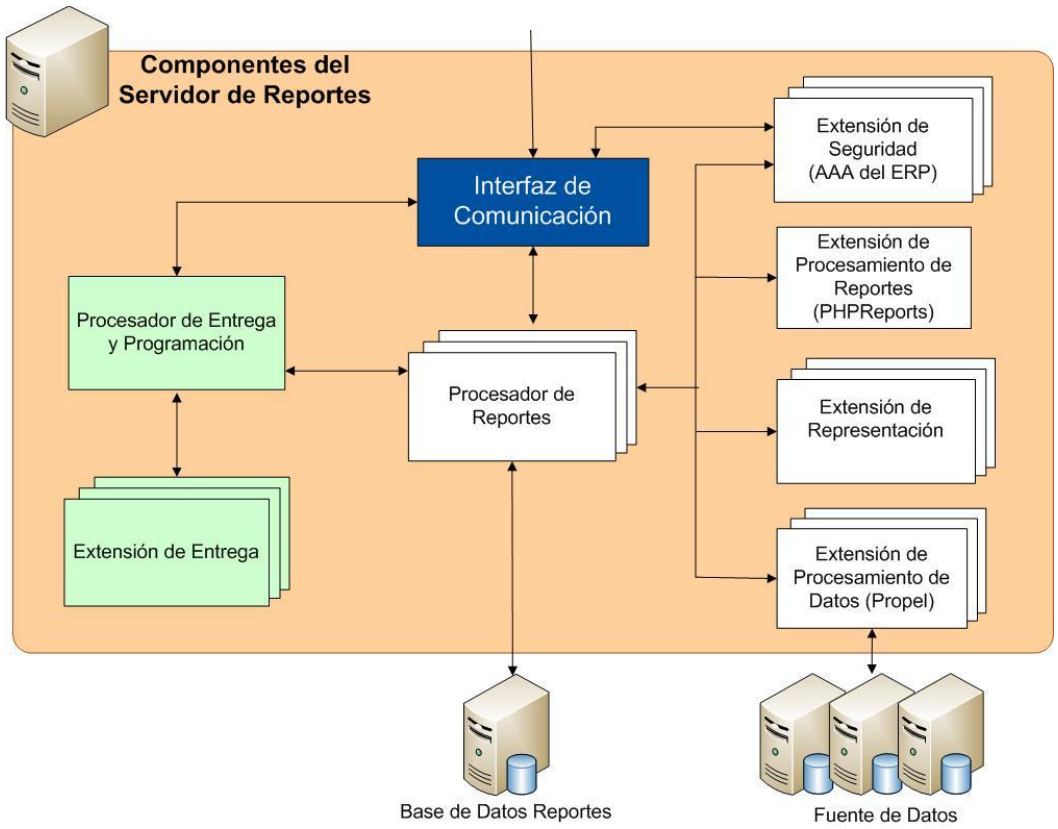


Figura 5 Estructura del Servidor de Reportes

Procesadores

El servidor de reportes incluye dos procesadores que realizan el procesamiento de reportes previo e intermedio, así como operaciones programadas y de entrega. El procesador de reportes recupera la definición del reporte, combina información de diseño con datos de la extensión de procesamiento de datos y representa el reporte en el formato solicitado. El Procesador de Programación y Entrega procesa reportes desencadenados a partir de una programación y la entrega a destinos.

Procesador de Reportes.

El Procesador de Reportes es un componente del servidor de reportes que inicia su procesamiento mediante solicitudes de un reporte publicado.

Cuando se realiza una solicitud de procesamiento de reportes para un reporte publicado, el Procesador de Reportes obtiene la definición de reporte de la base de datos del sistema, inicializa parámetros y variables que se encuentran en expresiones, y realiza otro procesamiento previo que prepara el reporte para datos, luego la extensión de procesamiento de datos se conecta al origen de datos y recupera los datos combinándolos con el diseño de la definición del reporte. Los datos se procesan por filas para cada sección. Las secciones contienen el encabezado y el pie del informe, los encabezados y pies de grupo y los detalles. En la fase de representación, la extensión de representación lleva a cabo la paginación del informe y procesa las expresiones que no se pueden procesar en la fase de ejecución. A continuación, el reporte se representa en el formato específico del dispositivo correspondiente.

El Procesador de Reportes responde a dos solicitudes:

Solicitud de un reporte a petición: La acción de un usuario que abre un reporte configurado para ejecutarse a petición desencadena el procesamiento de reportes en el servidor de reportes. El Procesador de Reportes recupera la definición de reporte, envía la solicitud de datos a una extensión de procesamiento de datos, combina la definición de informe con los datos, la envía a una extensión de representación y devuelve el informe representado.

Solicitud del Procesador de Programación y Entrega: El Procesador de Reportes recupera la definición de reporte, envía la solicitud de datos a una extensión de procesamiento de datos, combina la definición de reporte con los datos y realiza la entrega en el formato especificado.

Procesador de Programación y Entrega.

El procesador está incluido para permitir operaciones programadas y controlar las extensiones de entrega utilizadas para insertar reportes en buzones de correo electrónico o servidores de ficheros. El Procesador de Programación y Entrega ofrece las siguientes funcionalidades:

- Mantiene una cola de eventos y notificaciones.
- Llama al Procesador de Reportes para ejecutar reportes, procesar suscripciones o borrar reportes. Todo procesamiento de reportes que es consecuencia de un evento de programación se efectúa mediante el servicio del SO del Servidor de Reportes y no con el Servicio Web del Servidor de Reportes.
- Llama a la extensión de entrega especificada en una suscripción para que el reporte se pueda entregar.

Otros componentes y servicios que funcionan con el Procesador de Programación y Entrega controlan otros aspectos de la operación. En resumen, el Procesador de Programación y Entrega se ejecuta en el servicio Servidor de Reportes del SO y utiliza un temporizador para generar eventos programados.

Extensiones.

El servidor de reportes admite extensiones de autenticación, extensiones de procesamiento de datos, extensiones de procesamiento de reportes, extensiones de representación y extensiones de entrega. Un servidor de reportes requiere al menos una extensión de seguridad, una extensión de procesamiento de datos y una extensión de representación.

Extensiones de seguridad.

Las extensiones de seguridad se utilizan para autenticar y autorizar usuarios y grupos para un servidor de reportes. Se utiliza de forma predeterminada el Sistema de Gestión de Seguridad (AAA) implementado para el ERP Cubano.

Extensiones de procesamiento de datos.

Las extensiones de procesamiento de datos se utilizan para consultar un origen de datos. Y cuando esto sucede, devuelven un conjunto de filas planas.

Extensiones de representación.

Las extensiones de representación convierten los datos y la información de diseño del Procesador de Reportes en el formato específico de un dispositivo, ejemplo HTML, Excel, CSV, TSV y PDF.

Extensiones de procesamiento de reportes

La extensión de procesamiento de reportes es la encargada de utilizar la definición del reporte para la conformación de éste. Recurre a la extensión de procesamiento de datos para obtener la información necesaria de las fuentes de datos y finalmente genera el reporte para que pueda ser utilizado posteriormente por la extensión de representación.

Extensiones de entrega

El Procesador de Programación y Entrega utiliza las extensiones de entrega para enviar los reportes a diversas ubicaciones. De forma predeterminada se encuentra la entrega por correo electrónico y a recursos compartidos de un servidor de archivos. La extensión de entrega por correo electrónico envía un mensaje de correo electrónico mediante el Protocolo Simple de Transferencia de Correo (SMTP) que contenga el informe o un vínculo de dirección URL al mismo. La extensión de entrega a servidor de archivos (FTP) guarda reportes en una carpeta compartida en la red. Se puede especificar la ubicación, el formato de representación, el nombre de archivo y las opciones de escritura del archivo que se crea.

Interfaz de comunicación.

La interfaz de comunicación provee completo acceso a las funcionalidades del Servidor de Reportes a través de servicios web. Los servicios web usan Simple Object Access Protocol (SOAP) a través de HTTP y actúan como una interfaz de comunicación entre programas clientes y el Servidor de Reportes.

2.1 Objetivos y Restricciones Arquitectónicas

La arquitectura de software se conforma teniendo en cuenta los requerimientos funcionales y no funcionales del software, sin embargo estos no son los únicos aspectos que inciden en su estructura; restricciones impuestas por el ambiente en que el sistema debe operar, la necesidad de reutilización de algunos componentes, la adopción de estándares y la necesidad de compatibilidad entre diferentes sistemas constituyen ejemplos de elementos a tener en cuenta en el diseño arquitectónico de las aplicaciones. También puede haber un conjunto preexistente de principios y políticas de la arquitectura que guíen el desarrollo del proyecto y deben ser contempladas en esta sección, así como cualquier decisión arquitectónica derivada.

El objetivo de la arquitectura que se presenta es construir un sistema que sea capaz de brindar servicios de generación y administración de reportes.

2.1.1 Interoperabilidad con sistemas legados

Un sistema heredado es un sistema informático que se ha quedado anticuado y que continúa siendo utilizado por el usuario (típicamente una organización o empresa) y no se quiere o no puede ser reemplazado o actualizado de forma sencilla (Jaimes Rojas, 2008).

La interoperabilidad es la habilidad de dos o más sistemas informáticos de intercambiar efectivamente la información que requieren para la ejecución de sus funciones, aún cuando su arquitectura interna sea completamente diferente. Una definición más formal de la ISO establece:

“Habilidad de dos o más sistemas (computadores, medios de comunicación, redes, software y demás componentes) de interactuar y de intercambiar datos de acuerdo con un método definido, de forma de obtener los resultados esperados.”

En la OTI actualmente se encuentra en funcionamiento un sistema para la generación de los reportes del MENPET, el cual por las características que posee se vuelve obsoleto su uso y mantenimiento, evidenciado por el hecho de que los reportes son confeccionados de forma predeterminada, teniendo que modificar el código fuente de la aplicación cada vez que un nuevo informe es requerido. Por tales motivos, de las posibles alternativas existentes para el tratamiento de los sistemas legados se optó por retirar el sistema y reemplazarlo por el nuevo previo consenso con el cliente.

Sin embargo, dada las características del sistema propuesto de poder ser usado en diferentes entornos tanto empresariales como de desarrollo, se hace necesario el uso del estándar XML para el intercambio de datos entre las aplicaciones mediante servicios. Además, es posible la integración mediante URL y servicios que puedan ser usados por terceros, lo que brinda gran interoperabilidad entre las aplicaciones existentes en las empresas u organizaciones donde se despliegue el software y el sistema de Gestión de Reportes Dinámicos.

2.1.2 Estándares de datos

Los estándares de datos son definiciones semánticas que están estructuradas en un modelo. Como el sistema propuesto cuenta con varias aplicaciones es importante integrar éstas de forma coherente, por lo que para evitar posibles problemas de integración se usa XML como uno de los estándares a seguir, debido a la extensibilidad y portabilidad que brinda. Además, la arquitectura presenta una alta extensibilidad, lo que posibilita efectuar cambios sin necesidad de detener su funcionamiento.

Específicamente XML es utilizado para la definición del reporte, la representación del modelo semántico y la consulta fuente de datos.

2.1.2.1 XML Definición del reporte

La definición del reporte es una representación en XML que contiene información de la recuperación de los datos y el diseño del reporte, es un esquema abierto fácilmente extensible por los desarrolladores con la adición de nuevos atributos y elementos.

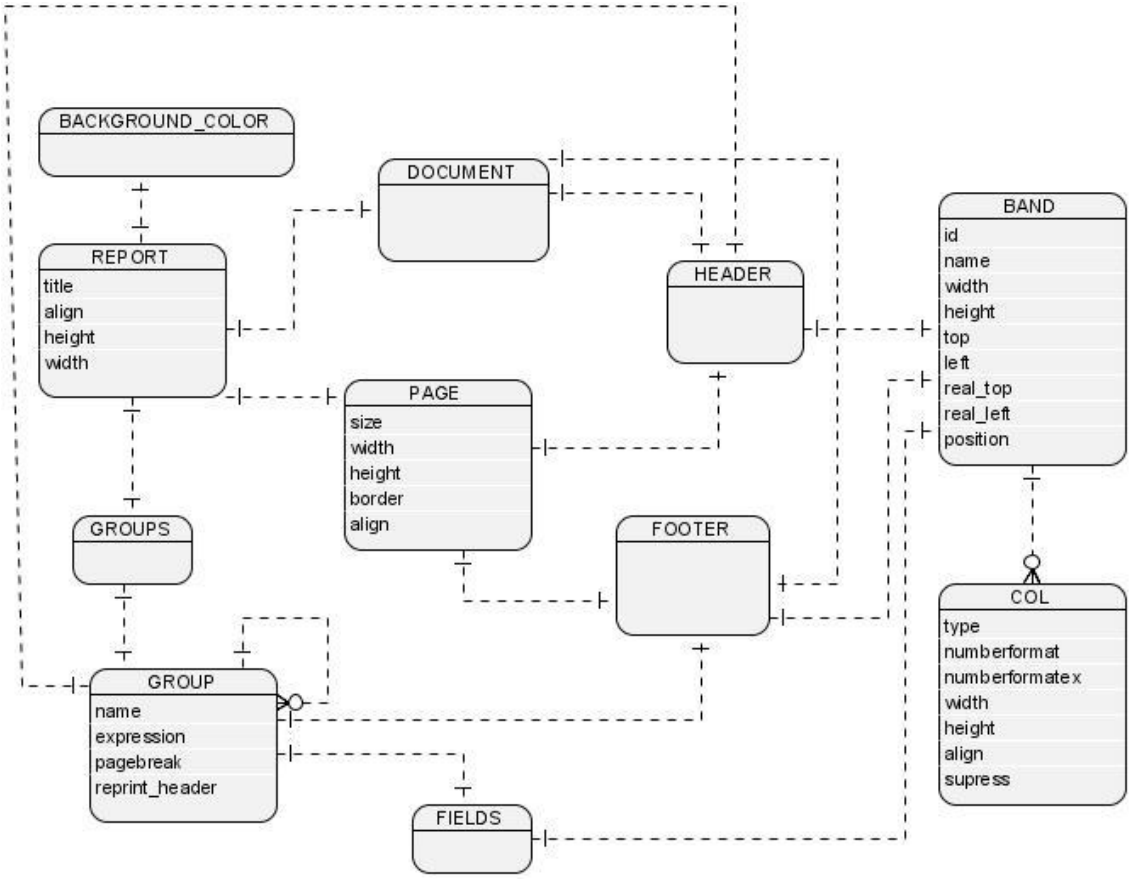


Figura 6 Estructura del XML del reporte

2.1.2.2 XML Modelo semántico

El modelo semántico ofrece una representación del modelo empresarial del esquema de la base de datos a través de la recuperación de los metadatos de la misma, ofreciendo una abstracción a usuarios finales. El XML que lo define recoge información sobre las entidades (tablas, funciones, vistas) presentes en la base de datos además posibilita la definición de atributos como alias para las entidades y campos. La Figura 7 muestra la estructura del XML que define el modelo semántico, el atributo type de la entidad table puede ser de tres tipos: function, table o view, en dependencia de las entidades presentes en el mapeo a la base de datos.

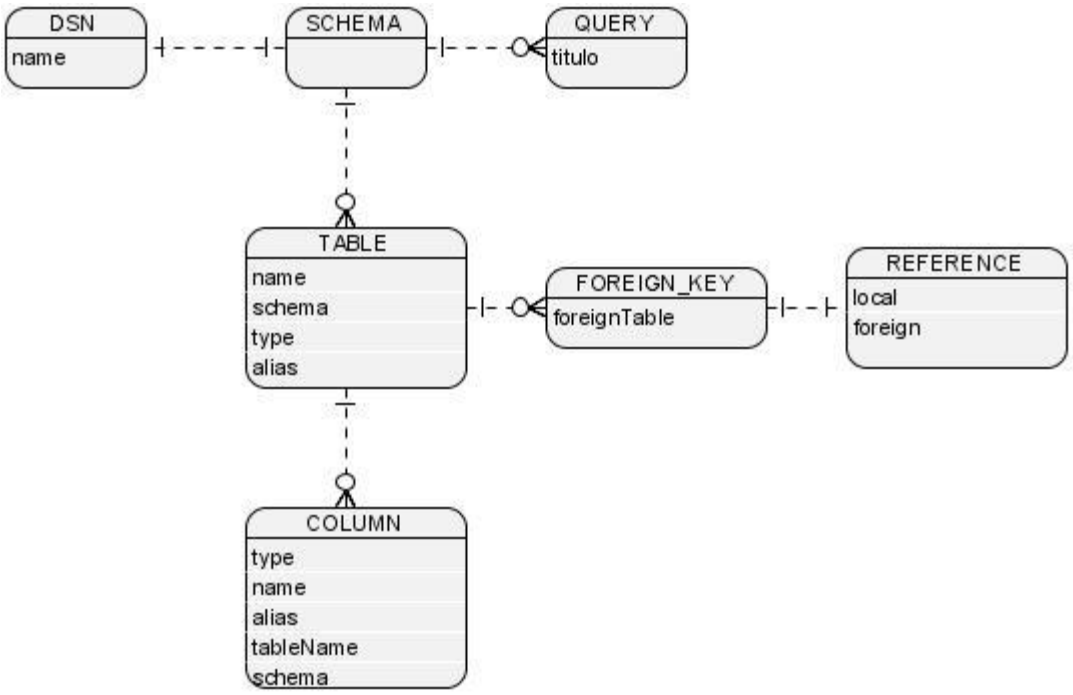


Figura 7 Estructura del XML del modelo semántico

2.1.2.3 XML Consulta

Las consultas que se crean con el sistema son mapeadas mediante un XML (ver Figura 8), esto posibilita un mejor manejo de la misma, establece una forma estándar para la comunicación entre las diferentes aplicaciones.

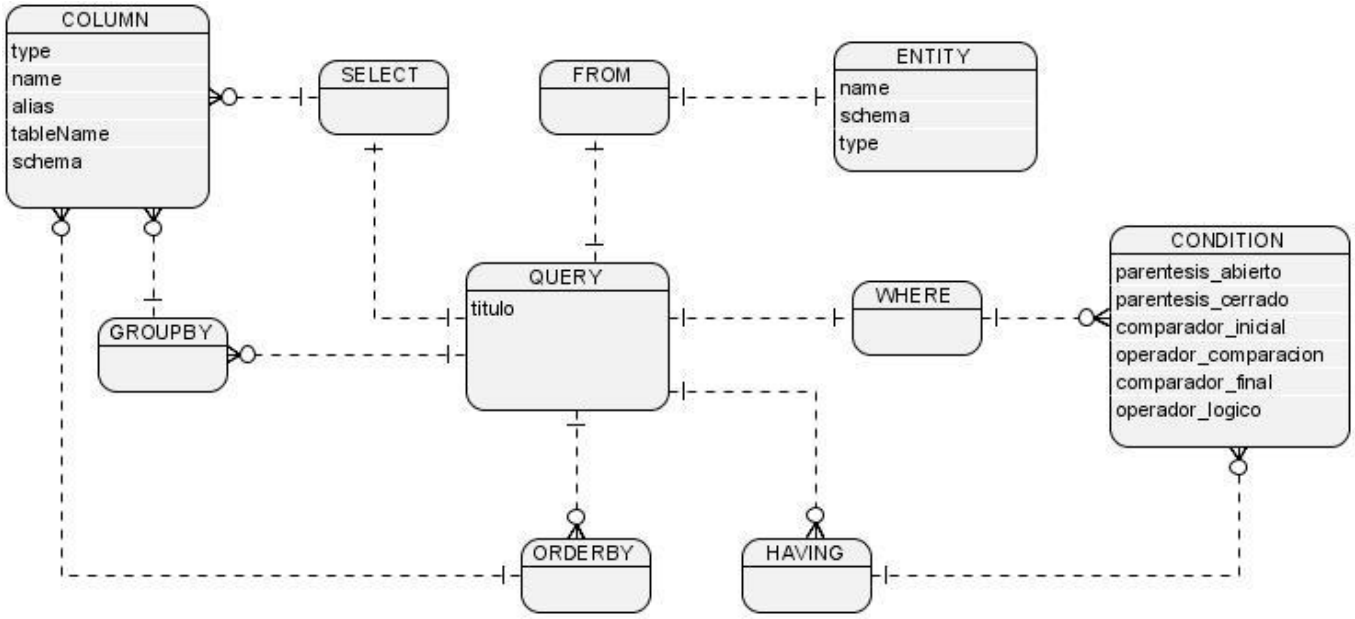


Figura 8 Estructura del XML de la consulta

2.1.3 Política de manejo de datos

Resulta esencial para las aplicaciones el garantizar la integridad y confidencialidad de la información que se maneja, es por ello que constituye un aspecto crítico a tener en cuenta en el desarrollo de software especialmente web. MD5 es un algoritmo de reducción criptográfico extensamente utilizado en el mundo del software para cifrar las claves de los usuarios. En el disco se guarda el resultado del MD5 de la clave que se introduce al dar de alta un usuario, y cuando éste quiere entrar en el sistema se compara la entrada con la que hay guardada en el disco duro, si coinciden, es la misma clave y el usuario será autenticado.

En el proyecto, para garantizar la seguridad del sistema se emplea un módulo que gestiona este proceso. Dicho módulo es reutilizado del proyecto Cedrux, que es un Sistema Integral de Gestión Empresarial, que constituye un sistema de Planificación de Recursos Empresariales (siglas en inglés, ERP). La aplicación reutilizada recibe el nombre de sistema de Gestión de Seguridad (AAA del ERP).

Para prevenir posibles vulnerabilidades en el servidor se procede a encriptar las contraseñas en el ordenador del cliente usando JavaScript MD5, en lugar de hacerlo en el servidor usando PHP.

2.1.4 Distribución física de la entidad a automatizar

Dada las características del sistema de poder ser usado en diferentes entornos ya sea empresariales o de desarrollo, la distribución física de las entidades que lo utilicen varía notablemente, se ha tomado como caso de estudio la composición del MENPET, por ser el cliente establecido en el contrato del proyecto, pero puede ser fácilmente extensible a otras entidades.

Los actores que integran el Ministerio y que influyen en el desarrollo del proyecto son los siguientes:

Oficina de Tecnología e Informática, sus Divisiones y Departamentos.

