Universidad de las Ciencias Informáticas

Facultad 3

Título: "Arquitectura de Software para el proyecto Sistema de Gestión Fiscal"

Trabajo de Diploma para optar por el título de

Ingeniero en Ciencias Informáticas

AUTOR: Yaisel Fernández Suárez

TUTOR: Ing. Alain Hernández López

Ciudad de la Habana, Junio 2009

DECLARACIÓN DE AUTORÍA

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

Para que así conste firmo la presente a los ____ días del mes de Junio del año 2009

_____	_____
Yaisel Fernández Suárez	Ing. Alain Hernández López

AGRADECIMIENTOS

Antes que todo a mi mama (tata), a mi papa y a mi hermanitos, por ser mi razón de ser.
A Yamile y al Kiki por darme buenos ejemplos, y ser incondicionales conmigo.
A la China, al Wilfre y al Yude por facilitarme la información siempre que necesite para mi tesis.
A mis compañeros y amigos del proyecto, por siempre estar dispuestos a aclararme una duda.
A mis compañeros de apartamento, por preocuparse por mis cortes de tesis.
A los compañeros de mi grupo 3505, que me ayudaron con dudas o con consejos.
Para cerrar con broche de oro, a Alain por ser mi guía y mí maestro, a través de quien aprendí mucho, sobre todo por que me ayudo a superarme, siempre exigiendo el máximo de mí.

DEDICATORIA

El resultado de lo que he llegado a ser, se lo debo a mi mama y a mi papa que con su sacrificio han logrado que yo este hoy graduado, ustedes que en los malos momentos siempre me apoyaron aunque estuviera equivocado, simplemente los llevo en el corazón, las palabras no podrían describir mis sentimientos hacia ustedes.

RESUMEN

Con el fin de lograr la automatización de los procesos fundamentales en los que interviene la Fiscalía General de la República, a fin de garantizar con mayor economía y profesionalidad, el control y la preservación de la legalidad, sobre la base de la vigilancia del estricto cumplimiento de la Constitución, las leyes y demás disposiciones legales, por los organismos del Estado, las entidades económicas y sociales, y por los ciudadanos, surge la necesidad de desarrollar este Sistema de Gestión Fiscal que constituye una completa ayuda a la toma de decisiones de los fiscales el cual viabiliza y humaniza el trabajo de los mismos, permitiendo reducir al máximo el margen de errores.

Por las características propias del sistema, fue necesario diseñar una arquitectura que respondiera a cualidades como la seguridad, la escalabilidad, la mantenibilidad, la flexibilidad, entre otras. Algunas de las contribuciones más importantes para alcanzar estas cualidades arquitectónicas fueron el estudio del estado del arte de la arquitectura de software, la selección de estilos, patrones y tecnologías web, el desarrollo de los artefactos propuestos por la metodología utilizada, y la validación de los principales escenarios arquitectónicos.

Como parte de la organización del proceso de desarrollo se construyeron una serie de de vistas arquitectónicas, que ayudaron a la compresión del sistema por los implicados en su construcción. El análisis de los resultados obtenidos corroboró que el sistema cumple en gran medida con los objetivos propuestos y responde a las restricciones impuestas. Como resultado del trabajo se obtuvo el documento de arquitectura de software, siendo uno de los artefactos de vital importancia para lograr las características deseadas del Sistema de Gestión Fiscal.

PALABRAS CLAVES: Arquitectura de software, estilos y patrones arquitectónicos, tecnologías informáticas, atributos de calidad de software.

TABLA DE CONTENIDO

INTRODUCCIÓN

Un problema que enfrentan hoy día las organizaciones que se dedican al desarrollo de software, está relacionado con la constante evolución de las tecnologías en el campo de la informática, esta incertidumbre tecnológica se refleja en el proceso de desarrollo, ya que se requieren sistemas cada vez más flexibles, heterogéneos y de fácil integración, provocando un aumento de la complejidad del diseño del software. Sumado que la mayoría de las veces el cliente no tiene conocimiento en el campo de la informática, por lo que es difícil que pueda aportar su visión del sistema. Estos obstáculos en el desarrollo de software son aminorados por una disciplina, aun joven pero con objetivos bien definidos, la Arquitectura de Software, tiene participación activa en el ciclo de vida del software, y propone un diseño de alto nivel que es la base para entender los sistemas más complejos.