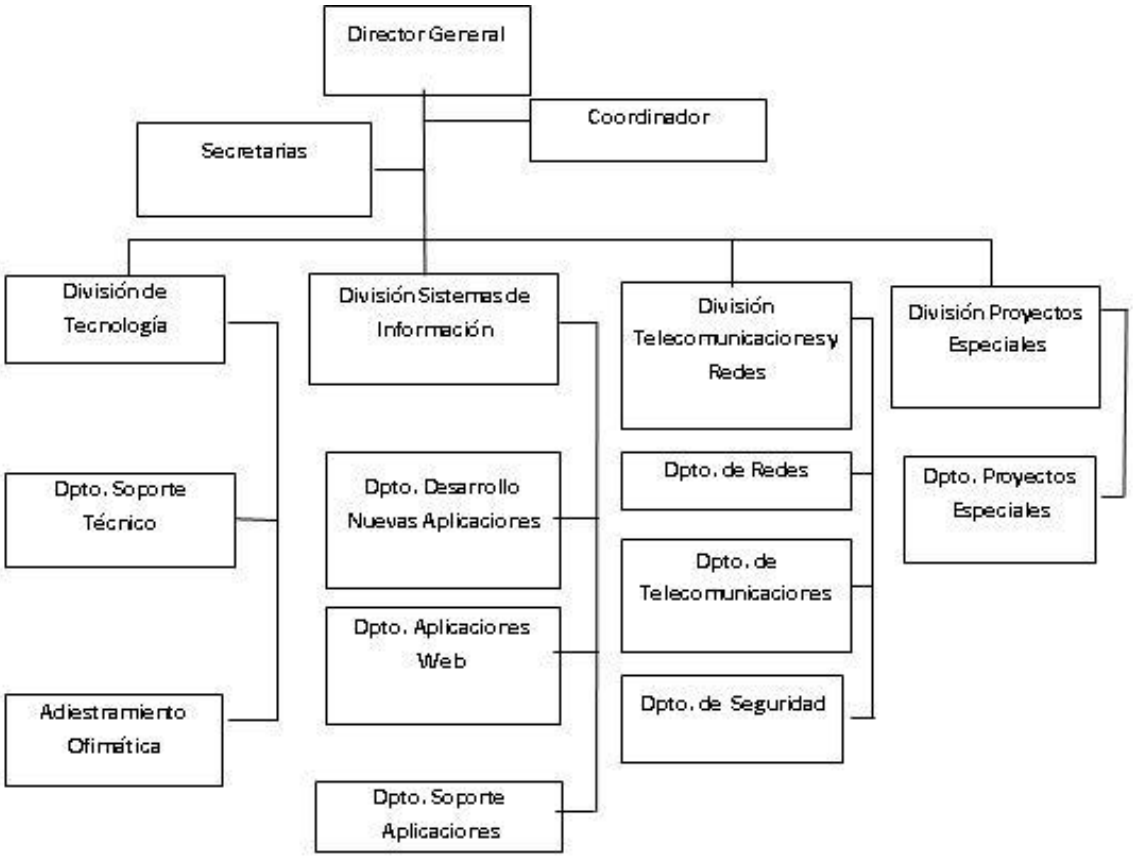


Figura 9 Estructura física MENPET

2.1.5 Requerimientos

Como parte de las restricciones de diseño más importantes se encuentra la que sostiene que el sistema debe ser desarrollado bajo Software Libre según decreto No 3.390 de fecha 23/12/2004, gaceta No 38.095 de fecha 28/12/2004 del gobierno de Venezuela, en el cual establece que la Administración Pública Nacional empleará prioritariamente Software Libre desarrollado bajo Estándares Abiertos, en sus Sistemas, Proyectos y Servicios Informáticos. El documento de Especificación de requisitos del proyecto recoge los requisitos funcionales que debe cumplir, a continuación se describen los requerimientos no funcionales fundamentales que le dan forma a la arquitectura propuesta.

Usabilidad

5. 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%.
6. El sistema será puesto en explotación en el MENPET, por lo que se necesita utilizar en el diseño del mismo un lenguaje que resulte familiar al personal que interactuará con él.

Disponibilidad del servicio

7. 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.
8. El sistema permanecerá fuera de servicio sólo en caso que se estén realizando tareas de mantenimiento o de configuración.

Eficiencia

9. El sistema debe ser rápido y ofrecer tiempos de respuesta relativamente bajos.
10. El sistema debe permitir la concurrencia.

Requisitos de Software

11. El sistema debe ser desarrollado sobre software libre.
12. El sistema debe correr y brindar el servicio desde un servidor con Sistema Operativo libre. (Debian GNU/Linux, rama Stable).
13. El sistema debe ser implementado en lenguaje de programación PHP, versión 5.

14. El sistema debe utilizar PostgreSQL versión 8.2 o superior como gestor de bases de datos.
15. El sistema debe brindar servicio a través del servidor web Apache2.
16. Todas las propiedades, los objetos y los metadatos del Sistema de Reportes deben mantenerse almacenados en una base de datos de PostgreSQL.
17. El Sistema de Reportes debe utilizar diferentes extensiones de procesamiento de datos para interactuar con distintos tipos de orígenes de datos.
18. 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.

Requisitos de Hardware

19. 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.
20. 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.
21. 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.

2.2 Tamaño y Rendimiento

El rendimiento de una arquitectura para una aplicación de tipo cliente-servidor que utiliza una base de datos como método de persistencia de la información, depende en gran medida del SGBD utilizado y su configuración.

Gracias a las sofisticadas características de PostgreSQL como el Control de Concurrencia Multi-Versión (MVCC), la creación de puntos de recuperación, los espacios de tablas, replicación asincrónica, transacciones embebidas, resguardos en línea y en caliente, 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.

2.2.1 Crecimiento de los datos

Teniendo en cuenta los diferentes escenarios de despliegue del sistema se estimó la cantidad de registros que deberá estar asociado a cada tabla del sistema de Gestión de Reportes Dinámicos, incluyendo un estimado de crecimiento de los datos de alrededor de un 15%, todo esto referente a un año de explotación. Por cada tabla se determinó el espacio que ella ocupa según sus atributos, cantidad de registros asociados, así como el espacio que se necesita para manejar sus índices asociados (ver Anexo 5).

Para el cálculo de estos valores se utilizó del Erwin 7.5 la herramienta Capacity Planning. En total se estima un tamaño de la base de datos de 160.7M, factor este que va a estar determinado por la explotación que se realice del sistema por parte de la empresa u organización donde se encuentre desplegada.

2.2.2 Tiempo de respuesta

Una técnica muy útil para determinar la concurrencia que puede soportar el servidor consiste en monitorear y determinar la cantidad de memoria promedio por cada hilo de apache, dividiendo la RAM

instalada entre ese número y dejando una cantidad razonable para otras funciones del Sistema Operativo se obtiene la cantidad de conexiones máximas a configurar. En sistemas basados en GNU/Linux con versiones del kernel superiores a la 2.4 se incluye una instrucción del sistema llamada “sendfile” la que facilita a Apache 2 entregar contenido estático mucho más rápido y con menor uso del CPU.

Los tiempos de respuesta están determinados en gran medida por el tamaño y la complejidad del reporte que se desee generar, se estima que para un reporte de más de 70 000 tuplas la respuesta debe obtenerse en menos de 3 minutos.

2.2.3 Niveles concurrencia

Los niveles de concurrencia para la lectura serán los más críticos para el sistema, se estiman 50 conexiones. El MVCC se encarga de gestionar esta situación en el gestor de base de datos, en el caso del servidor web Apache soporta una gran carga de concurrencia, la clave se encuentra en configurar el servidor para no permitir más conexiones de las que este es capaz de atender sin necesidad de hacer paginado de memoria ya que esta memoria adicional es en disco duro y no en RAM, afectando considerablemente los tiempos de respuesta.

2.3 Vistas de la arquitectura

La arquitectura, al ser el centro del desarrollo de software, debe ser descrita de forma que tanto desarrolladores como involucrados sean capaces de comprender la parte del sistema que les concierne, debe constituir un mapa donde se estructure desde diferentes perspectivas el software a construir. RUP propone que sea descrita a través de vistas arquitectónicas (4+1 vistas) que permiten especificar diferentes partes del sistema mediante diagramas y especificaciones.

2.3.1 Vista de casos de uso

El sistema de Gestión de Reportes Dinámicos tiene como principal objetivo la generación de reportes, proceso que se garantiza mediante la interrelación entre diferentes aplicaciones. 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.

2.3.1.1 Diseñador de modelos

El Diseñador de modelos recoge todos los casos de uso que se utilizan para definir, editar y publicar modelos semánticos. Un modelo semántico es una descripción de la estructura de la base de datos del negocio. El modelo semántico brinda una mayor seguridad al sistema ya que el diseño de los reportes se realiza a partir de estos y no directamente con la base de datos, además evita sobrecargar la base de datos con peticiones continuamente. Los modelos semánticos son guardados en ficheros XML y posteriormente almacenados en el Servidor de Reportes donde podrán ser consultados.

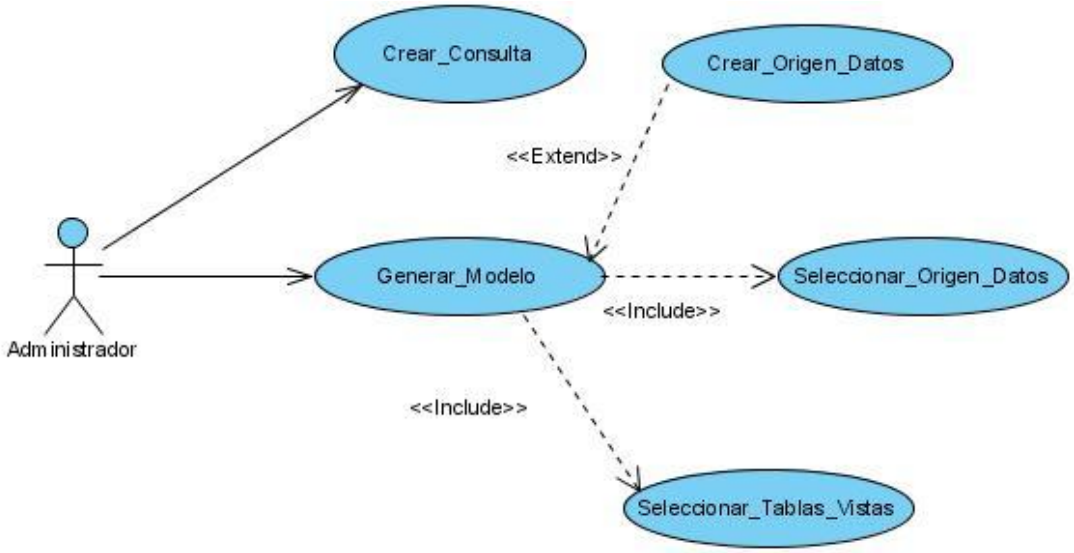


Figura 10 DCU Diseñador de modelos

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

Crear_Origen_Datos: Posibilita la creación del origen de datos al cual se conectará el usuario para la creación del modelo semántico, se especifica el tipo de gestor, el servidor, usuario y nombre de la base de datos. Es posible la búsqueda de bases de datos disponibles, así como probar la conexión.

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

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

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

2.3.1.2 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.

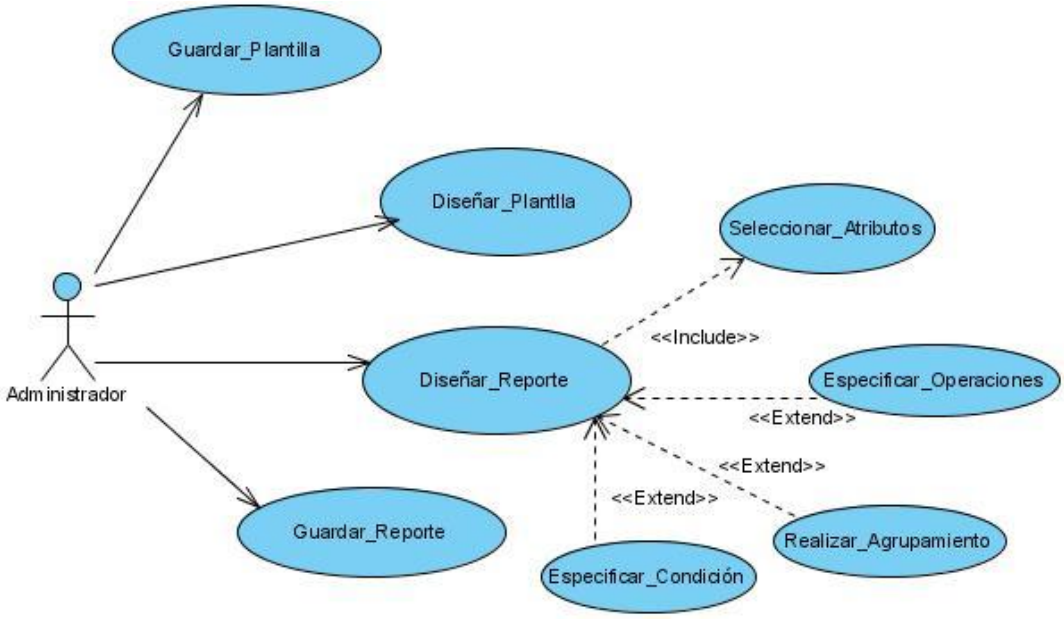


Figura 11 DCU significativos del Diseñador de reportes

Diseñar_Reporte: El caso de uso permite diseñar un reporte, ya sea desde una plantilla existente o sin plantilla. 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_Atributos: Permite seleccionar los campos que conformarán el reporte. Para crear la fuente de datos se debe seleccionar el modelo, la entidad dentro del modelo y los campos que se requieren sean incorporados en el diseño del reporte.

Especificar_Operaciones: Realizar operaciones sobre las columnas de los grupos o sobre las columnas de todo el reporte. Permite la sumarización de los datos del reporte a diferentes niveles de agrupamiento, lo que le da valor agregado puesto brinda información adicional para el que requiere la información.

Realizar_Agrupamiento: Posibilita el agrupamiento de los datos por uno o varios campos del reporte, se pueden especificar diferentes niveles de agrupamiento, es una forma para el análisis de los datos del reporte.

Especificar_Condición: Posibilita filtrar los datos, especificando condiciones sobre los campos del reporte.

Guardar_Reporte: Procesa el diseño del reporte y genera la definición del mismo, se salva para su posterior uso.

Diseñar_Plantilla: Permite la creación de plantillas a partir de las cuales se pueden crear nuevos reportes. Estas plantillas no están relacionadas con ninguna fuente de datos, solo se define el formato y estilo de los elementos a mostrar en el informe.

Guardar_Plantilla: Procesa el diseño de la plantilla y genera la definición de la misma, se salva para su posterior uso.

2.3.1.3 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.

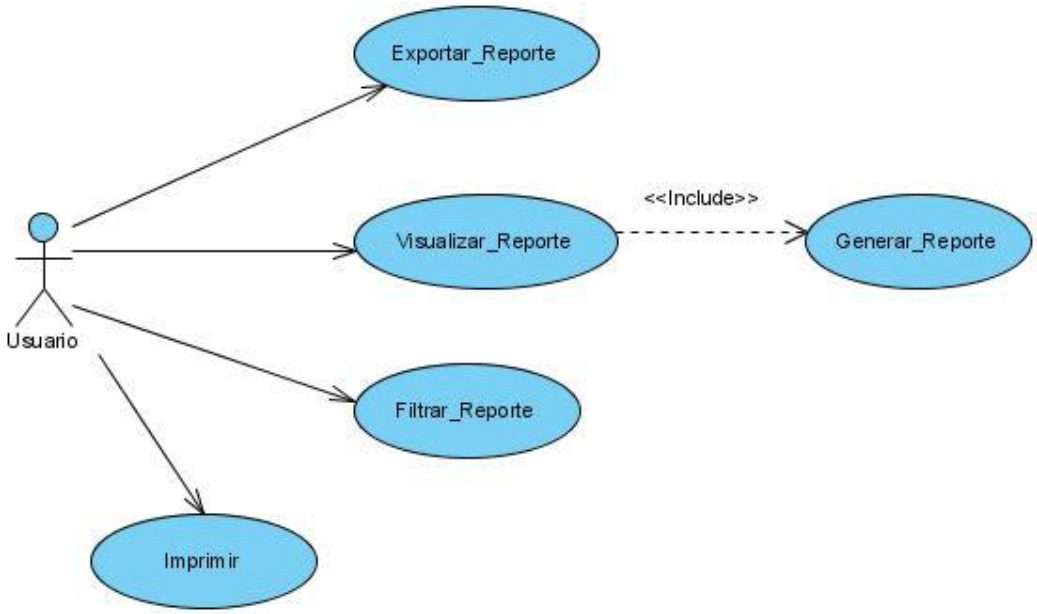


Figura 12 DCU significativos Visor de Reportes

Visualizar_Reporte: Luego de la selección de un reporte previamente creado, posibilita la visualización de una vista previa del mismo con un máximo de diez filas. Para ello es necesaria la ejecución del caso de uso Generar_Reporte.

Generar_Reporte: El sistema se conecta a la fuente de datos recupera la definición del reporte y lo transforma al formato de salida por defecto (HTML).

Exportar_Reporte: Posibilita la exportación del reporte a diferentes formatos de salida: HTML, PDF, CSV, TXT, entre otros.

Imprimir: Permite imprimir un reporte.

2.3.1.4 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 (ver Figura 13).

Gestionar_Modelo: Permite eliminar, modificar y renombrar un modelo previamente creado.

Gestionar_Consulta: Permite adicionar y eliminar una consulta previamente creada en un modelo.

Gestionar_Categoría: Posibilita la adición, eliminación y modificación de categorías que es una forma para organizar los reportes en el sistema.

Gestionar_Reporte: Permite copiar o mover un reporte previamente creado hacia la categoría deseada. Se puede también eliminar un reporte existente.

Gestionar_Suscripción: El objetivo es la creación o eliminación de una suscripción a un reporte creada anteriormente.

Programar_Suscripciones: Permite especificar atributos a las suscripciones como un identificador, fecha de inicio y culminación de la tarea programada.

Programar_Recurrencia: Posibilita configurar la programación de la suscripción baja determinadas frecuencias, ya sea diariamente, semanalmente, a una hora específica o solo una vez.

Configurar_Seguridad: Permite establecer permisos a los roles y a los usuarios del sistema creados a determinado reporte.

Configurar_Archivo: El caso de uso permite configurar el envío de una suscripción a un servidor de ficheros. Se especifica la dirección de donde se encuentra el servidor, el formato del reporte, las credenciales para acceder y opciones de escritura.

Configurar_Correo: Posibilita configurar el envío de la suscripción por correo electrónico, estableciendo el destinatario, el asunto del correo, las copias y el cuerpo del mismo.

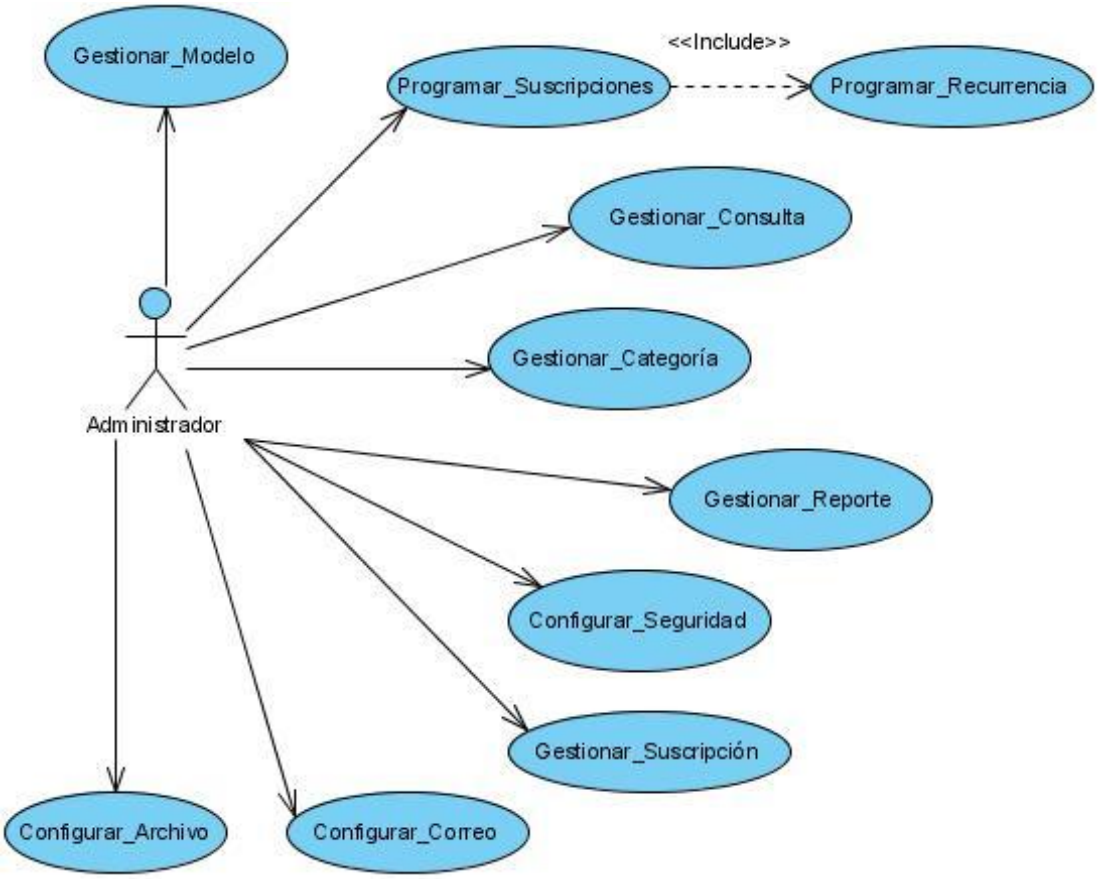


Figura 13 DCU significativos Administrador de reportes

2.3.2 Vista Lógica

Cuando se crea una aplicación es importante iniciar con una arquitectura lógica que clarifique los roles de todos los componentes, separe funcionalidades de forma tal que los miembros del equipo de trabajo trabajen juntos efectivamente. 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.

2.3.2.1 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.

La arquitectura del sistema está organizada en 6 capas, que constituye una variante del conocido estilo 3 capas. Hoy en día, no es usual encontrarse tan solo 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) en el caso de las aplicaciones web. La Figura 14 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. En la solución la Vista brindada por Symfony se utiliza como una interfaz para la comunicación e intercambio de información con la capa de Presentación, 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 nuestro 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.

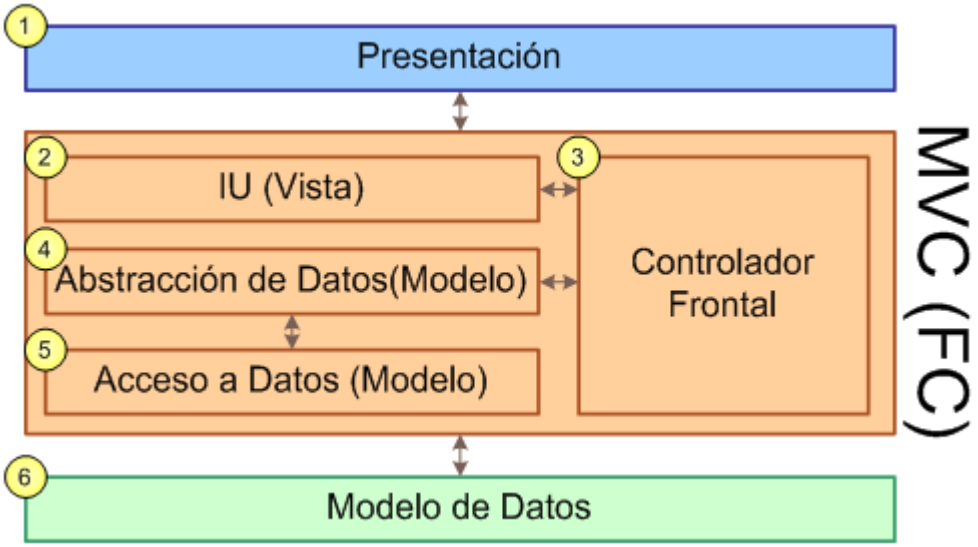


Figura 14 Estructura en capas del sistema

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 2.2 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 14, 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. En la solución se implementa con la aplicación report_generator.
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 Creole 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

Creole utiliza Pear::DB como paquete de abstracción, constituye una interfaz para la comunicación con las base de datos completamente orientada a objetos. Se utiliza TCP/IP como protocolo de comunicación.

2.3.2.2 Frameworks y librerías de desarrollo

En cada una de las capas mencionadas anteriormente se emplean frameworks y librerías que contienen un conjunto de componentes implementados e interfaces bien definidas que aceleran el desarrollo del sistema y proveen a la aplicación una estructura organizada del código.

La distribución de los componentes a utilizar en cada capa y las librerías son:

- Capa de Presentación: Framework ExtJS y OpenJacob.
- Capa de Negocio: Symfony
  - Librerías: PHPReports, TCPDF, pChart.
- Capa de Abstracción y Acceso a Datos: Propel.
  - Librería: Creole.

ExtJS

ExtJS es un Framework para el desarrollo de aplicaciones web con una experiencia de usuario mejorada, introduciendo el concepto de Rich Internet Application (RIA).

Sus principales características son:

- Alto rendimiento en ejecución debido a la optimización de código JavaScript.
- Controles de usuario personalizables.
- Modelo orientado a componentes, bien diseñado y extensible.
- Posee una API intuitivo y fácil de utilizar.
- Distribuido bajo licencias OpenSource y comerciales.

A través de la inclusión de los ficheros Ext-all.js y Ext-base.js se tiene acceso a todas las clases y paquetes que componen el framework, destacando entre los más utilizados la clase Ext, la cual implementa el patrón Singleton para asegurar el acceso a una única instancia en todo momento debido a su importancia, pues esta clase es la que provee los métodos encode y decode, ampliamente utilizados para la codificación y decodificación de información en formato JSON para el intercambio con el servidor de forma rápida y segura (ExtJS, 2009).

Otros paquetes y clases importantes se mencionan a continuación:

Paquete data: Contiene las clases JsonReader y JsonStore que como su nombre indican intervienen en la lectura y almacenamiento de información en JSON.

Paquete form: En este paquete se encuentran los componentes utilizados para la creación de formularios así como las clases que se encargan de administrar el envío de información a través de éstos.

Paquete grid: Éste contiene componentes de tipo cuadrícula en sus distintas variantes para la visualización y entrada de información de forma tabular o matricial así como los elementos que se utilizan para la construcción de dichos componentes.

Paquete layout: Aquí se encapsula un grupo de clases que permiten configurar y definir aspectos estéticos de las interfaces.

Paquete menú: Contiene los componentes que forman los distintos tipos de menús y clases como MenuMgr, la cual implementa el patrón Singleton y provee un registro de todos los elementos de tipo menú en una página haciéndolos fácilmente accesibles mediante su identificador.

Paquete tree: Este paquete contiene los componentes de tipo árbol y los elementos con los que se pueden conformar éstos.

En un nivel superior de jerarquía existen otras clases y componentes relevantes como es el caso de la clase Ajax, encargada de este tipo de peticiones asíncronas al servidor (ver Figura 15).

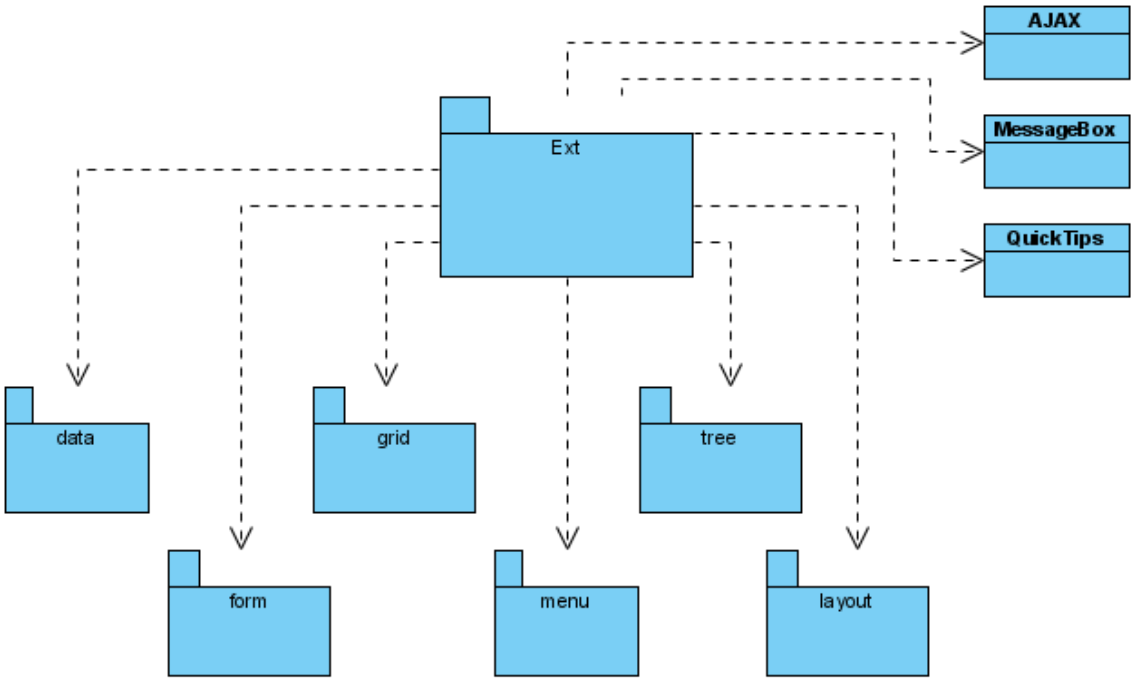


Figura 15 Diagrama de paquetes de ExtJS

OpenJacob

Es un framework libre para la construcción de aplicaciones web de nueva generación. Draw2D es el componente gráfico del editor de flujos de trabajo basado en la web de OpenJacob. En otras palabras Draw2D se puede ver como un paquete de métodos y clases embebidas que facilitan notablemente la creación de editores de formularios y componentes al estilo "arrastrar y soltar" directamente en la web y sin necesidad de complementos adicionales para los navegadores. Algunas de las clases que incluye son: draw2d.Rectangle, draw2d.Graphics, draw2d.Figure, draw2d.Circle, draw2d.Line, draw2d.Connection, como sus nombres indican, instanciando estas clases se obtienen objetos que representan figuras que se mostrarán y podrán personalizar. Por otra parte existen métodos generales para la manipulación de esos objetos como es el caso del método Drag para los efectos arrastrar y soltar (Herz, 2009).

PHPReports

PHPReports es un motor de generación de reportes para PHP, es software libre y provee un alto nivel de productividad en este tipo de aplicaciones. Un reporte en PHPReports está dividido en capas que contienen unas a otras, la Figura 16 muestra esta estructura.

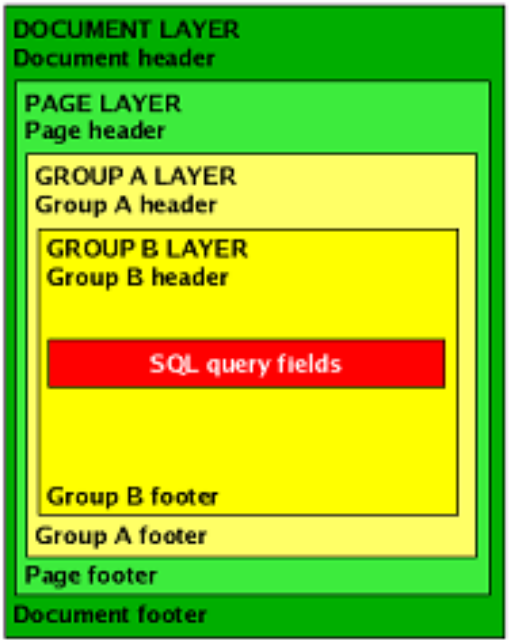


Figura 16 Estructura de un reporte

Como se puede observar, un reporte sería un documento, contiene un encabezado, un pie y está formado por páginas, las que a su vez cuentan con un encabezado, un pie de página y puede contener grupos anidados en los que se cargarán los campos obtenidos de la consulta a la base de datos.

El funcionamiento de PHPReports se basa en transformaciones XSL (*siglas de Extensible Stylesheet Language*) conocidas como XSLT a partir de la clase PHPReportMaker. Una instancia de esta clase es la que recibe los parámetros básicos para la creación de un reporte como son: una conexión válida a una fuente de datos y un documento XML que contiene la definición del reporte que será transformada mediante XSLT a código HTML u otros formatos de salida como PDF, XLS, CSV entre otros. Luego de especificarle a la instancia de la clase PHPReportMaker los parámetros mencionados anteriormente el método run desencadena la conexión con el origen de los datos, una vez obtenido un XML de datos se combina con información de diseño y se obtiene el reporte (Rangel, 2006).

TCPDF

TCPDF es una clase para la generación de documentos en formato PDF desde PHP 4 o PHP 5, liberada bajo una licencia de software libre denominada FLOSS (*Free Libre Open Source Software*). Esta clase es una extensión de otra existente para este propósito conocida como FPDF pero incluye la posibilidad de escribir contenido en posiciones específicas del documento y no sólo de forma continua como su antecesora. La creación de un documento comienza con la creación de un objeto de la clase TCPDF indicando algunos parámetros como la orientación de la página, las unidades de medida y la codificación de caracteres entre otros. Luego a ese objeto se le pueden cambiar propiedades como el tipo de fuente, márgenes y saltos de páginas a través de los métodos SetFont, SetMargins y SetAutoPageBreak respectivamente. El próximo paso hacia la creación del documento es adicionar una página mediante el método AddPage y comenzar a agregar contenido a cada página haciendo llamadas al método MultiCell, el cual permite escribir celdas con la información (Tecnick.com, 2009).

Algunas de sus características más relevantes son:

- No se requiere ninguna librería externa para las funcionalidades básicas.
- Soporta todos los formatos de páginas ISO.
- Permite la creación de documentos encriptados.
- Soporta de forma nativa la inclusión de imágenes en los formatos JPG, PNG, GIF, JPEG, BMP y XPM.

Pchart

pChart es un framework orientado a objetos para PHP diseñado para crear gráficos. La información a representar puede ser obtenida directamente de consultas SQL, ficheros CSV o suministrada manualmente. La siguiente figura describe el proceso de generación de gráficos con pChart.

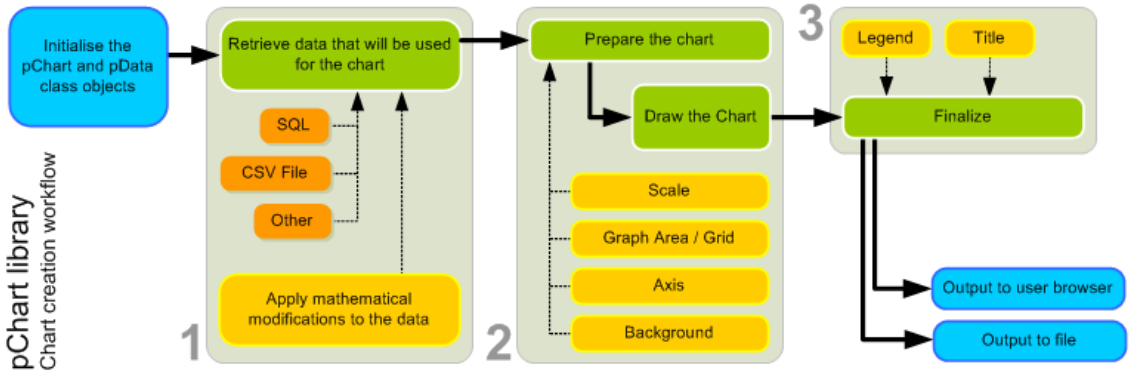


Figura 17 Proceso de creación de gráficos

Crear un gráfico se divide en tres partes: Primero se accede a los datos y se preparan para ser graficados, para esto se puede utilizar la clase pData para ayudar en este proceso. Segundo, preparar el diseño del gráfico definiendo como serán los bordes, el fondo y el área de dibujo entre otras propiedades creando finalmente una imagen. Por último se colocan etiquetas, títulos y leyendas para almacenar el gráfico en un servidor o enviarlo directamente al navegador.

Propel

Propel es un framework para el mapeo objeto-relacional escrito en PHP 5. Este permite el acceso a las bases de datos a través de un grupo de objetos proporcionando una API sencilla para almacenar y recuperar información. Con el uso de Propel los desarrolladores de sistemas web trabajan con las bases de datos de igual forma que con otras clases y objetos de PHP 5. Este framework brinda herramientas por línea de comandos para la generación automática de la capa de acceso a datos a partir de una base de datos e incluso es posible generar la base de datos a partir de una descripción en formato XML o YML del modelo de datos. Debido a que se utiliza Creole como capa de abstracción de bases de datos, Propel es capaz de interactuar con varios sistemas gestores de bases de datos. Para utilizar dichas herramientas se necesita configurar el fichero propel.ini, el cual contiene la configuración que cargarán las herramientas de Propel. Luego de hacer las configuraciones donde se debe especificar el tipo de gestor de bases de datos,

la dirección del servidor, un usuario y una contraseña válida así como la base de datos a mapear, el framework puede ser utilizado (Propel project, 2009).

Creole

Creole es una capa de abstracción de bases de datos para PHP 5 que abstrae la API específica nativa de PHP para cada gestor creando un código mucho más portable y brindando a los desarrolladores una interfaz completamente orientada a objetos. Fue creado originalmente como un subproyecto de Propel para resolver el problema de que ninguna de las capas de abstracción disponibles satisfacía las necesidades de éste. Actualmente ha dejado de desarrollarse motivo del surgimiento de PDO (PHP Data Objects) como nueva capa de abstracción de bases de datos para PHP 5 (Creole project, 2009).

Los estilos y frameworks antes descritos se evidencian en cada uno de los módulos del sistema, que contienen una estructura determinada por la combinación de los diferentes estilos y representan los componentes de diseño principales para la realización de los casos de uso.

2.3.2.3 Elementos del modelo arquitectónicamente significantes

La Figura 18 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.

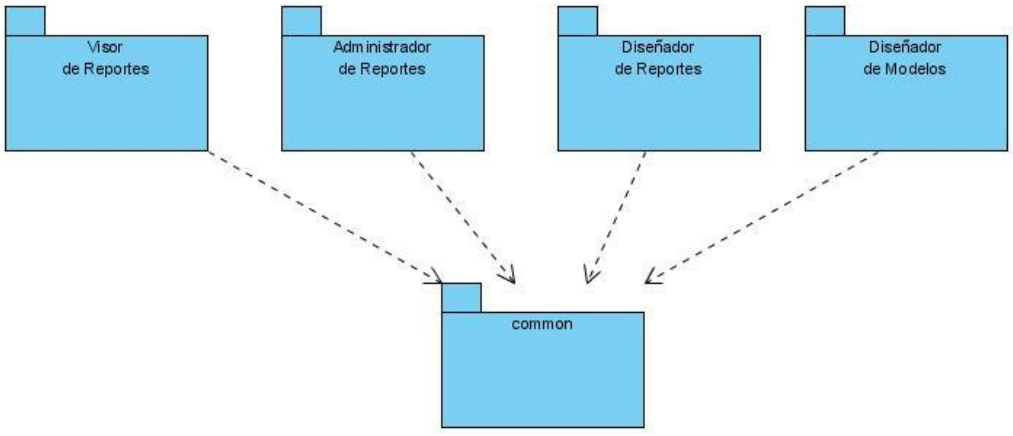


Figura 18 Diagrama de módulos del sistema

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 19 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.

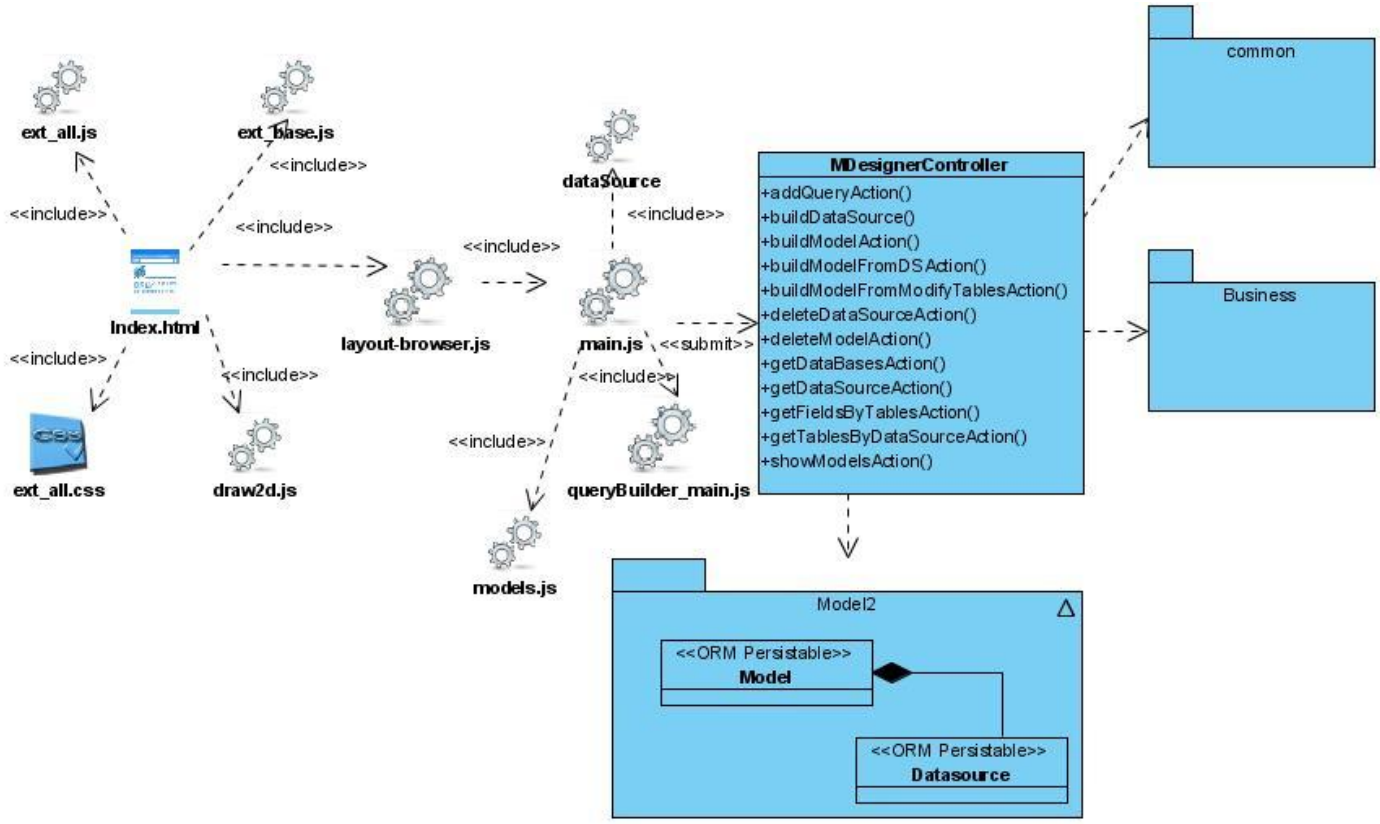


Figura 19 Diagrama de clases Diseñador de modelos

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 20 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.

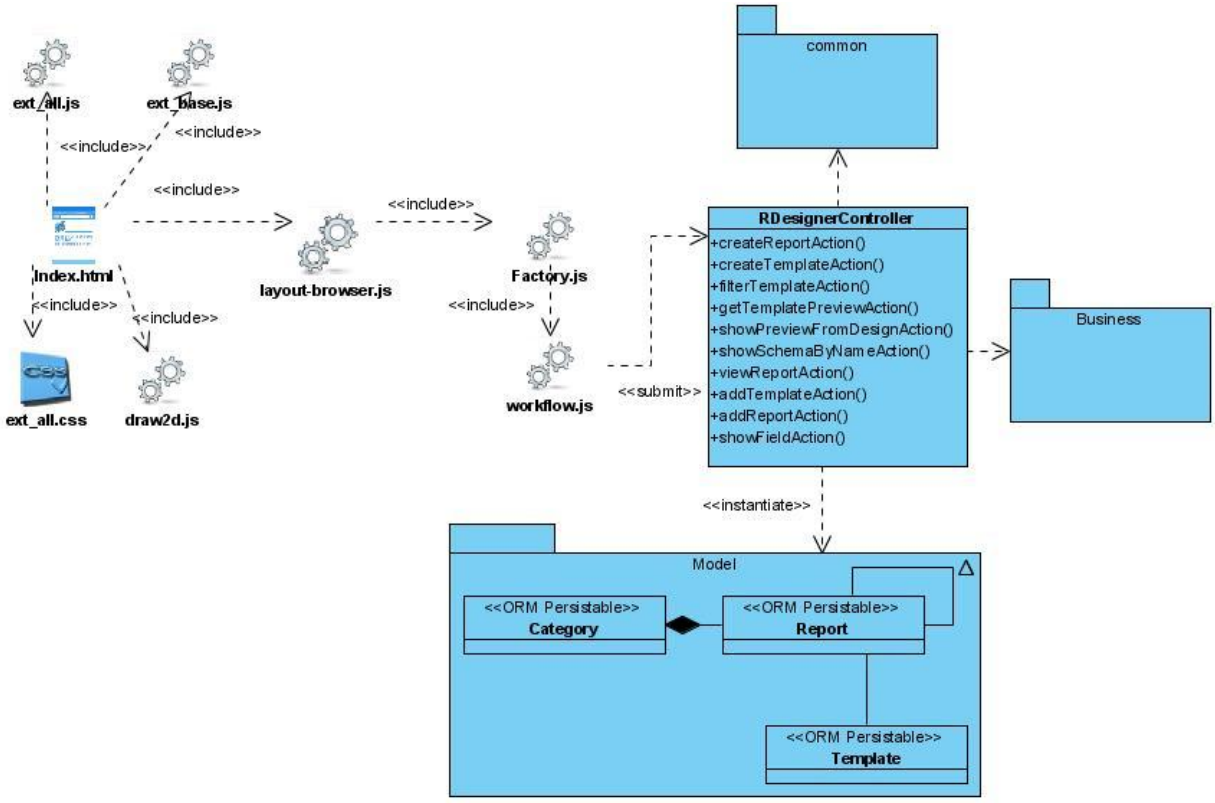


Figura 20 Diagrama de clases Diseñador de reportes

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 21 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.

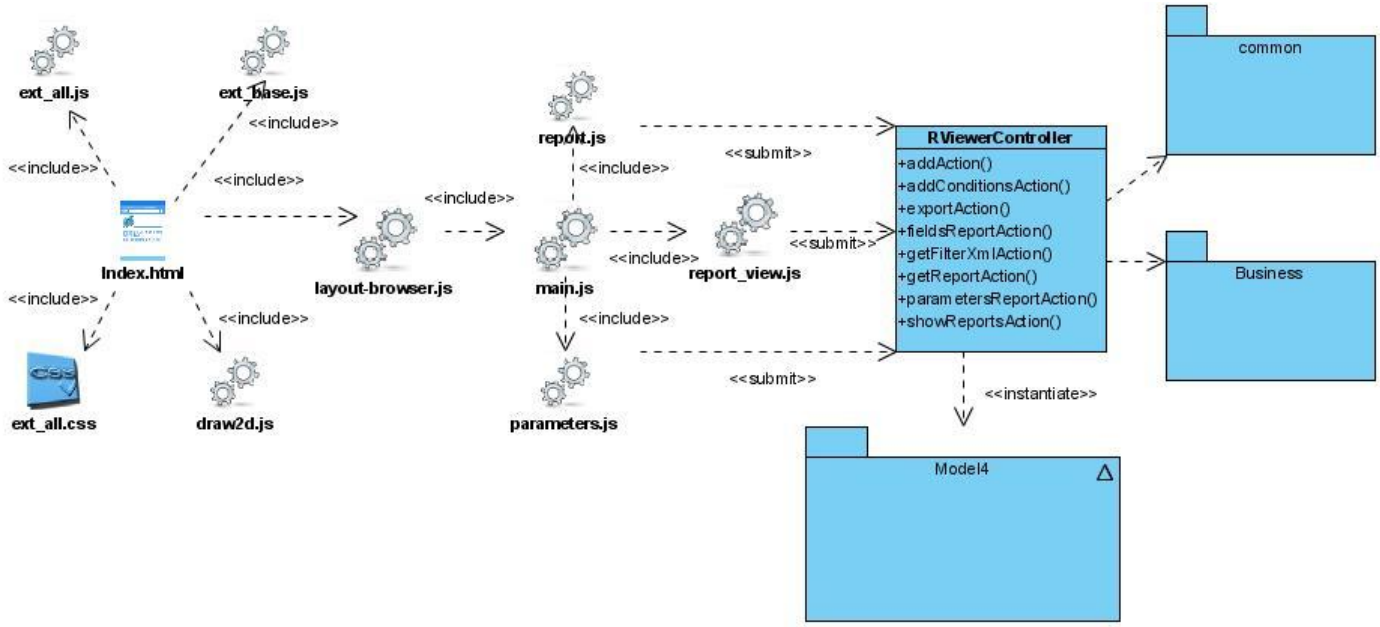


Figura 21 Diagrama de clases Visor de reportes

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.

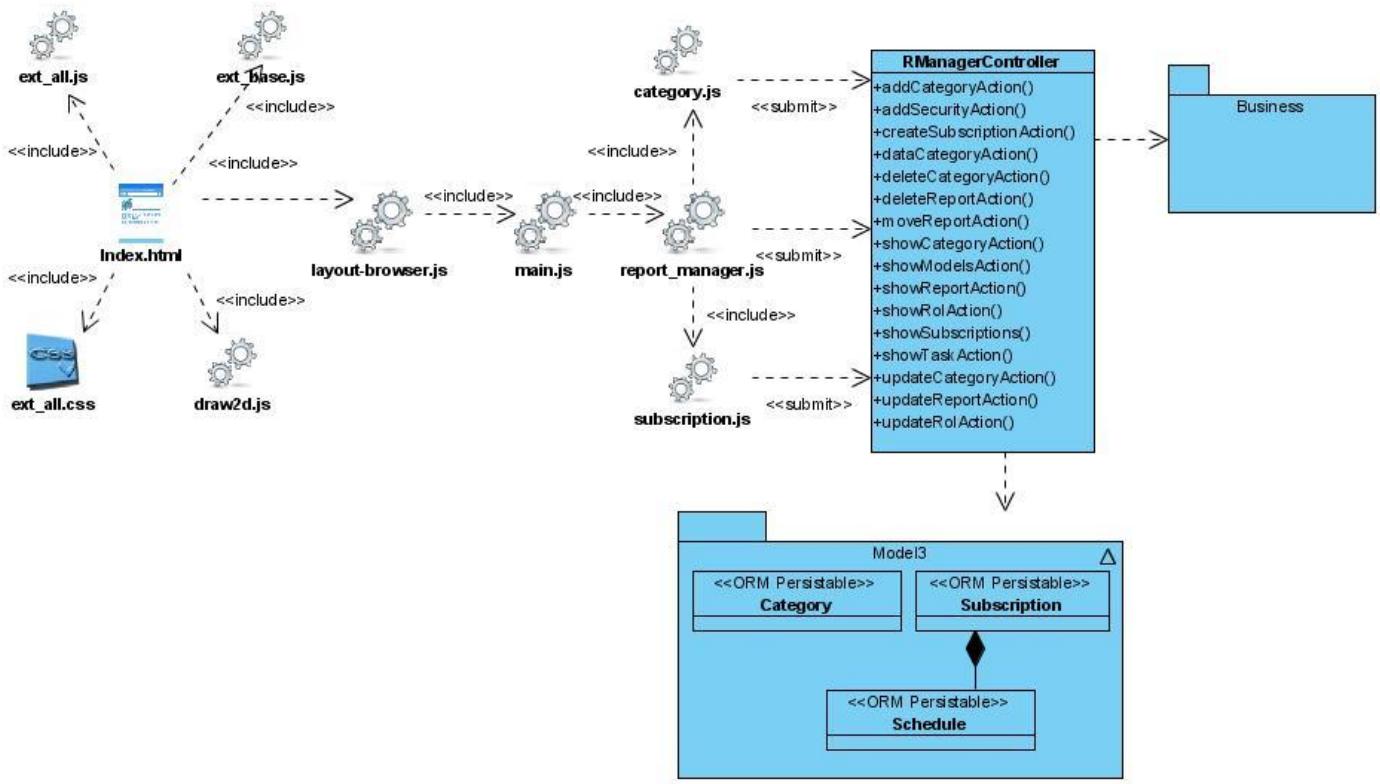


Figura 22 Diagrama de clases Administrador de reportes

2.3.3 Vista de procesos

Esta vista describe los procesos e hilos activos del sistema de Gestión de Reportes Dinámicos, éstos estarán en ejecución de forma simultánea. Además se describe el soporte multiusuario de la aplicación.

2.3.3.1 Procesos distribuidos

Al ser un sistema basado en la web se encuentra distribuido de forma que a nivel de usuario final corre el navegador Mozilla Firefox, encargado de presentar la interfaz de la aplicación y de enviar al servidor las acciones que el usuario realiza. Por el lado del servidor se encuentra otra aplicación (servidor web), en este caso Apache2 con el módulo para PHP5, éste recibe las solicitudes de los usuarios utilizando el protocolo HTTP y las redirige a las páginas servidoras ubicadas en la capa de Interfaz de usuario correspondientes a la Vista de la arquitectura MVC. Es el servidor web el encargado de gestionar la

atención a múltiples clientes en forma simultánea, para lo cual Apache2 abre un nuevo hilo de ejecución de PHP por cada conexión.

Por otra parte se ejecuta de forma solapada otro hilo de PHP como "daemon" (Disk and Status Monitor) o demonio del sistema operativo desde que este inicia. Este demonio es el encargado de mantener actualizada la cola de tareas programadas según las suscripciones que se encuentran activas y almacenadas en la base de datos. Para dar respuesta a estas suscripciones el demonio va ejecutando las tareas que tiene en cola para generar y entregar los informes según lo programado y además monitorea las suscripciones para determinar en qué momento debe entrar una tarea en cola y cuando salir.

2.3.3.2 Distribución de las capas

Los módulos presentes en las diferentes capas que componen al sistema no se encuentran distribuidos, las razones de no utilizar distribución a este nivel son por rendimiento de las operaciones marcadas como críticas y para disminuir el riesgo del proyecto en función de las tecnologías.

Los servicios consumidos son de dos tipos, aplicaciones que deben ser utilizadas y el propio motor de base de datos. El motor de base de datos corre en forma independiente, por otro lado aplicaciones utilizadas como el módulo de seguridad del ERP exportan sus servicios mediante web services. Así los módulos presentes en Common son un adaptador propio a dichos servicios.

2.3.3.3 Arquitectura de procesos

En base a las decisiones descritas en la sección anterior, se muestra a continuación la arquitectura de procesos del sistema de Gestión de Reportes Dinámicos.

La Figura 23 muestra la arquitectura de uno de los procesos que se ejecuta en el sistema asociado al servidor web.

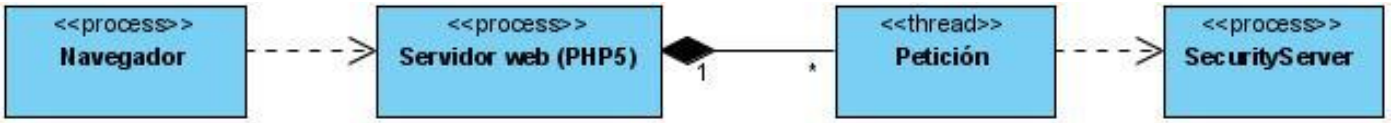


Figura 23 Vista de procesos asociados al servidor web

El servidor web destina un hilo de ejecución (Petición en la figura) a cada conexión. Cada hilo está asociado a una sesión que es preservada entre diferentes peticiones. Cada Petición ejecuta todos los

módulos en la aplicación que sean necesarios para llevar adelante la solicitud del usuario. Cuando se requiere se comunica mediante web services con SecurityServer (servicio que puede estar corriendo en otro servidor).

Se hace necesario destacar que la aplicación desarrollada no codifica en forma explícita esta arquitectura de procesos. La tecnología subyacente brinda servicios de alto nivel que lo encapsulan.

Por último, cuando el servidor crea una Petición y comienza con la ejecución del código de las páginas dinámicas, éstas asocian al hilo el contexto transaccional. El módulo de acceso a datos toma dicho contexto y lo utiliza al momento de realizar acciones contra el motor de base de datos.

La Figura 24 muestra el otro proceso que ejecuta el sistema el cual es el encargado de la programación y entrega. Al iniciarse los sistemas operativos GNU/Linux, en sus distintas distribuciones, se ejecuta el "init.d", el cual no es más que un script encargado de poner en ejecución cada uno de los servicios y demonios que corren en segundo plano, este script pondría en ejecución el proceso descrito en la figura como "daemon", este proceso creará principalmente dos hilos de ejecución sobre PHP los cuales se encargarán, uno de monitorear las suscripciones activas en la base de datos y mantener actualizada la cola de tareas y otro atenderá esa cola para ejecutar las acciones necesarias para obtener los reportes y entregarlos ya sea por correo electrónico mediante SMTP (Simple Mail Transfer Protocol) a través de un servidor Postfix o por servidores FTP (File Transfer Protocol).

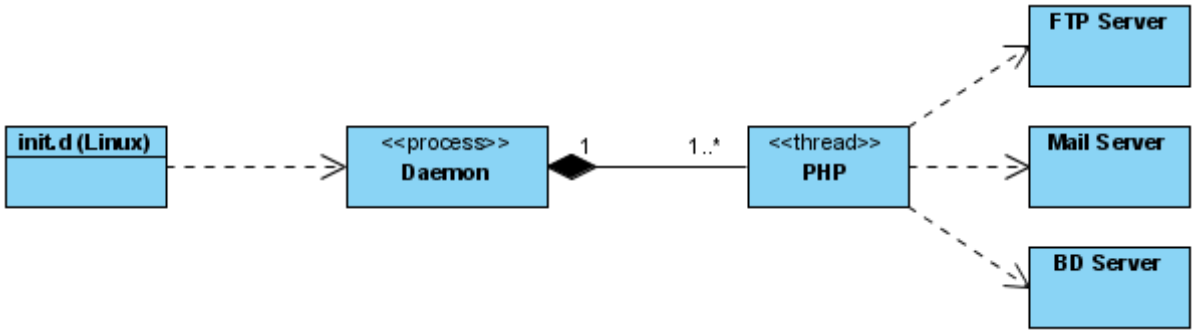


Figura 24 Vista de procesos programación y entrega

2.3.4 Vista de Despliegue

Debido a que el sistema de Gestión de Reportes Dinámicos puede manejar información delicada para determinada institución, está diseñado para que cada organización instale una instancia del sistema en sus servidores y lo utilice de acuerdo a sus necesidades teniendo en cuenta los distintos 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 (ej. uno de los escenarios de integración con el ERP cubano), como aplicación independiente pero con acceso a los reportes desde otra aplicación o sistema, entre otros.

2.3.4.1 Instalación como sistema independiente

En este caso se necesita un servidor web como entorno de ejecución con Apache2, soporte para PHP 5 y sistema operativo basado en GNU/Linux en cualquiera de sus distribuciones, preferentemente Debian o Ubuntu. La base de datos puede estar instalada en el mismo servidor o en uno distinto en dependencia de las posibilidades de la organización, siendo necesario PostgreSQL Server, versión 8.3 o superior. 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 25).

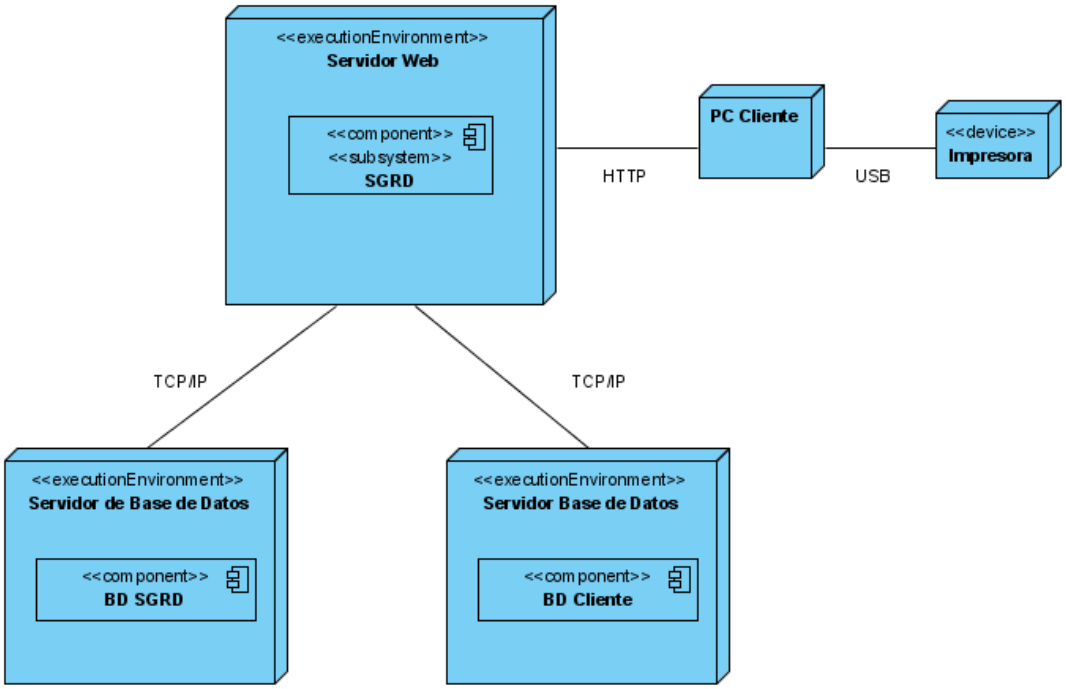


Figura 25 Vista de despliegue del sistema independiente.

2.3.4.2 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 SGRD, debido a restricciones establecidas por el funcionamiento de Symfony (ver Figura 26).

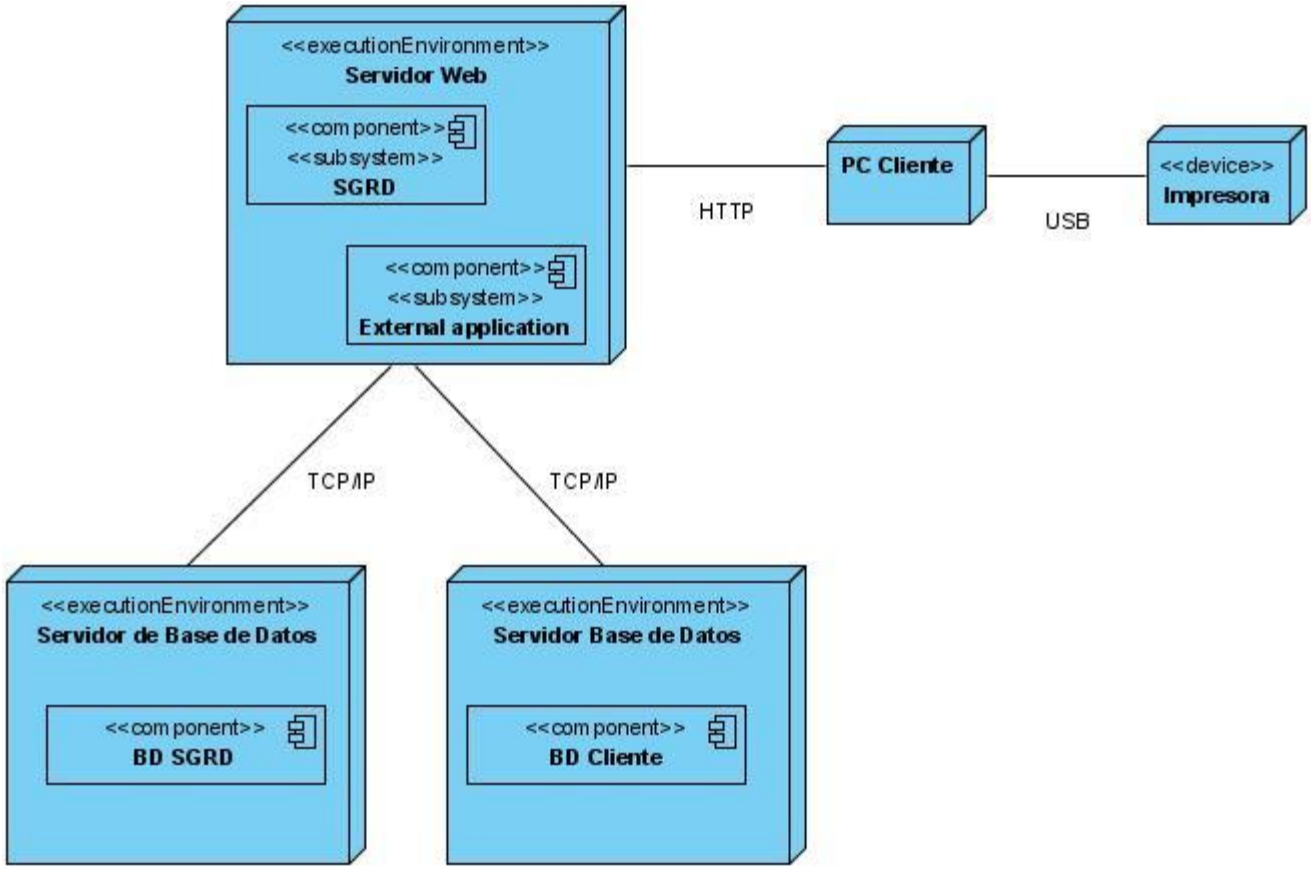


Figura 26 Vista de despliegue sistema embebido

2.3.4.3 Instalación como sistema independiente con acceso a los reportes desde otra aplicación o sistema.

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 SGRD debido a restricciones establecidas por el funcionamiento de Symfony. 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 27).

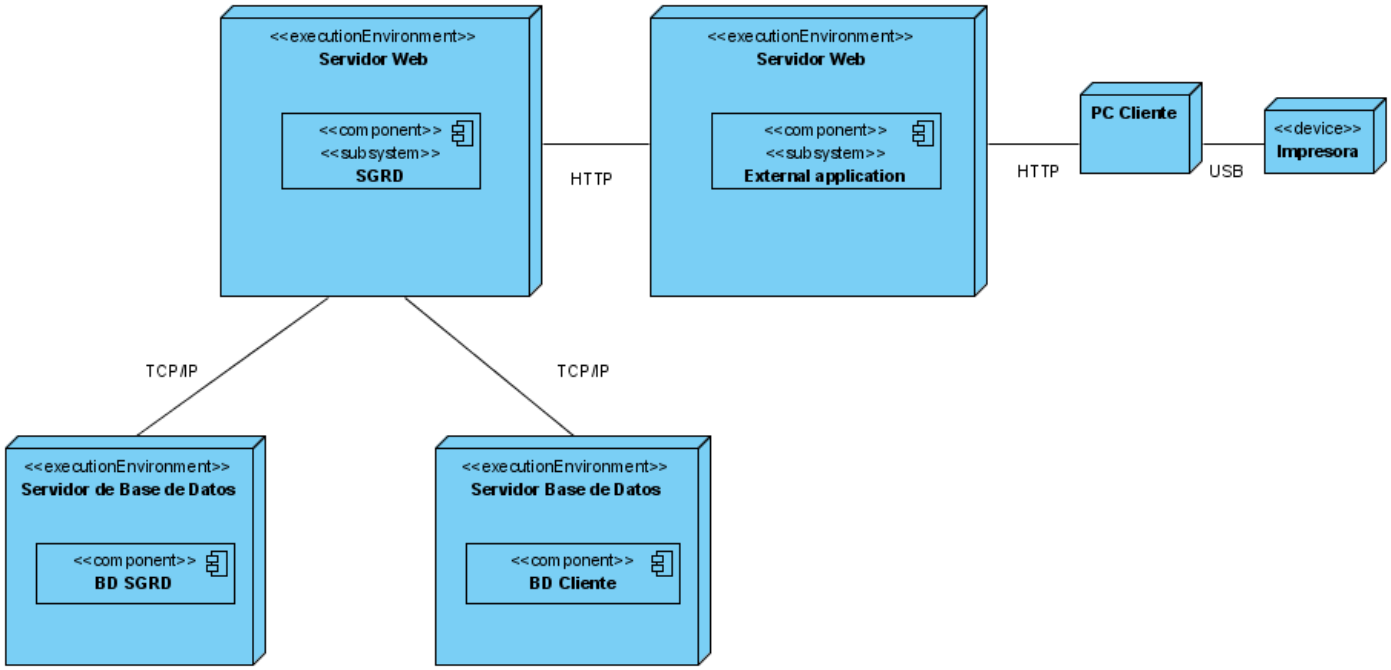


Figura 27 Vista de despliegue sistema independiente con acceso remoto a reportes

2.3.5 Vista de Implementación

Esta vista se centra en la organización real de los módulos en el ambiente de desarrollo del software. Describe la estructura general del Modelo de Implementación y el mapeo de los subsistemas, paquetes y clases de la Vista Lógica a subsistemas y componentes de implementación (Alvez, et al., 2006).

Symfony 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 los cuales almacenan las acciones, que representan cada una de las operaciones que se puede realizar (Potencier, et al.). La siguiente figura muestra la estructura física del sistema, con la distribución por módulos y acciones significativas.

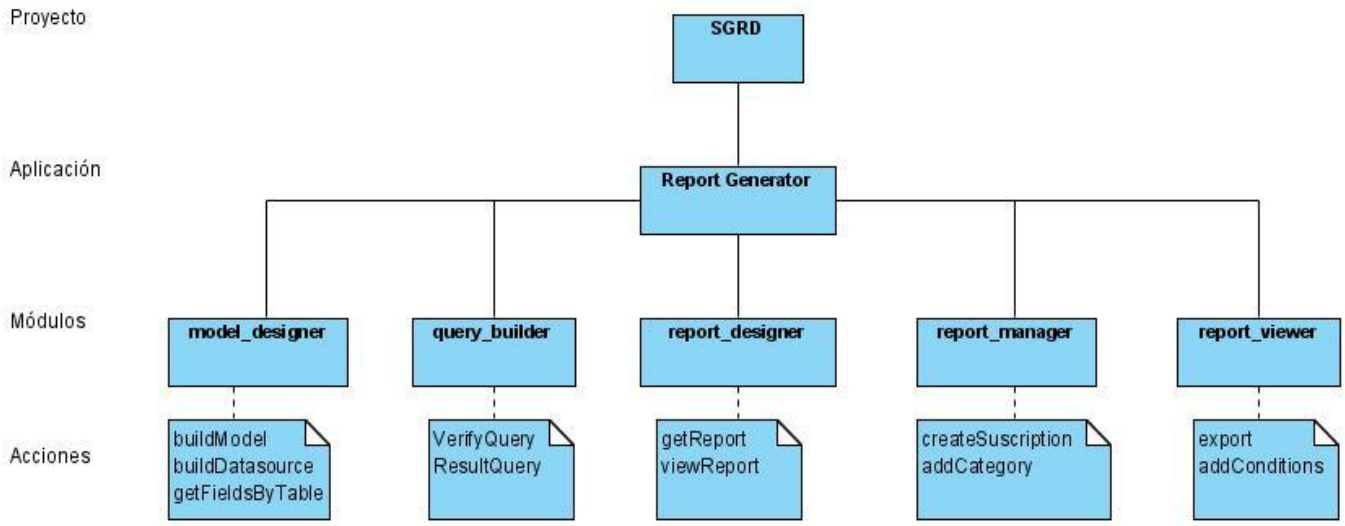


Figura 28 Estructura del proyecto Symfony del sistema.

La Figura 29 muestra la distribución de los componentes de implementación por cada una de las capas de la arquitectura y las relaciones de dependencia que se establecen entre éstos. De forma vertical se encuentra el paquete Common que agrupa componentes generalmente estereotipados como librerías o frameworks utilizados en los diferentes estratos en que se encuentran divididos los componentes software del sistema. Si se analiza de forma horizontal la arquitectura propuesta se encuentra los subsistemas de implementación que desarrollan las funcionalidades previstas. La estructura de cada uno de estos subsistemas se encuentra determinada por el framework de desarrollo utilizado. Cada uno de dichos paquetes encapsulan uno o más componentes que se interrelacionan entre ellos para darle solución a la aplicación.

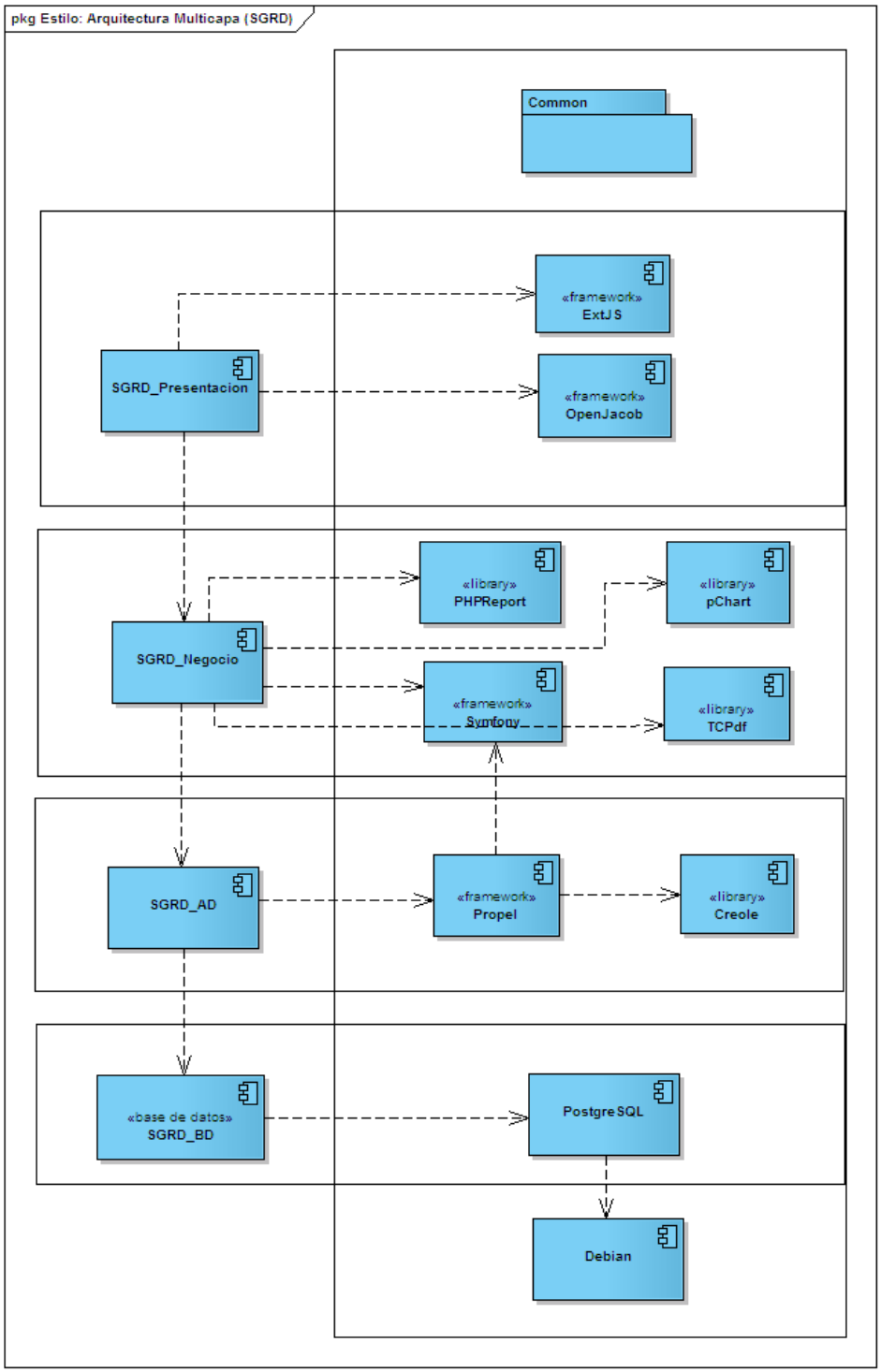


Figura 29 Vista de implementación del SGRD distribuido en la Arquitectura Multicapa

Se presenta a continuación la descripción detallada de los artefactos más importantes de los subsistemas desde el punto de vista de su implementación.

2.3.5.1 SGRD_Presentación

Contiene un conjunto de ficheros y librerías Javascript para la conformación de la presentación del sistema a los usuarios finales. En la Figura 30 el paquete js agrupa los componentes javascript y las relaciones de dependencia entre estos. 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.

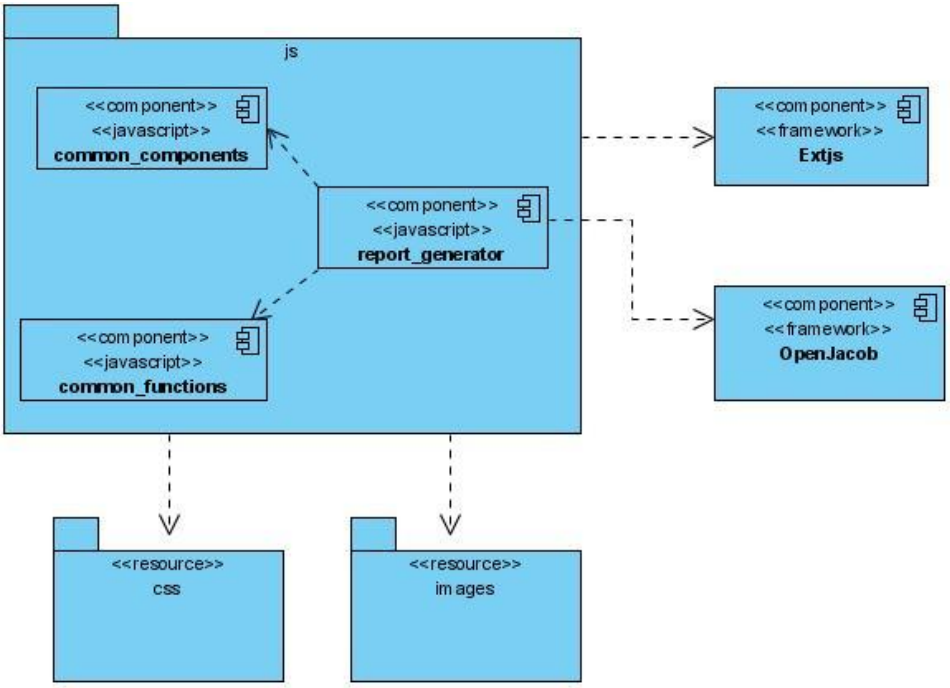


Figura 30 Diagrama de componentes capa de Presentación

2.3.5.2 SGRD_Negocio

La composición de este componente se representa en la Figura 31. El paquete lib recoge las librerías PHPReports (para la generación de los reportes), pChart (generación de gráficos), TCPdf (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.

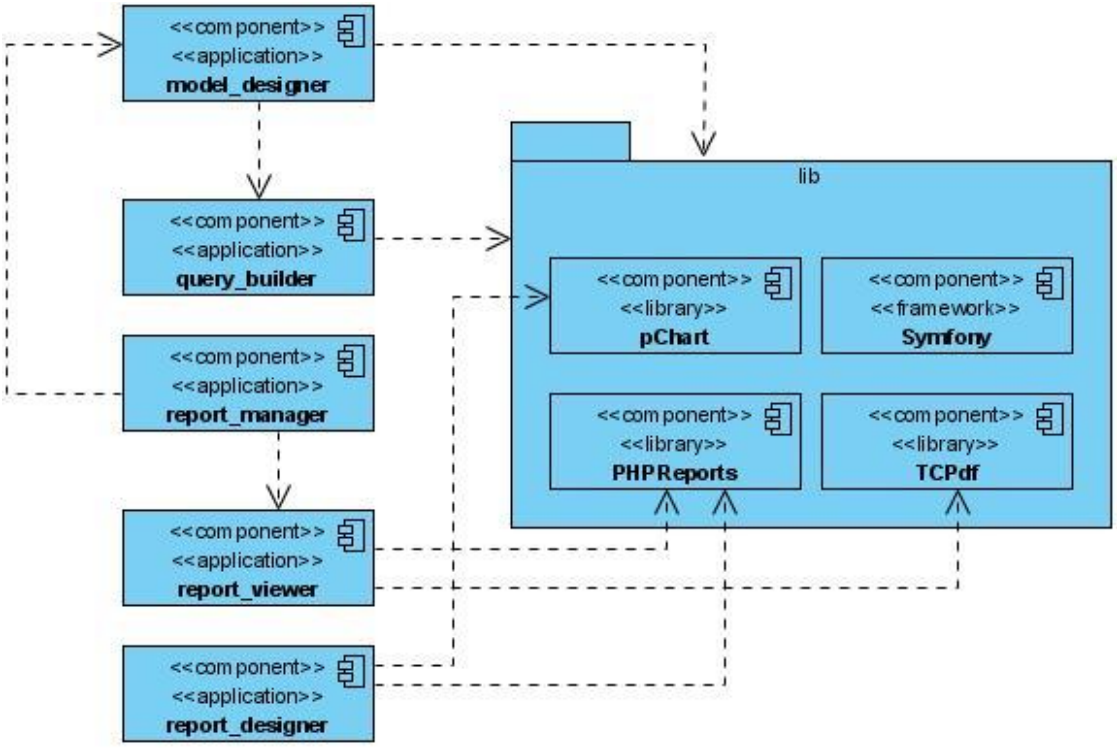


Figura 31 Diagrama de componentes capa de Negocio

2.3.5.3 SGRD_AD

El componente om (object model) representado en la Figura 32 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.

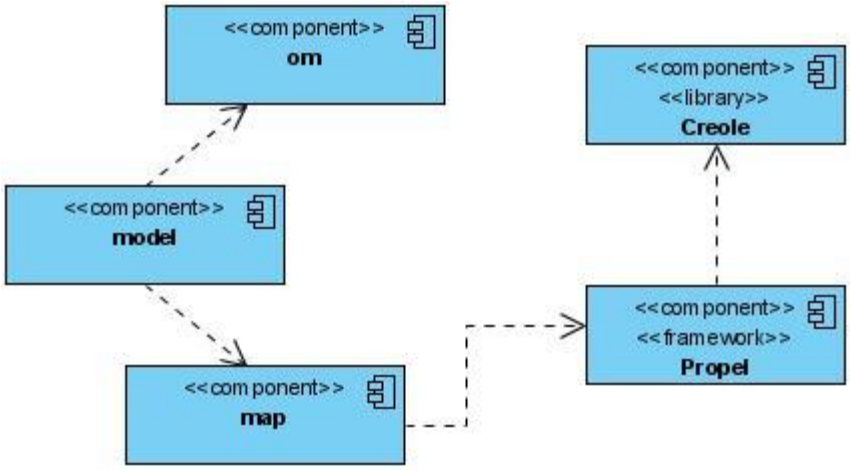


Figura 32 Diagrama de componentes capa de Acceso a Datos

2.3.6 Vista de Datos

En esta vista se presenta el modelo de datos utilizado con la descripción de las tablas más importantes. La siguiente figura muestra la estructura del modelo de datos empleada 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 33).

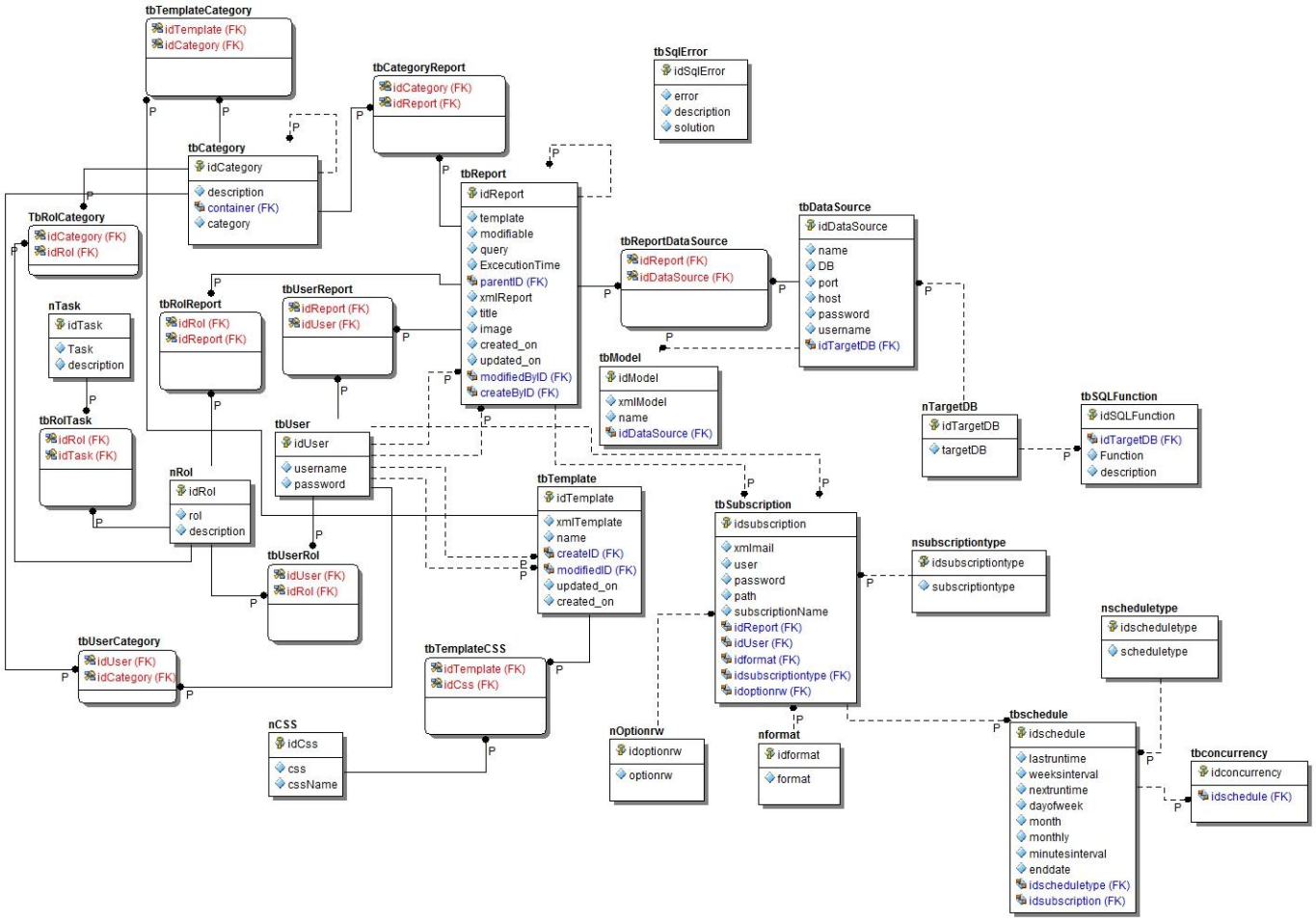


Figura 33 Modelo de datos del sistema

2.3.6.1 Descripción de los elementos del Modelo de datos.

Se identifican varios subsistemas de entidades que responden a funciones específicas:

Subsistema Diseñador de reportes: lo constituyen aquellas entidades que guardan información sobre los reportes creados.

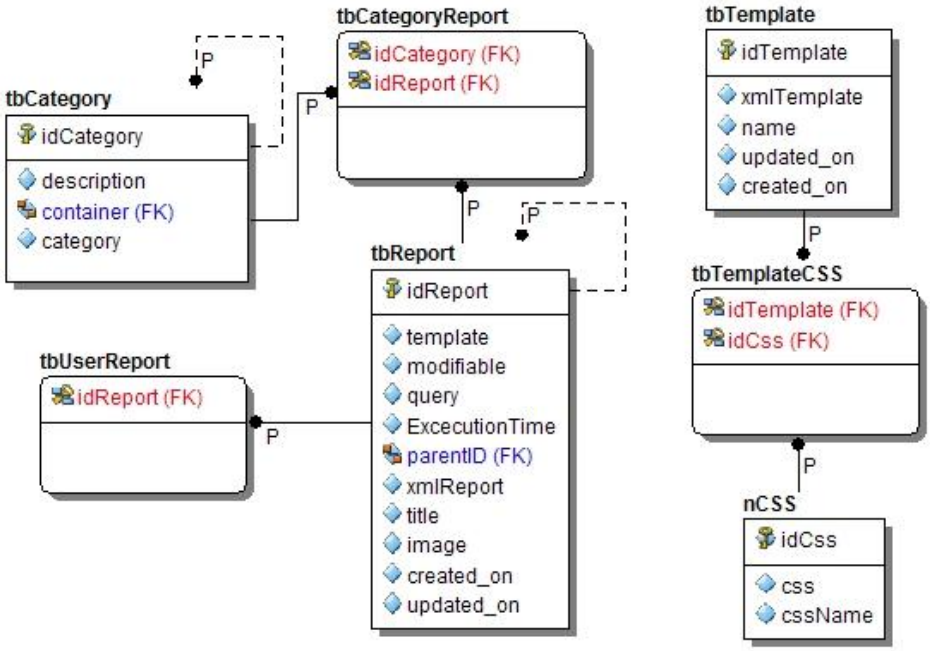


Figura 34 Modelo de datos Diseñador de reportes

Elemento	Identificador	Descripción
Tablas	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

Tabla 2 Descripción tablas principales Diseñador de reportes

Subsistema Diseñador de modelos: el conjunto de entidades que contiene la información que define la estructura de los modelos semánticos y la fuente de datos relacionadas a ellos.

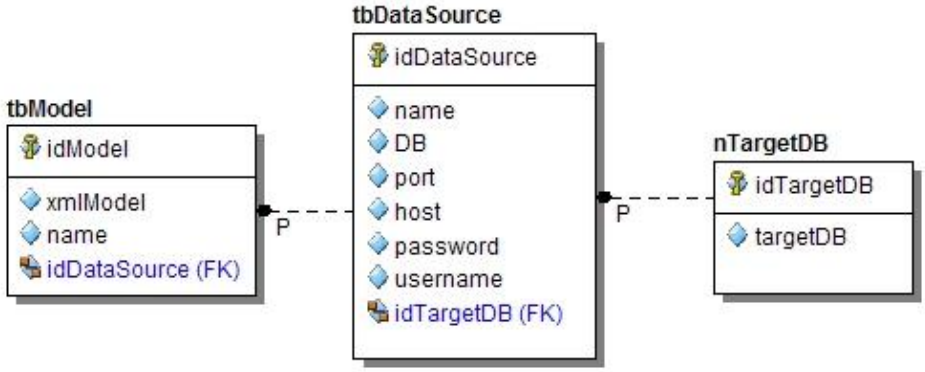


Figura 35 Modelo de datos Diseñador de modelos

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

Tabla 3 Descripción tablas principales Diseñador de modelos

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

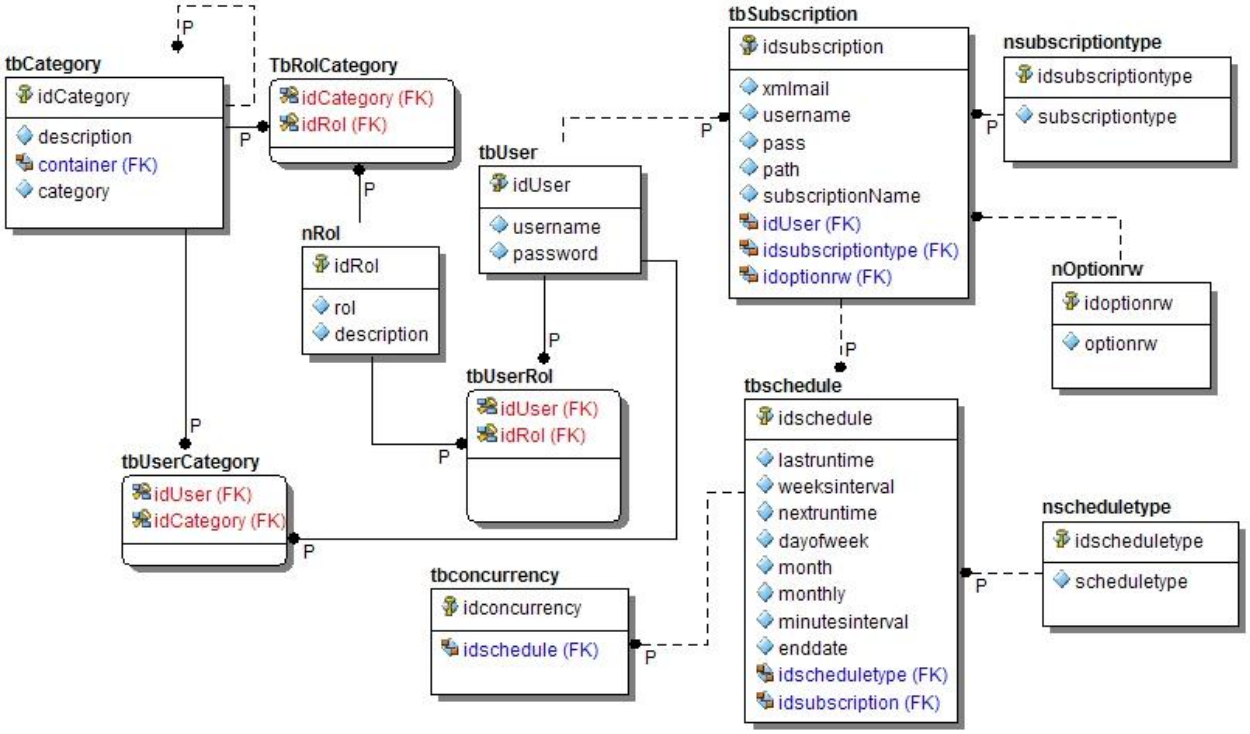


Figura 36 Modelo de datos Administrador de reportes

Elemento	Identificador	Descripción
Tablas	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

Tabla 4 Descripción tablas principales Administrador de reportes

2.4 Conclusiones Parciales

En este capítulo se realizó una descripción de la arquitectura del sistema utilizando el enfoque de las 4+1 vistas que propone RUP teniendo en cuenta los objetivos y restricciones impuestas, 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.

3 Evaluación de la Arquitectura

En el proceso de desarrollo se pueden aplicar diversos enfoques para garantizar el cumplimiento de los requerimientos arquitectónicos, así como la evaluación de las alternativas presentadas. La evaluación provee indicadores que permiten, en las fases tempranas, la oportunidad de resolver problemas que pueden presentarse a nivel arquitectónico (Camacho, et al., 2004).

Independientemente de la metodología implementada, la intención es obtener una arquitectura con la documentación necesaria, y asegurar que el sistema cumple con los servicios y la funcionalidad que espera el usuario, además de los atributos de calidad asociados que deben cumplirse, y que dirigen las decisiones al momento de la construcción de la arquitectura del sistema (Malan, et al., 2005).

En (Bass, 2000) se afirma que cada decisión incorporada en una arquitectura de software puede afectar potencialmente los atributos de calidad. Estas decisiones arquitectónicas están relacionadas con la selección de los estilos y patrones adecuados para cada solución. La siguiente tabla muestra los estilos arquitectónicos empleados en la definición del sistema, destacando los atributos de calidad que propician y los que entran en conflicto (Buschmann, 1996).

Estilo Arquitectónico	Atributos asociados	Atributos en conflicto
Layers	Reusabilidad Portabilidad Facilidad de Prueba	Desempeño Mantenibilidad
Model-View- Controler	Funcionalidad Mantenibilidad	Desempeño Portabilidad
Cliente-Servidor	Modificabilidad Mantenibilidad	Desempeño

Tabla 5 Estilos Arquitectónicos y Atributos de Calidad

Los estilos y patrones ayudan al arquitecto a definir la composición y el comportamiento del sistema de software, y una combinación adecuada de ellos permite alcanzar los requerimientos de calidad (Camacho, et al., 2004).

Se hace necesaria la evaluación de la propuesta arquitectónica, para determinar en qué grado se satisfacen dichos atributos de calidad. Según (In, et al., 2001), el primer paso para la evaluación de una arquitectura es conocer qué es lo que se quiere evaluar.

La evaluación de una arquitectura de software 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. Las mediciones que se realizan sobre una arquitectura de software pueden tener distintos objetivos: cualitativos, cuantitativos y máximos y mínimos teóricos (Bosch, 2000).

La medición cuantitativa busca la obtención de valores que permitan tomar decisiones en cuanto a los atributos de calidad de una arquitectura de software. El esquema general es la comparación con márgenes establecidos, como es el caso de los requerimientos de desempeño, para establecer el grado de cumplimiento de una arquitectura candidata, o tomar decisiones sobre ella. Este enfoque permite establecer comparaciones, pero se ve limitado en tanto no se conozcan los valores teóricos máximos y mínimos de las mediciones con las que se realiza la comparación. La medición de máximo y mínimo teórico contempla los valores teóricos para efectos de la comparación de la medición con los atributos de calidad especificados. El conocimiento de los valores máximos o mínimos permite el establecimiento claro del grado de cumplimiento de los atributos de calidad.

En este sentido, (Bosch, 2000) plantea las técnicas de evaluación: basada en escenarios, basada en simulación, basada en modelos matemáticos y basada en experiencia. Por ser la empleada en el método escogido para evaluar la arquitectura se explica a continuación la técnica basada en escenarios y basada en simulación.

3.1 Evaluación basada en escenarios

Un escenario es una breve descripción de la interacción de alguno de los involucrados en el desarrollo del sistema con éste (In, et al., 2001). 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. 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.

Un 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 (In, et al., 2001). 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, el segundo nivel los atributos de calidad asociados al mismo, los cuales se refinan hasta la obtención de un escenario lo suficientemente concreto para ser analizado (Camacho, et al., 2004). 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.

3.2 Evaluación basada en simulación

La evaluación basada en simulación utiliza una implementación de alto nivel de la arquitectura de software (Bosch, 2000). El enfoque básico consiste en la implementación de componentes de la arquitectura y la implementación –a cierto nivel de abstracción- del contexto del sistema donde se supone va a ejecutarse. La finalidad es evaluar el comportamiento de la arquitectura bajo diversas circunstancias (Camacho, et al., 2004).

El proceso de evaluación basada en simulación sigue los siguientes pasos (Bosch, 2000):

Definición e implementación del contexto: Consiste en identificar las interfaces de la arquitectura de software con su contexto, y decidir cómo será simulado el comportamiento del contexto en tales interfaces.

Implementación de los componentes arquitectónicos: La descripción del diseño arquitectónico debe definir, por lo menos, las interfaces y las conexiones de los componentes, por lo que estas partes pueden

ser tomadas directamente de la descripción de diseño. El comportamiento de los componentes en respuesta a eventos sobre sus interfaces puede no ser especificado claramente, aunque generalmente existe un conocimiento común y es necesario que el arquitecto lo interprete, por lo que éste decide el nivel de detalle de la implementación.

Implementación del perfil: Dependiendo del atributo de calidad que el arquitecto de software intenta evaluar usando simulación, el perfil asociado necesitará ser implementado en el sistema. El arquitecto de software debe ser capaz de activar escenarios individuales, así como también ejecutar un perfil completo usando selección aleatoria, basado en los pesos normalizados de los mismos.

Simulación del sistema e inicio del perfil: El arquitecto de software ejecutará la simulación y activará escenarios de forma manual o automática, y obtendrá resultados de acuerdo al atributo de calidad que está siendo evaluado.

Predicción de atributos de calidad: Dependiendo del tipo de simulación y del atributo de calidad evaluado, se puede disponer de cantidades excesivas de datos, que requieren ser condensados. Esto permite hacer conclusiones acerca del comportamiento del sistema.

3.2.1 Prototipos

En el ámbito de las simulaciones, se encuentra la técnica de implementación de prototipos (prototyping). Esta técnica implementa una parte de la arquitectura de software y la ejecuta en el contexto del sistema. Es utilizada para evaluar requerimientos de calidad operacional, como desempeño y confiabilidad (Camacho, et al., 2004). Se obtiene un resultado de evaluación con mayor exactitud (Bosch, 2000).

La exactitud de los resultados de la evaluación depende, a su vez, de la exactitud del perfil utilizado para evaluar el atributo de calidad y de la precisión con la que el contexto del sistema simula las condiciones del mundo real.

En el núcleo de la evaluación basada en prototipo se encuentra el prototipo de la arquitectura que se aproxima al comportamiento del sistema completo. Según (Martensson, 2006) al no contar con una descripción de los pasos involucrados en la creación de un prototipo de arquitectura, se puede seguir el flujo de trabajo básico utilizado en la evaluación basada en simulación adaptándolo a las necesidades propias en cada caso.

3.3 Método ATAM

Un método de evaluación sirve de guía a los involucrados en el desarrollo del sistema, en la búsqueda de conflictos que puede presentar una arquitectura, y sus soluciones.

El método ATAM está inspirado en tres áreas distintas: los estilos arquitectónicos, el análisis de atributos de calidad y el método de evaluación SAAM, explicado anteriormente (In, et al., 2001). Revela la forma en que una arquitectura específica satisface ciertos atributos de calidad, y provee una visión de cómo los atributos de calidad interactúan con otros; esto es, los tipos de acuerdos que se establecen entre ellos.

Comprende nueve pasos, agrupados en cuatro fases. El Anexo 6 presenta una tabla con las fases y pasos enumerados, junto con su descripción.

3.4 Procedimiento de evaluación propuesto

La estrategia de evaluación implementada consta de 2 fases:

Fase I: Se evalúan los atributos de calidad no observables vía ejecución, determinando el impacto de las decisiones arquitectónicas tomadas en dichos atributos.

Fase II: En esta fase se procede a la evaluación de la arquitectura mediante pruebas de carga y stress, para la medir el impacto arquitectónico sobre los atributos observables vía ejecución.

La Tabla **6** muestra un resumen de las técnicas, instrumentos y método utilizados en este proceso evaluativo.

Atributos de calidad	**Técnica**	**Instrumento**	**Método**
Observables	Simulación	Prototipos	
No observables	Basada en escenarios	Utility Tree	ATAM

Tabla 6 Técnicas, instrumento y método a aplicar según tipo de atributo de calidad

Si bien es cierto que la calidad del sistema depende en gran parte de la implementación, también es cierto que gran parte de ella depende de la arquitectura. Es por ello que el ambiente de análisis, diseño y evaluación de arquitecturas de software se propone como un medio que permitiría la satisfacción de los requerimientos de calidad del sistema establecidos por los involucrados en el desarrollo del mismo.

3.4.1 Evaluando la arquitectura (Fase I)

3.4.1.1 Escenarios

Como resultado de las actividades de aplicar el método ATAM se obtiene el árbol de utilidades, que se enfoca fundamentalmente en 4 atributos de calidad: desempeño, modificabilidad, escalabilidad y configurabilidad. Como nodos hijos de estos se encuentran atributos que constituyen una refinación de estos atributos de calidad (ver Figura 37).

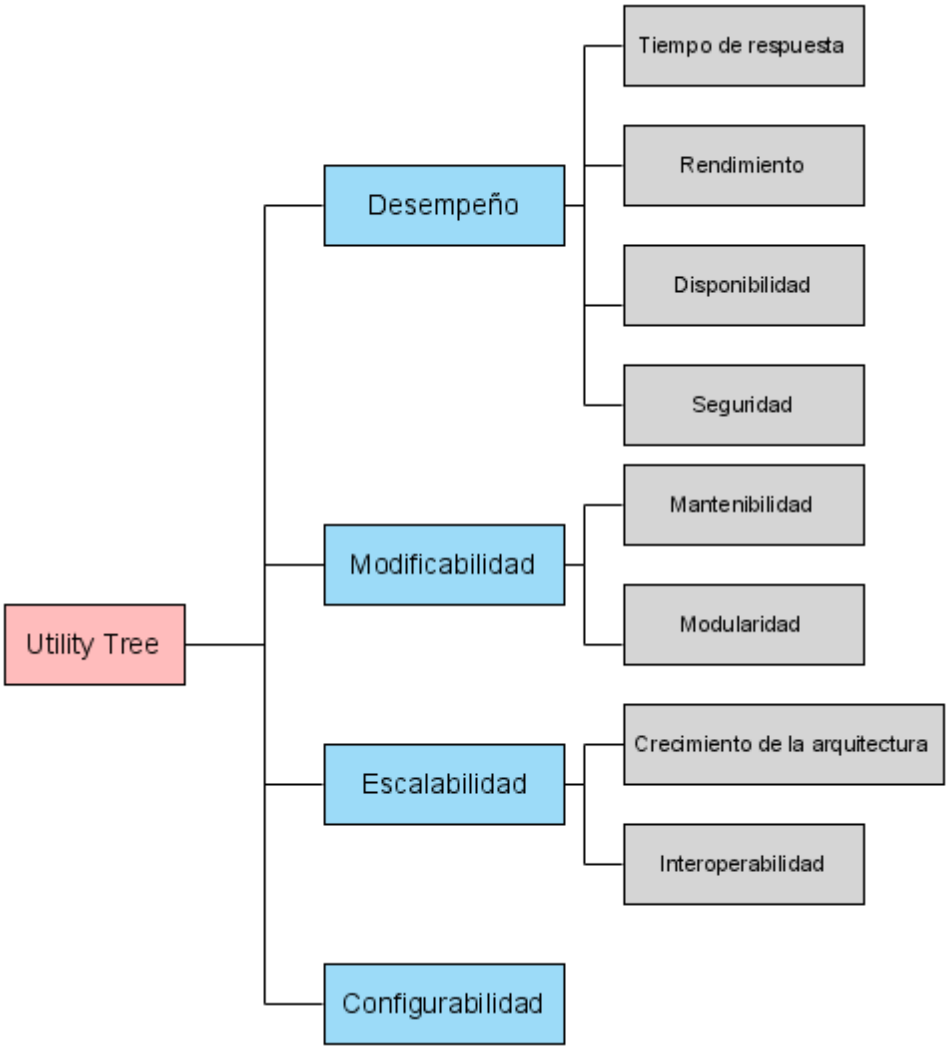


Figura 37 Utility Tree

Los escenarios fueron determinados como resultado de una tormenta de ideas entre los involucrados. Como parte de la refinación de los mismos fueron agrupados y categorizados tomando en consideración el árbol de generación de utilidades obtenido. Luego precede la asignación de la prioridad de cada uno de ellos teniendo en cuenta la importancia del atributo y la estimación de la dificultad para llevarlo a cabo (ver Anexo **7**).

3.4.1.2 Análisis de los escenarios

Para esta etapa no se tomó en consideración el tiempo de respuesta y rendimiento categorizados dentro del atributo desempeño, puesto que serán evaluados en la Fase II.

Se procede al análisis de los escenarios de mayor prioridad arquitectónica, teniendo en consideración cómo reacciona la arquitectura del sistema. Como resultado se obtiene un listado con los puntos sensibles, las relaciones entre atributos, riesgos y no riesgos.

A continuación se muestra el análisis detallado realizado a los principales escenarios involucrados. 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 2 (ver Tabla **7**) 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, actualmente forma parte del Sistema de Gestión Estadística (SIGE). Asociado con este escenario se identificó el riesgo 2 explicado en la próxima sección.

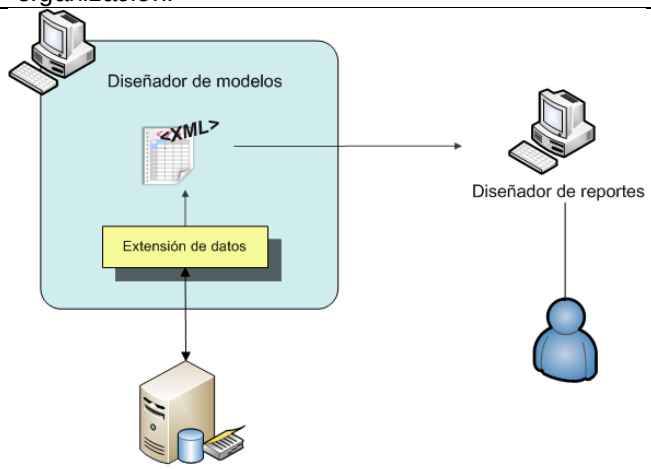
Escenario #:E2	**Descripción:**	El usuario está diseñando un reporte y la base de datos fuente no está disponible en ese momento. El sistema permite la creación del reporte mediante el modelo semántico. (A,M)
Atributo(s):	Desempeño-Disponibilidad	
Ambiente:	Operación normal	
Estímulo:	Creación de un reporte sin la base de datos disponible	
Respuesta:	Creación del reporte mediante modelo semántico	
Decisiones arquitectónicas	**Razonamiento**	
Modelo semántico	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 bd 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.	
Diagrama Arquitectura		

Tabla 7 Escenario #2 Modelo semántico

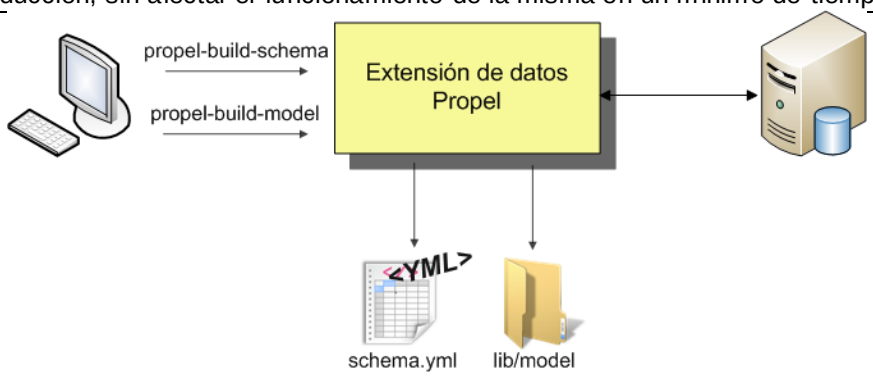
<table>
<tr><td>Escenario #:E9</td><td>Descripción:</td><td>Se requiere agregar una nueva tabla al modelo de datos, se genera automáticamente un nuevo modelo para el acceso a datos(A,M)</td></tr>
<tr><td>Atributo(s):</td><td colspan="2">Modificabilidad- mantenibilidad</td></tr>
<tr><td>Ambiente:</td><td colspan="2">Sistema en explotación</td></tr>
<tr><td>Estímulo:</td><td colspan="2">Agregar nueva tabla al modelo de datos</td></tr>
<tr><td>Respuesta:</td><td colspan="2">Generación automática del modelo y puesta a punto de la aplicación en menos de 1 hora.</td></tr>
<tr><td>Decisiones arquitectónicas</td><td colspan="2">Razonamiento</td></tr>
<tr><td>Estilo Capas
Estilo arquitectónico MVC
Framework de acceso a datos Propel</td><td colspan="2">El framework de acceso a datos Propel posee un grupo de herramientas por línea de comandos para la generación automática de la capa acceso a datos, para este escenario los pasos son:
Construir esquema de la bd: Propel se conecta a la bd 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 bd como objetos.
El estilo en capas posibilita realizar modificaciones a las capas 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 bd del sistema tanto en desarrollo como en producción, sin afectar el funcionamiento de la misma en un mínimo de tiempo.</td></tr>
<tr><td>Diagrama Arquitectura</td><td colspan="2"> 
</td></tr>
</table>

Tabla 8 Escenario #9 Modelo de datos

El escenario 9 (ver Tabla 8) refleja decisiones arquitectónicas que repercuten en el grado de modificabilidad, específicamente mantenibilidad que posee la arquitectura del sistema, relacionado fundamentalmente con el uso del framework Propel para el acceso a datos. La selección de los estilos arquitectónicos MVC y Capas incide directamente sobre estos atributos. A pesar de que el estilo en Capas afecta la modificabilidad (ver Tabla 5) debido a que generalmente cambios en capas inferiores conllevan cambios en las superiores, este impacto se reduce al mínimo porque 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.

Como resultado del análisis de este escenario se detectó el punto de relación entre atributos número dos, 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.

El escenario 17 (ver Tabla 10) 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 SGRD 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 y no a través de servicios web que suelen ser un poco más lentos debido a características propias de este tipo de interfaz de comunicación. Este tipo de acceso acepta varios parámetros que afectan la forma en que un reporte es mostrado como por ejemplo el formato de salida y si se desea obtener solo el reporte o visualizarlo en el Visor de reportes.

A través de este método, sistemas externos pueden contener URLs de acceso directo a reportes específicos o crearlas dinámicamente.

El acceso por URL (link) tiene la siguiente sintaxis:

http: //<*servidor*>:<*puerto*>/<*report_generator.php*>/<*módulo*>/<*acción*>

El controlador frontal es *report_generator.php,* a continuación se especifica el *módulo* (report_viewer) y la *acción* a ejecutarse (getReport), el parámetro a pasar el identificador del reporte, ejemplo: id=1.

Para llevar a cabo la integración donde se construye dinámicamente la URL se efectúan los siguientes pasos:

1. Incluir en la aplicación el fichero javascript DinamicReport.js, que contiene todas las operaciones necesarias para obtener y filtrar un reporte (ver Anexo 9).
2. Instanciar el objeto de esta clase con los datos necesarios como el servidor, puerto y render (es opcional).
3. Llamar las funciones de la clase, en dependencia de lo que se necesite.

Una vez se ejecute el paso 2, en el constructor de la clase se inicializan otros parámetros por defecto que cambiarán su valor en dependencia de las funciones que sean invocadas (ver Tabla 9).

Parámetro	**Descripción**	**Ejemplo**
Show	Para que el visor muestre todo el reporte además de generarlo (sólo lectura).	Show=1
OutSide	Para que reconozca las peticiones desde otro entorno (sólo lectura).	OutSide = 1
Limit	Cantidad de tuplas que desea ver.	Limit = 20
ReportID	Identificador del reporte al cual se desea acceder.	ReportID = 4
CategoryID	Identificador de la categoría a la cual pertenece el reporte.	CategoryID = 1
ReportName	Nombre del reporte	ReportName = 'Mi reporte'
Format	Formato en el cual se desea obtener el reporte.	Format = 'pdf'

Tabla 9 Parámetros de la clase DinamicReport

Escenario #:E17	**Descripción:**	Se requiere que las aplicaciones de la empresa utilicen el generador de reportes para extraer información de los procesos de negocio (A,A)
Atributo(s):	Escalabilidad - Interoperabilidad	
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	**Razonamiento**	
Estilo capas	La integración por URL es uno de los métodos de integración.	
Estilo arquitectónico MVC	Está diseñada para brindar un máximo nivel de rendimiento para el acceso remoto a los reportes comunicándose directamente con el Servidor de reportes.	
Estándar de datos XML	SGRD chequea si tiene permisos y una cookie de autenticación válida antes de acceder al reporte.	
Formato JSON	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.	
Diagrama Arquitectura		

Tabla 10 Escenario #17 Integración con otros sistemas

3.4.1.3 Resultados

Se analizaron un total de 18 escenarios, identificándose 3 puntos sensibles, 2 puntos de relación entre atributos, 2 riesgos y 2 no riesgos.

Puntos sensibles

1. Cambio del sistema operativo.
2. Soportar navegador Internet Explorer.
3. Aumento de usuarios y del tamaño del reporte a generar.

Puntos de relación entre atributos

1. Posibilitar la entrada de parámetros para el filtrado mejora el rendimiento a la vez que afecta la disponibilidad del reporte por entrada no válida.
2. Las actualizaciones a la base de datos asegura la mantenibilidad a la vez que afecta la disponibilidad.

Riesgos

1. Estrategia de manejo y recuperación de errores no está totalmente definida.
2. Base de datos del cliente no disponible.

No riesgos

1. La creación del modelo semántico mejora el rendimiento y seguridad del sistema.
2. Se garantiza el diseño del reporte una vez creado el modelo semántico.

3.4.2 Evaluando la arquitectura (Fase II)

En esta fase se procede a la evaluación utilizando la técnica de simulación con el instrumento prototipo. El flujo de trabajo de la evaluación basada en simulación debe ser adaptado para incorporar algunos pasos a desarrollar en la evaluación basada en prototipo. Estas adaptaciones se describen a menudo que se van explicando los pasos de la evaluación.

Para ejecutar una evaluación basada en prototipo es necesario cumplir con algunas condiciones:

1. Debe haber al menos una arquitectura definida, si el objetivo de la evaluación es comparar arquitecturas alternativas es necesario contar con más de una.

2. Si se desea evaluar el rendimiento de una o más arquitecturas candidatas deben estar disponibles todos los componentes que forman parte de cada una.

Además es recomendable pero no necesario tener disponible la plataforma sobre la cual se ejecutará el sistema. Si es posible hacer las pruebas sobre el hardware correcto, los resultados serán más exactos.

Luego de integrar las adaptaciones al método de evaluación obtenemos los siguientes pasos a seguir para la evaluación basada en prototipo.

Definir el objetivo de la evaluación: Definir que debe ser evaluado, si se está buscando una arquitectura entre varias candidatas o componentes arquitectónicos y qué atributos de calidad se está interesado en evaluar.

Integrar los componentes arquitectónicos: Adaptar los componentes arquitectónicos de forma tal que el framework pueda interactuar con ellos.

Implementar el modelo de la arquitectura: Implementar un modelo de la arquitectura soportado por el framework, este modelo junto al framework y los componentes que serán evaluados constituyen un prototipo ejecutable.

Ejecutar el prototipo: Ejecutar el prototipo y obtener información para el próximo paso, asegurar que el ambiente de ejecución sea lo más parecido posible al hardware donde será implantado definitivamente el sistema.

Analizar los registros: Analizar los registros generados y extraer la información relacionada con los atributos de calidad que están bajo evaluación. El análisis debe ser lo más automatizado posible porque la cantidad de información puede volverse abrumadora.

Predecir atributos de calidad: Predecir los atributos de calidad que serán evaluados basándose en la información de los registros analizados.

Si es necesario, iterar: Se recomienda iterar si los resultados no son los esperados o si se cree que los resultados pueden ser mejores, se pueden ajustar elementos del prototipo en general e iterar para comparar resultados.

En la ejecución de los prototipos es necesario definir las variables que afectan directamente el desempeño del sistema así como las métricas utilizadas en la simulación.

3.4.2.1 Variables de rendimiento

Es necesario conocer de antemano las necesidades de carga de trabajo a la hora de determinar los requisitos de rendimiento. Los factores que intervienen en el rendimiento y la carga de trabajo son los siguientes:

- **El número de usuarios que solicitan la creación de informes de forma simultánea:** mientras más sesiones estén activas simultáneamente, más recursos de equipo se consumirán.

- **El tamaño y la complejidad de las definiciones de los informes:** los informes complejos que procesan un gran número de filas exigen muchos más recursos que los informes sencillos que procesan sólo unas cuantas.
- **El rendimiento de los sistemas de origen de datos de los que el generador de reportes obtiene los datos de los informes:** Si las consultas que se ejecutan en estos sistemas son lentas, parecerá que los informes se ejecutan con lentitud.
- **El formato solicitado al presentar un informe:** Los formatos del tipo PDF o Microsoft® Excel® consumen más recursos que el formato HTML.

Una serie de cuestiones relacionadas con el diseño, o que puede configurar el usuario, también influyen en el rendimiento de los informes. Entre ellas se incluyen las siguientes:

- La configuración de la aplicación, según esté especificada en los archivos de configuración del sistema y el sistema operativo Ubuntu.
- La configuración y el diseño del hardware que admite la aplicación de generación de informes.
- Factores externos, tales como la infraestructura de entrega, incluida la capacidad de la red, el rendimiento del servidor de correo electrónico para la entrega de suscripciones e incluso la disponibilidad y rendimiento de los recursos compartidos de archivos.

La carga de trabajo puede variar con el tiempo y experimentar fluctuaciones de uso de naturaleza mensual, semanal o diaria en función de una serie de factores empresariales.

3.4.2.2 Métricas para el análisis

Al determinar una línea de referencia de rendimiento para el sistema, debe determinar también métricas con significado. Se utilizó las siguientes métricas en las pruebas:

- Sesión simultánea
- Solicitudes promedio por segundo
- Tiempo de respuesta promedio

La métrica Sesión simultánea es una buena variable para determinar las demás métricas. Se comienza con un número de sesiones simultáneas y se aplican incrementos con un índice de crecimiento en cada ejecución para generar una prueba de carga de trabajo.

La métrica Solicitudes promedio por segundo le ayuda a saber el número de solicitudes Web que puede procesar correctamente cada una de las posibles configuraciones del sistema de forma simultánea. Dado

que el sistema no se dedica a procesar una carga de trabajo en concreto, cada prueba de sistema debe estar en condiciones de procesar en igualdad de condiciones cada solicitud.

La métrica Tiempo de respuesta promedio (tiempo para la recepción del último byte o TTLB) es una métrica habitual que se utiliza para comprender el tiempo que tarda en responder una solicitud de informe. Es obvio que cuanto menor sea el tiempo de respuesta, mejor.

3.4.2.3 Implementación del perfil

Objetivo de la evaluación

Se definió como objetivo de la evaluación el desempeño del SGRD debido a que este deberá manipular en ocasiones grandes volúmenes de información que los usuarios necesitan visualizar en un momento determinado. Este atributo se evalúa teniendo en cuenta el rendimiento y el tiempo de respuesta, que van a ser los perfiles sobre los cuales se van a generar una serie de escenarios para determinar cómo se comporta el rendimiento del sistema sobre la arquitectura.

Integración de los componentes arquitectónicos

Para integrar los componentes arquitectónicos se montaron sobre el framework Symfony 1.1.7 los componentes que soportan las funcionalidades implicadas en el objetivo de evaluación.

Implementación del modelo de la arquitectura

Con el framework listo para utilizarse se creó el modelo de la arquitectura, este fue basado en la arquitectura del sistema de navegación y centrado en modelar el flujo de información y los mensajes entre los distintos componentes.

Ejecución del prototipo

El prototipo fue ejecutado en una plataforma Intel con un procesador Intel Pentium 4 HT a 3.0 GHz y 1 GB de RAM, sistema operativo Ubuntu Server 8.10. El prototipo de arquitectura estuvo en ejecución durante 3 horas, período de tiempo en el que se realizaron las pruebas utilizando la herramienta JMETER para simular la concurrencia y generar los registros.

Se elaboraron dos perfiles, el primero contiene 6 escenarios y el segundo 7 escenarios, mostrados en la Tabla 11 y la Tabla 12 respectivamente.

Atributo:	Desempeño	**Perfil:**	Tiempo de respuesta
Escenario 1			
Simular 1 conexiones simultáneas. Recuperar 1000 tuplas.			
Escenario 2			
Simular 2 conexiones simultáneas. Recuperar 1000 tuplas.			
Escenario 3			
Simular 3 conexiones simultáneas. Recuperar 1000 tuplas.			
Escenario 4			
Simular 5 conexiones simultáneas. Recuperar 1000 tuplas.			
Escenario 5			
Simular 10 conexiones simultáneas. Recuperar 1000 tuplas.			
Escenario 6			
Simular 20 conexiones simultáneas. Recuperar 1000 tuplas.			

Tabla 11 Escenarios del perfil Tiempo de respuesta

Atributo:	Desempeño	**Perfil:**	Rendimiento
Escenario 1			
Simular 1 conexión. Recuperar 500 tuplas.			
Escenario 2			
Simular 1 conexión. Recuperar 1000 tuplas.			
Escenario 3			
Simular 1 conexión. Recuperar 5000 tuplas.			
Escenario 4			
Simular 1 conexión. Recuperar 10000 tuplas.			
Escenario 5			
Simular 1 conexión. Recuperar 20000 tuplas.			
Escenario 6			
Simular 1 conexión. Recuperar 50000 tuplas.			
Escenario 7			
Simular 1 conexión. Recuperar 75000 tuplas.			

Tabla 12 Escenarios del perfil Rendimiento

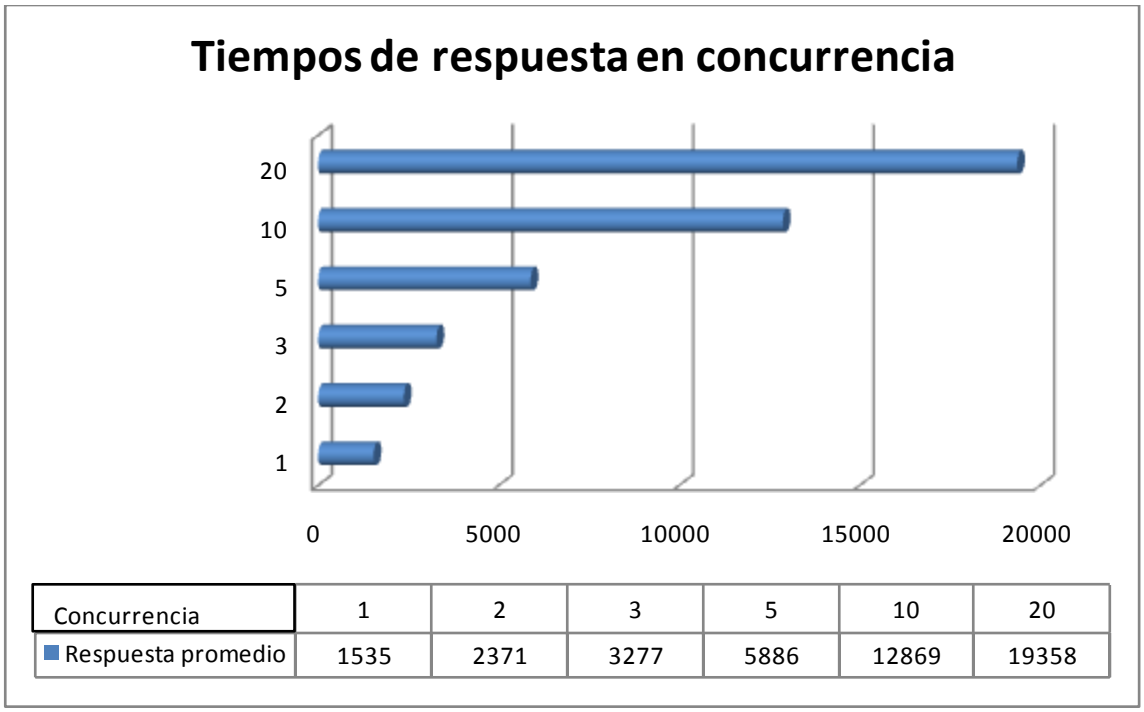
El primer perfil sirve para analizar el comportamiento del sistema ante la concurrencia, simulando la generación de reportes de un tamaño de 1000 tuplas de forma concurrente y aumentando la cantidad de conexiones desde 1 a 20. Vale destacar que la simulación de la concurrencia con este tipo de herramientas supone una carga en el servidor casi tres veces mayor a lo que realmente provocaría igual número de conexiones en un entorno real, debido a que en la simulación se lanzan las conexiones en el mismo instante de tiempo hacia el servidor y en un ambiente común es poco probable que lleguen dos o más peticiones al servidor en el mismo instante de tiempo.

En el segundo perfil se va aumentando el número de registros a recuperar desde una única conexión, esto nos permite medir el rendimiento del sistema en términos de tiempo de respuesta promedio.

3.4.2.4 Resultados simulación del sistema

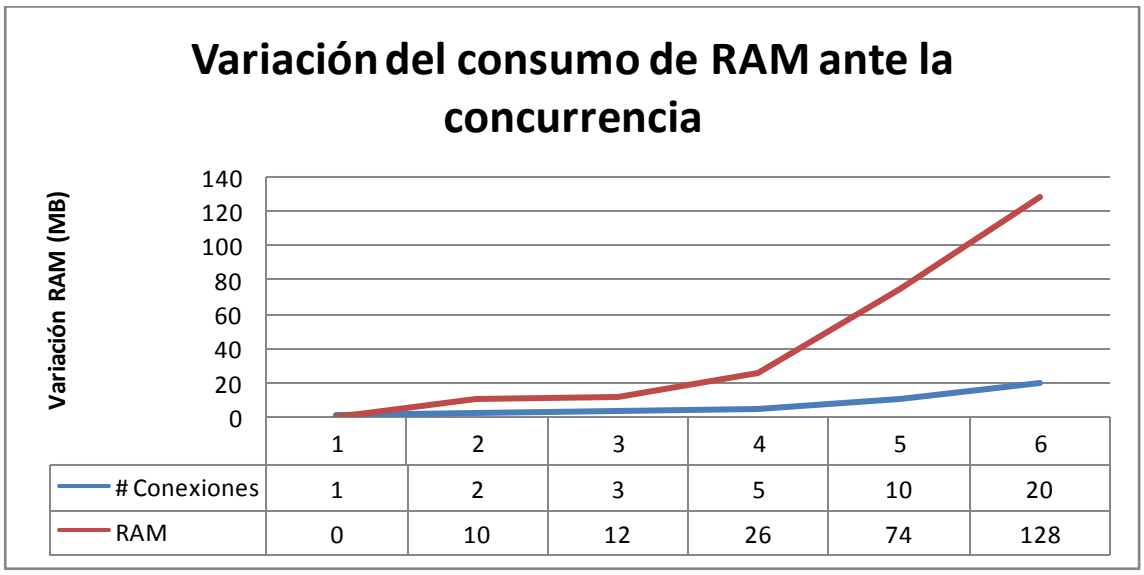
Luego de la ejecución de cada uno de los perfiles mediante la simulación de cada escenario asociado, se obtuvieron resultados cuantitativos utilizando las métricas definidas. Se demostró por una parte, la robustez de la arquitectura al soportar altos niveles de concurrencia y por otro lado el desempeño del sistema al comportarse de forma estable durante el aumento del tamaño de los reportes a generar.

La Figura 38 muestra los resultados obtenidos luego de la ejecución del primer perfil, se registró el tiempo de respuesta promedio para cada uno de los escenarios en milisegundos (ver Anexo 10). Se puede arribar a la conclusión que para un reporte de tamaño promedio un usuario no debe esperar más de 20 segundos por la entrega del mismo, a pesar de que el servidor se encuentre trabajando con un alto nivel de concurrencia. También se monitoreó el consumo de RAM bajo estas condiciones, este recurso es el más afectado por la concurrencia puesto que se crea un hilo de ejecución por cada conexión. Se observa que para los requerimientos de hardware especificados el consumo de este recurso se comporta de forma moderada.



Concurrencia	1	2	3	5	10	20
Respuesta promedio	1535	2371	3277	5886	12869	19358

Figura 38 Tiempo de respuesta en concurrencia.



	1	2	3	4	5	6
# Conexiones	1	2	3	5	10	20
RAM	0	10	12	26	74	128

Figura 39 Consumo de RAM en el perfil Concurrencia.

La Figura 40 presenta una gráfica que recoge los tiempos de respuesta promedio como resultado de la ejecución de los escenarios asociados al perfil de Rendimiento (ver Anexo 11). Como se puede observar para un reporte que recupera 75 000 registros el tiempo de respuesta medio es de 1.6 minutos, tiempo inferior al previsto como aceptable (3 minutos). En la simulación de este perfil se monitoreó el porciento de uso de CPU, debido a que mientras mayor sea el tamaño del informe, se necesita más tiempo de ejecución. Como resultado, el nivel de procesamiento se mantiene estable al no sobrepasar en ningún caso el 50% (ver Figura 41).

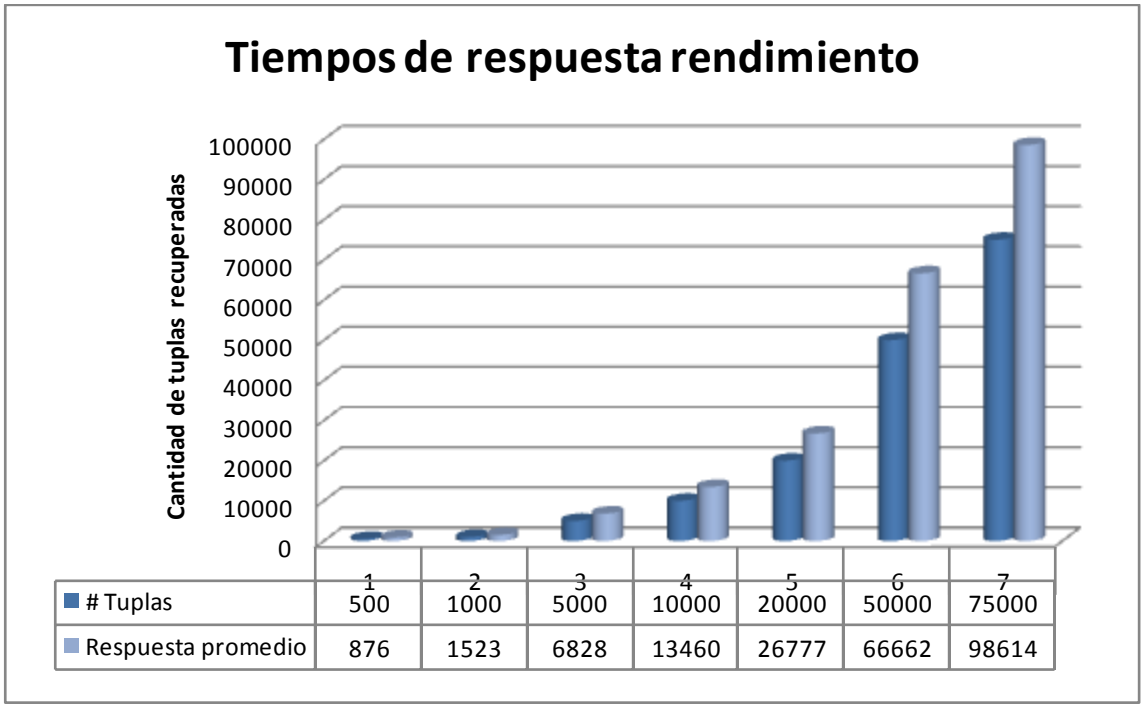


Figura 40 Tiempo de respuesta perfil Rendimiento

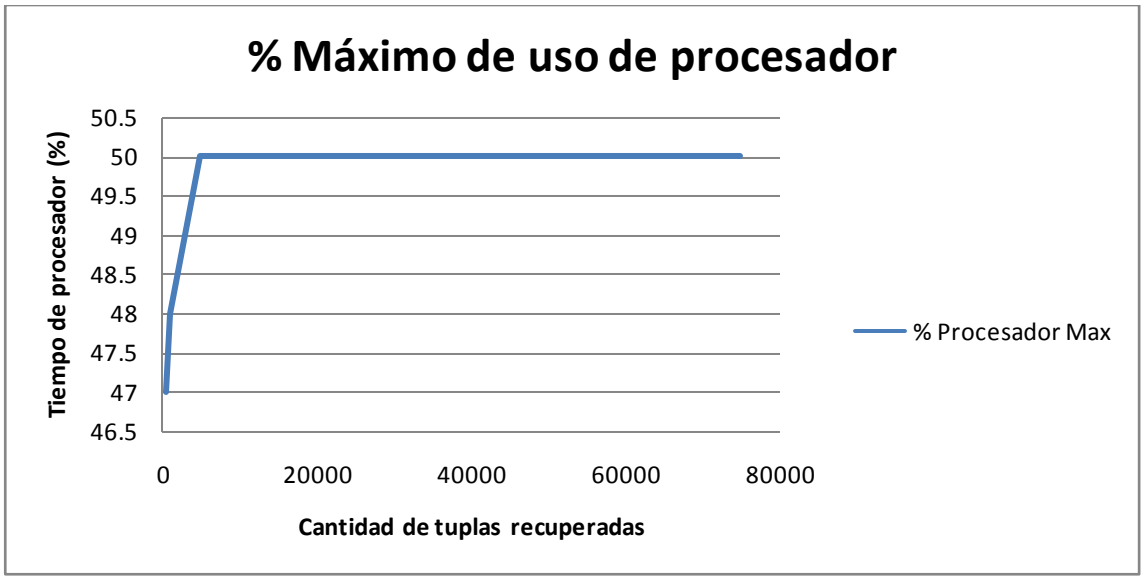


Figura 41 Uso del procesador perfil Rendimiento

Predecir atributos de calidad

Basado en la información extraída de los registros se concluyó que la plataforma sobre la que se corrió el prototipo fue más que suficiente para obtener resultados satisfactorios durante las pruebas, lo que demuestra que es posible la implantación del sistema en entornos menos potentes. Como resultado de la simulación de los escenarios se evidenció que la arquitectura responde a los atributos de calidad desempeño, rendimiento y tiempo de respuesta de forma satisfactoria.

Si es necesario, iterar

Debido a los resultados satisfactorios se decidió que la arquitectura cumplía con las expectativas y no era necesario hacer grandes ajustes para una nueva iteración.

3.4.3 Casos de éxito

La versión del sistema 1.5 se encuentra desplegada con éxito en 5 proyectos de la UCI, uno de ellos es el ERP Cubano Cedrux, donde en cada uno de sus módulos se utiliza SGRD en sus diferentes niveles de integración para la gestión de los reportes. Dentro de las configuraciones de integración desplegadas se encuentran:

1. La base de datos de SGRD se encuentra como un esquema de la base de datos del ERP.

2. El sistema de reportes se encuentra integrado como otro módulo.
3. Se instancia la aplicación Visor de reportes desde cada una de las aplicaciones del ERP.
4. Integración por URL con la construcción dinámica de la misma.

El Anexo 12 muestra el aval correspondiente a este escenario de despliegue, manifestando la conformidad con los servicios prestados.

3.5 Conclusiones Parciales

En este capítulo se aplicó el método de evaluación de arquitecturas ATAM y la técnica basada en escenarios con el instrumento Utility Tree, en aras de determinar cómo la propuesta arquitectónica cumple con los atributos de calidad requeridos. Como resultado de este proceso se identificaron algunos riesgos, no riesgos, puntos sensibles y relaciones entre atributos presentes en la arquitectura del sistema. En una segunda fase se aplicó la técnica basada en simulación con el instrumento prototipo, realizando pruebas de carga y estrés para medir el rendimiento del sistema, como resultado se obtuvieron tiempos de respuesta aceptables de acuerdo a los requerimientos no funcionales del sistema.

Conclusiones Generales

- Se realizó el estudio del estado del arte de algunos sistemas generadores de reportes, llegando a la conclusión que ninguno cumple cabalmente con los requisitos solicitados por el cliente.
- Se seleccionaron las herramientas y tecnologías a utilizar, siguiendo el principio de software libre.
- Se describió la arquitectura de software del sistema utilizando diferentes niveles de abstracción, haciéndose uso de estilos y patrones que potencian atributos de calidad.
- La aplicación del estilo multicapas atribuye una lógica organizacional que conserva la alta cohesión y el bajo acoplamiento entre los componentes del sistema.
- La aplicación del estilo 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 confeccionó el documento de Descripción de la Arquitectura quedando plasmado las características del ambiente de desarrollo del sistema así como los elementos esenciales del Modelo de Casos de Uso, Modelo de Diseño, Modelo de Despliegue, Modelo de Implementación y Modelo de Datos.
- Se evaluó la arquitectura con la aplicación del método ATAM y la técnica basada en simulación, demostrando cómo esta potencia los atributos de calidad requeridos.
- Se demostró que la elección y aplicación adecuada de decisiones arquitectónicas como los estilos, patrones y frameworks conllevaron a la obtención de una arquitectura flexible en términos de implementación, robusta en cuanto a rendimiento y lo suficientemente escalable como para evolucionar ante la necesidad de nuevas funcionalidades.

Recomendaciones

- Eliminar las dependencias existentes con el sistema operativo en la programación y entrega del módulo Administrador de reportes.
- Crear fachada en la capa de presentación para soportar correctamente el uso de diferentes navegadores web.
- Definir desde el punto de vista arquitectónico una estrategia centralizada para el manejo y recuperación de errores.
- Crear un sistema de seguimiento de trazas que posibilite monitorear el seguimiento de la generación de informes dentro del servidor.

Referencias Bibliográficas

Alvez, Pablo, Foti, Patricia y Scalone, Marco. 2006. *Documento de Arquitectura de Software.* 2006.

Barbacci, Mario, y otros. 2003. *Using the Architecture Tradeoff Analysis Method (ATAM) to Evaluate the Software Architecture for a Product Line of Avionics Systems: A Case Study.* s.l. : CMU/SEI-2003-TN-012, 2003. Technical Note.

Bass, L., Klein, M., & Bachmann, F. 2000. *Quality Attribute Design Primitives.* s.l. : Software Engineering Institute, Carnegie Mellon University, 2000. Technical Report.

Bass, Len y Kazman, Rick. 1999. *Architecture-Based Development.* s.l. : CMU/SEI-99-TR-007, 1999. TECHNICAL REPORT. ESC-TR-99-007.

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

Bastarrica, María Cecilia. 2003. *Atributos de Calidad y Arquitectura del Software.* 2003.

Bosch, Jan. 2000. *Design and use of software architectures: adopting and evolving a product-line approach.* Universidad de Michigan : Addison-Wesley, 2000. ISBN 0201674947, 9780201674941.

Buschmann, Frank y colectivo de autores. 1996. *Patterns Oriented Software Achitecture.* 1996. I – II.

—. 4 de mayo 2007. *Patterns Oriented Software Achitecture: A Pattern Language for Distributed Computing.* 4 de mayo 2007. Volumen 4.

cakephp.org. cakephp.org. [En línea] [Citado el: 12 de Febrero de 2009.] http://cakephp.org/.

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

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

Clements, Paul y Northrop, Linda. 1996. "Software architecture: An executive overview". *Technical Report, CMU/SEI-96-TR-003, ESC-TR-96-003.* Febrero de 1996.

Creole project. 2009. Creole. [En línea] 2009. [Citado el: 6 de Mayo de 2009.] http://creole.phpdb.org/trac.

CrystalReports.com. 2009. CrystalReports.com. [En línea] 2009. [Citado el: 2 de febrero de 2009.] http://www.crystalreports.com/.

DeRemer, Frank y Kron, Hans. 1976. *Programming-in-the-large versus programming-in-the-small.* s.l. : lEEE Transaction in Software Engineering, 1976.

Designing software for ease of extension and contraction. **IEEE. Marzo 1979.** 2, s.l. : lEEE Transactions on Software Engineering, Marzo 1979, Vol. 5.

ExtJS. 2009. Ext 2.2.1 - API Documentation. *Ext 2.2.1 - API Documentation.* [En línea] 2009. [Citado el: 2009 de Mayo de 6.] http://extjs.com/deploy/dev/docs/.

Fielding, Roy Thomas. 2000. *Architectural styles and the design of network-based software architectures.* University of California, Irvine : Tesis doctoral, 2000.

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

GrapeCity, inc. www.datadynamics.com. [En línea] [Citado el: 26 de enero de 2009.] http://www.datadynamics.com/Products/ProductOverview.aspx?Product=ARNET3.

Herrington, Jack D. 2006. lBM. *Five common PHP design patterns.* [En línea] 18 de Julio de 2006. [Citado el: 14 de Febrero de 2009.] http://www.ibm.com/developerworks/library/os-php-designptrns.

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

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

ISO/IEC. 1998. *ISO/IEC 9126.* s.l. : ISO/IEC (Intenational Standart Organitation), 1998.

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

Jaimes Rojas, Nolberto. 2008. *Conviviendo con sistemas legados.* Universidad de los Andes, Bogotá, Colombia : s.n., 2008. Vol. I.

Jasperforge.org. 2008. Jasperforge.org. [En línea] 2008. [Citado el: 26 de enero de 2009.] http://jasperforge.org/plugins/project/project_home.php?group_id=102.

Kazman, Rick, Nord, Robert L. y Klein, Mark. 2003. *A Life-Cycle View of Architecture Analysis and Design Methods.* s.l. : TECHNICAL NOTE. CMU/SEI-2003-TN-026, 2003.

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

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

Kruchten, P. 1995. *The 4+1 View Model of Architecture.* s.l. : IEEE Software, 1995.

Malan, Ruth y Bredemeyer, Dana. 2005. The Visual Architecting Process. *http://www.bredemeyer.com.* [En línea] 5 de Julio de 2005. [Citado el: 18 de Abril de 2009.] http://www.bredemeyer.com/papers.htm.

Mark Klein, Rick Kazman. Octubre de 1999. *Attribute-based architectural styles.* Carnegie Mellon University : Technical Report, CMU/SEI-99-TR-022, Octubre de 1999. ESC-TR-99-022.

Martensson, Frans. 2006. *SOFTWARE ARCHITECTURE QUALITY EVALUATION: APPROACHES IN AN INDUSTRIAL CONTEXT.* s.l. : Blekinge Institute of Technology, 2006. Series No. 2006:03.

Mary Shaw, David Garlan. 1996. *Software Architecture: Perspectives on an Emerging Discipline.* s.l. : Prentice Hall, 1996. 0131829572, 9780131829572.

Microsoft Corporation. [En línea] [Citado el: 6 de febrero de 2009.] http://msdn.microsoft.com/es-es/library/ms225260%28VS.80%29.aspx.

—. 2006. Tecnologías de Microsoft SQL Server 2005. [En línea] 14 de Abril de 2006. [Citado el: 12 de Enero de 2009.] http://msdn.microsoft.com/es-es/library/ms155786(SQL.90).aspx.

Parnas, David. 1972. "On the Criteria for Decomposing Systems into Modules.". s.l. : Communications of the ACM 15(12), 1972, págs. pp. 1053-1058.

Paul Clements, Mark Klein, Rick Kazman. 2002. *Evaluating software architectures: methods and case studies.* Universidad de Michigan : Addison-Wesley, 2002. ISBN 020170482X, 9780201704822.

Planos arquitectónicos. El Modelo de las "4 + 1" vistas de la Arquitectura de software. **Kruchten, Philippe.**

Potencier, Fabien. 2009. symfony-project. [En línea] 2009. [Citado el: 12 de Febrero de 2009.] http://www.symfony-project.org/.

Potencier, Fabien y Zaninotto, François. *Symfony. La guía definitiva.*

Pressman, R. 2002. *Ingeniería de Software. Un Enfoque Práctico.* s.l. : Quinta Edición, 2002.

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

Rangel, Eustaquio. 2006. *PHPReports Manual.* Brasil : s.n., 2006.

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

Rumbaugh, James, Jacobson, Ivar y Booch, Grady. 1999. *El Lenguaje Unificado de Modelado.* s.l. : Addison Wesley, 1999.

Schumacher, Robin y Lentzm, Arjen. 2008. *MySQL: Dispelling the Myths.* 2008.

Sun Microsystems. 2008. *MySQL 5.0 Manual de Referencia.* 2008.

Tecnick.com. 2009. TCPDF, PHP class for generating PDF documents. [En línea] 2009. [Citado el: 6 de Mayo de 2009.] http://www.tecnick.com/public/code/cp_dpage.php?aiocp_dp=tcpdf.

The Eclipse Foundation. 2008. Eclipse. [En línea] 2008. [Citado el: 10 de Diciembre de 2008.] http://www.eclipse.org.

Wirth, Niklaus. Abril de 1971. *Program development by stepwise refinement.* s.l. : Communications of the ACM, Abril de 1971.

Wolf, Alexander. Enero de 1997. *Succeedings of the Second International Software Architecture Workshop.* s.l. : ACM SIGSOFT Software Engineering Notes, Enero de 1997. págs. 42-56.

Wolf, Dewayne, Perry, E. y L., Alexander. Octubre de 1992. *Foundations for the study of software architecture.* s.l. : ACM SIGSOFT Software Engineering Notes, Octubre de 1992.

Zend Company. 2009. ZendFramework.com. [En línea] 2009. [Citado el: 12 de Febrero de 2009.] http://framework.zend.com/.

Zend Technologies Ltd. 2008. Zend: The PHP Company. [En línea] 2008. [Citado el: 10 de Diciembre de 2008.] http://www.zend.com.

Anexos

		Purpose		
		Creational	**Structural**	**Behavioral**
Scope	**Class**	Factory Method (107)	Adapter (139)	Interpreter (243) Template Method (325)
	Object	Abstract Factory (87) Builder (97) Prototype (117) Singleton (127)	Adapter (139) Bridge (151) Composite (163) Decorator (175) Facade (185) Proxy (207)	Chain of Responsibility (223) Command (233) Iterator (257) Mediator (273) Memento (283) Flyweight (195) Observer (293) State (305) Strategy (315) Visitor (331)

Anexo 1 Clasificación de los patrones de diseño.

Atributo de Calidad	Descripción
Disponibilidad *(Availability)*	Es la medida de disponibilidad del sistema para el uso (Barbacci et al., 1995).
Confidencialidad *(Confidentiality)*	Es la ausencia de acceso no autorizado a la información (Barbacci et al., 1995).
Funcionalidad *(Functionality)*	Habilidad del sistema para realizar el trabajo para el cual fue concebido (Kazman et al., 2001).
Desempeño *(Performance)*	Es el grado en el cual un sistema o componente cumple con sus funciones designadas, dentro de ciertas restricciones dadas, como velocidad, exactitud o uso de memoria. (IEEE 610.12). Según Smith (1993), el desempeño de un sistema se refiere a aspectos temporales del comportamiento del mismo. Se refiere a capacidad de respuesta, ya sea el tiempo requerido para responder a aspectos específicos o el número de eventos procesados en un intervalo de tiempo. Según Bass et al. (1998), se refiere además a la cantidad de comunicación e interacción existente entre los componentes del sistema.
Confiabilidad *(Reliability)*	Es la medida de la habilidad de un sistema a mantenerse operativo a lo largo del tiempo (Barbacci et al., 1995).
Seguridad externa *(Safety)*	Ausencia de consecuencias catastróficas en el ambiente. Es la medida de ausencia de errores que generan pérdidas de información (Barbacci et al., 1995).
Seguridad interna *(Security)*	Es la medida de la habilidad del sistema para resistir a intentos de uso no autorizados y negación del servicio, mientras se sirve a usuarios legítimos (Kazman et al., 2001).

Anexo 2 Descripción de atributos de calidad observables vía ejecución. Fuente: (Camacho, et al., 2004 p. 9)

Atributo de Calidad	Descripción
Configurabilidad *(Configurability)*	Posibilidad que se otorga a un usuario experto a realizar ciertos cambios al sistema (Bosch et al., 1999).
Integrabilidad *(Integrability)*	Es la medida en que trabajan correctamente componentes del sistema que fueron desarrollados separadamente al ser integrados. (Bass et al. 1998)
Integridad *(Integrity)*	Es la ausencia de alteraciones inapropiadas de la información (Barbacci et al., 1995).
Interoperabilidad *(Interoperability)*	Es la medida de la habilidad de que un grupo de partes del sistema trabajen con otro sistema. Es un tipo especial de *integrabilidad* (Bass et al. 1998)
Modificabilidad *(Modifiability)*	Es la habilidad de realizar cambios futuros al sistema. (Bosch et al. 1999).
Mantenibilidad *(Maintainability)*	Es la capacidad de someter a un sistema a reparaciones y evolución (Barbacci et al., 1995). Capacidad de modificar el sistema de manera rápida y a bajo costo (Bosch et al. 1999).
Portabilidad *(Portability)*	Es la habilidad del sistema para ser ejecutado en diferentes ambientes de computación. Estos ambientes pueden ser hardware, software o una combinación de los dos (Kazman et al., 2001).
Reusabilidad *(Reusability)*	Es la capacidad de diseñar un sistema de forma tal que su estructura o parte de sus componentes puedan ser reutilizados en futuras aplicaciones (Bass et al. 1998).
Escalabilidad *(Scalability)*	Es el grado con el que se pueden ampliar el diseño arquitectónico, de datos o procedimental (Pressman, 2002).
Capacidad de Prueba *(Testability)*	Es la medida de la facilidad con la que el software, al ser sometido a una serie de pruebas, puede demostrar sus fallas. Es la probabilidad de que, asumiendo que tiene al menos una falla, el software fallará en su próxima ejecución de prueba (Bass et al. 1998).

Anexo 3 Descripción de atributos de calidad no observables vía ejecución

Característica	Subcaracterística	Elementos de tipo arquitectónico
Funcionalidad	Adecuación	Refinamiento de los diagramas de secuencia
	Exactitud	Identificación de los componentes con las funciones responsables de los cálculos
	Interoperabilidad	Identificación de conectores de comunicación con sistemas externos
	Seguridad	Mecanismos o dispositivos que realizan explícitamente la tarea
Confiabilidad	Tolerancia a fallas	Existencia de mecanismos o dispositivos de software para manejar excepciones
	Recuperabilidad	Existencia de mecanismos o dispositivos de software para reestablecer el nivel de desempeño y recuperar datos
Eficiencia	Desempeño	Componentes involucrados en un flujo de ejecución para una funcionalidad
	Utilización de recursos	Relación de los componentes en términos de espacio y tiempo
Mantenibilidad	Acoplamiento	Interacciones entre componentes
	Modularidad	Número de componentes que dependen de un componente
Portabilidad	Adaptabilidad	Presencia de mecanismos de adaptación
	Instalabilidad	Presencia de mecanismos de instalación
	Coexistencia	Presencia de mecanismos que faciliten la coexistencia
	Reemplazabilidad	Lista de componentes reemplazables para cada componente

Anexo 4 Atributos de calidad planteados por Losavio et al. (2003), que poseen subcaracterísticas asociadas con elementos de tipo arquitectónico.

Tablas	Tamaño Inicial	Tamaño Total
nCSS	2.1K	10.3K
nformat	0.1K	0.3K
nOptionrw	6.0K	10.0K
nRol	2.1K	205.7K
nscheduletype	0.3K	0.3K
nsubscriptiontype	0.2K	0.2K
nTargetDB	0.2K	1.1K
nTask	22.5K	225.2K
tbCategory	2.1K	2.0M
tbCategoryReport	0.0K	781.3K
tbDataSource	0.5K	117.7K
tbModel	2.3K	2.2M
tbReport	8.3K	80.9M
tbReportDataSource	0.0K	1.9M
tbschedule	0.1K	1015.6K
tbSubscription	4.8K	46.6M
tbTemplate	2.3K	22.2M
tbUser	2.1K	514.2K
tbUserCategory	0.0K	195.3K
tbUserReport	0.0K	1.9M
tbUserRol	0.0K	19.5K
Total	55.7K	160.7M

Anexo 5 Tamaño en memoria de las tablas en la base de datos

Fase 1: Presentación	
1. Presentación del ATAM	El líder de evaluación describe el método a los participantes, trata de establecer las expectativas y responde las preguntas propuestas.
2. Presentación de las metas del negocio	Se realiza la descripción de las metas del negocio que motivan el esfuerzo, y aclara que se persiguen objetivos de tipo arquitectónico.
3. Presentación de la arquitectura	El arquitecto describe la arquitectura, enfocándose en cómo ésta cumple con los objetivos del negocio.
Fase 2: Investigación y análisis	
4. Identificación de los enfoques arquitectónicos	Estos elementos son detectados, pero no analizados.
5. Generación del *Utility Tree*	Se elicitan los atributos de calidad que engloban la "utilidad" del sistema (desempeño, disponibilidad, seguridad, modificabilidad, usabilidad, etc.), especificados en forma de escenarios. Se anotan los estímulos y respuestas, así como se establece la prioridad entre ellos.
6. Análisis de los enfoques arquitectónicos	Con base en los resultados del establecimiento de prioridades del paso anterior, se analizan los elementos del paso 4. En este paso se identifican riesgos arquitectónicos, puntos de sensibilidad y puntos de balance.
Fase 3: Pruebas	
7. Lluvia de ideas y establecimiento de prioridad de escenarios.	Con la colaboración de todos los involucrados, se complementa el conjunto de escenarios.
8. Análisis de los enfoques arquitectónicos	Este paso repite las actividades del paso 6, haciendo uso de los resultados del paso 7. Los escenarios son considerados como casos de prueba para confirmar el análisis realizado hasta el momento.
Fase 4: Reporte	
9. Presentación de los resultados	Basado en la información recolectada a lo largo de la evaluación del ATAM, se presentan los hallazgos a los participantes.

Anexo 6 Pasos del método de evaluación ATAM

Atributo de calidad	Refinamiento del atributo	Escenarios
Desempeño	**Disponibilidad**	E20. Debido a fallas en el servidor web, el sistema se encuentra no disponible, el sistema detecta el error y es capaz de recuperarse y ponerse online(B,M)
		E22. El usuario está diseñando un reporte y la base de datos fuente no está disponible en ese momento. El sistema permite la creación del reporte mediante el modelo semántico. (A,M)
		E23. El usuario requiere la creación de un modelo semántico, mientras el servidor de base de datos fuente no está disponible. El sistema detecta la falla y notifica al usuario del problema (M,B)
		E6. El sistema está fuera de servicio porque se están realizando tareas de mantenimiento o de configuración, 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)
		E7. Detectar fallas en la base de datos y recuperarlo en menos de 2 minutos.(B,M)
	Seguridad	E8. No se ha recibido ninguna petición de una sesión de usuario por más de 10 min. El sistema toma esta sesión como potencialmente insegura e invalida las credenciales del usuario redireccionándolo a la página de logueo.(M,B)
		E9. Un usuario no autorizado intenta acceder a un reporte sin tener los permisos, el sistema no muestra el reporte(A,B)

Anexo 7 Escenarios categorizados y priorizados

Modificabilidad	**Mantenibilidad**	E10. El equipo de soporte detecta deficiencias en el funcionamiento, arregla el error y lo comunica en menos de 1 semana (M, B).
		E11. 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)
		E12. Se requiere la instalación de una nueva versión de la base de datos en el mínimo tiempo posible (M, M).
	Modularidad	E21. Un desarrollador desea cambiar la interfaz de usuario. Este cambio se ejecutará en el código en tiempo de diseño. Se requiere menos de 3 horas para hacer y probar el cambio y esto no tiene implicaciones en el comportamiento del sistema (A, M)
		E13. Incorporar nuevas funcionalidades a los módulos sin detener la explotación del sistema (M, A).
		E14. Desplegar un nuevo componente para el análisis de datos sin volver a desplegar la aplicación que está corriendo.(A, A)
Escalabilidad	**Crecimiento de la arquitectura**	E24. Se desea optimizar el rendimiento de la aplicación distribuyéndola en diferentes servidores cuando se despliegue la misma. Se ubica la BD en un servidor, la aplicación web en otro y las fuentes de datos de los clientes distribuidas (A, M).
		E16. Integración con la suite de herramientas para el Análisis Inteligente de Datos (M, M).
		E17. Agregar Oracle como fuente de datos para la generación de reportes (A, A).
	Interoperabilidad	E18. Se requiere que las aplicaciones de la empresa utilicen el generador de reportes para sacar información de los procesos de negocio (A,A)
Configurabilidad		E19. Se requiere cambiar la URL de entrada al sistema, con el sistema en explotación, se configura solo un fichero de configuración (A, A).

Anexo 8 Escenarios categorizados y priorizados (continuación)

Leyenda

Alta (A) Media (M) Baja (B)

Operaciones	Descripción	Parámetros
loadFunction	Se utiliza cuando un reporte tiene como fuente de datos una función, permite cargar la función.	*@param {} SqlfunctionName* Función Sql donde se recupera la información (sql store). *@param {} SqlSchema* Esquema donde está la Función o sql store *@param {} SqlField* Parámetro de la entidad *@param {} ValueField* Valor del campo de la Función con el que se compara. *@param {} IsNull* Determina si el parámetro es null o no.
setFormat	Setea el formato en el que se va a visualizar un reporte, por defecto es HTML.	*@param {} Format* Define el formato en que se desea visualizar el reporte [HTML,PDF,EXCEL]
addParameter	Permite añadirle los parámetros a la función.	*@param {} SqlFunctionName* Función Sql donde se recupera la información (sql store). *@param {} ParameterName* Parámetro de la entidad. *@param {} Schema* Esquema donde está la Función o sql store. *@param {} Value* Valor del campo de la Función con el que se compara. *@param {} IsNull* Determina si el parámetro es null o no.
addCondition	Para filtrar el reporte	*@param {} OpenParentesis* Paréntisis Abierto si se desea encapsular la condición. *@param {} RelacionalOperator* Operador relacional *@param {} Value* Valor que se desea comparar. *@param {} ClosedParentesis* Paréntisis Cerrado si se abrieron para la condición. *@param {} LogicalOperator* Operador lógico. *@param {} Schema* Esquema donde se encuentra la entidad o sql store del reporte. *@param {} Table* Sql store del reporte. *@param {} Field* Campo que se desea filtrar. *@param {} IsNull* Determina si el valor del campo puede ser null o no.

addReport	Setea la descripción y nombre del reporte que desea se muestre en el navegador.	*@param {} title* Titulo del reporte *@param {} description* Descripción *@param {}* url Url de recuperación (Lo proporciona la función PrepareUrl())
PrepareUrl	Prepara la URL con los datos necesarios para filtrar un reporte.	
getReportById	Obtiene el reporte dado un identificador.	*@param {} id* Id del reporte *@param {} title* Título del reporte *@param {} description* Descripción del reporte

Anexo 9 Descripción de los principales métodos para la integración dinámica por URL

Tuplas	Conexiones	Muestras	Respuesta promedio	% Error	Rendimiento	Kb/sec	% Procesador Max	RAM
1000	1	1	1535	0.0	0.6514	19.6317	35	invariable
1000	2	2	2371	0.0	0.6985	21.0511	98	+10 MB
1000	3	3	3277	0.0	0.7088	21.3620	100	+12 MB
1000	5	5	5886	0.0	0.7176	21.6267	100	+26 MB
1000	10	10	12869	0.0	0.7243	21.8272	100	+74 MB
1000	20	20	19358	0.0	0.7277	21.9313	100	+128 MB

Anexo 10 Resultados de la simulación perfil Tiempo de respuesta.

Tuplas	Conexiones	Muestras	Respuesta promedio	% Error	Rendimiento	Kb/sec	% Procesador Max	RAM
500	1	10	876	0.0	1.1396	17.0873	47	invariable
1000	1	10	1523	0.0	0.6556	19.7566	48	invariable
5000	1	10	6828	0.0	0.1464	22.7744	50	+15 MB
10000	1	10	13460	0.0	0.0742	23.5168	50	+15 MB
20000	1	5	26777	0.0	0.0373	24.2243	50	+50 MB
50000	1	5	66662	0.0	0.0150	25.3680	50	+50 MB
75000	1	5	98614	0.0	0.0101	25.7528	50	+50 MB

Anexo 11 Resultados de la simulación perfil Rendimiento.

UCI
Universidad de las Ciencias Informáticas

Centro de Tecnologías de Almacenamiento y Análisis de Datos

La Habana, 23 de marzo de 2009

Año del 50 Aniversario

ACTA DE ACEPTACIÓN

De una parte, el Centro de Tecnologías de Almacenamiento y Análisis de Datos, en lo sucesivo denominada CENTALAD, de la Universidad de las Ciencias Informáticas, representado en este acto por el Jefe del Proyecto Luis E. Saballo, y de otra parte, la _____ conocida de forma abreviada como ERP representada en este acto por Yoandry Morejón Borbón.

Primero: Que en cumplimiento de los acuerdos; han sido efectuadas las actividades que se describen, **Las Partes** DECLARAN:

CONSIDERANDO: Que se han efectuado las actividades siguientes:

1. Instalación y despliegue del sub-sistema generador de reportes dinámicos.
2. Personalización de la solución cuando ha sido requerido.
3. Capacitación del uso de la misma.

CONSIDERANDO: Que las actividades realizadas han sido desarrolladas con la calidad requerida y bajo las condiciones pactadas y aprobadas por **Las Partes**.

CONSIDERANDO: Que las actividades que se han ejecutado cumplen con los requerimientos establecidos.

CONSIDERANDO: Que **CENTALAD** ha entregado la documentación que avala la ejecución de este acto a Yoandry Morejón Borbón

I

CONSIDERANDO: Que el costo total de realización del módulo ha sido de _____.

POR TANTO: Las Partes acuerdan formalizar mediante la presente Acta, las actividades que han sido ejecutadas en esta fecha.

Y para que así conste, se extiende la presenta **Acta** en dos (2) ejemplares, rubricados por **Las Partes**.

Fecha y Firmas de las partes

Dr. C Pedro Yobanis Piñero Pérez
Centro de Tecnología de Almacenamiento y Análisis de Datos.

Anexo 12 Acta de aceptación del producto