El tema de la arquitectura se ha planteado en las bibliografías estudiadas y en las diferentes investigaciones realizadas, hace unos años no era común que los proyectos tuvieran una arquitectura documentada, pero por la necesidad de la organización del desarrollo, la reutilización de diseño y códigos, y el mejor entendimiento del software, se hace necesario la realización de una descripción de la arquitectura a la hora de desarrollar un software.

La economía y el logro de altos índices de desarrollo social, están relacionados estrechamente con la gestión fiscal llevada a cabo desde los mismos inicios de la revolución cubana. La jerarquización de los programas sociales en el presupuesto tales como: la educación, la salud pública, la seguridad social, el deporte, entre otros aspectos, han coadyuvado a elevar sistemáticamente el nivel de vida de nuestra población lo que ha provocado un proceso amplio de descentralización del aparato social y por consiguiente de la gestión fiscal.

A pesar de los avances tecnológicos en el proceso de informatización de la gestión fiscal en países desarrollados, los países de América Latina no se encuentran en la misma situación, estos países no cuentan con sistemas informatizados tan desarrollados que agilicen el proceso de gestión fiscal.

En nuestro país el proceso de gestión fiscal suele ser lento, y resulta agotador para los fiscales, quienes atienden más de un caso judicial a la vez, procesando altos volúmenes de información y generando gran cantidad de documentación que demanda altos gastos y esfuerzos. El ciclo de cada tarea consume papel en exceso que puede extraviarse o causar demora en la resolución de los casos; problema que no ha sido satisfecho por el actual sistema de gestión con que cuenta la Fiscalía General de la República (FGR), el cual es una aplicación de escritorio implementada en *Delphi* que no complementa todas las funcionalidades que se necesitan.

A raíz de esta problemática, la FGR en acuerdo con la Universidad de las Ciencias Informáticas (UCI) deciden desarrollar un nuevo Sistema de Gestión Fiscal (SGF) cuyo propósito principal sería la automatización de los procesos en los que intervienen los fiscales y la mejora de la calidad en los servicios brindados a la población. Para cumplir con las expectativas del cliente se hace necesario trazar una arquitectura que responda a las cualidades que se desean obtener del nuevo sistema y centre el proceso de desarrollo en torno a ellas.

La situación anteriormente expuesta conlleva al siguiente **problema**:
¿Qué elementos debe definir la línea base de la arquitectura de software del Sistema de Gestión Fiscal que contribuya a lograr la seguridad, escalabilidad, mantenibilidad y flexibilidad del producto final?

El **objeto de estudio** es el Proceso de Desarrollo de Software, y el **campo de acción** la arquitectura de software del Sistema de Gestión Fiscal.

Se trazó como **objetivo** definir una arquitectura de software que contribuya a lograr la seguridad, escalabilidad, mantenibilidad y flexibilidad del Sistema de Gestión Fiscal.

Para darle solución al objetivo propuesto se plantearon las siguientes **tareas investigativas**:

1. Realizar el estudio del estado del arte de la arquitectura de software.
2. Desarrollar los artefactos correspondientes al rol de arquitecto de software en el subsistema Gestión de Cuadros y Personal de Apoyo del proyecto Sistema de Gestión Fiscal.
3. Realizar evaluación de los resultados obtenidos con el desarrollo de los artefactos y llegar a conclusiones concretas al respecto.

En la investigación se planteó la siguiente **idea a defender**: Si se definen correctamente los elementos de la arquitectura de software se contribuirá al logro de un sistema seguro, escalable, mantenible y flexible.

Para el desarrollo de la investigación se utilizaron los siguientes **métodos científicos**:

Métodos Teóricos:

- Analítico – Sintético: Permitió realizar un análisis de la arquitectura de software para extraer los elementos más importantes y arribar a conclusiones en torno al problema.
- Inductivo – Deductivo: Permitió decidir cuales módulos desarrollar primero de acuerdo a su importancia inmediata para la FGR.
- Histórico – Lógico: Permitió organizar los principales sucesos en la historia de la Arquitectura de Software.

Métodos Empíricos:

- Observación: Permitió chequear el comportamiento de las principales características del sistema en entornos de producción.

El trabajo está estructurado en tres **capítulos** de la siguiente forma**:**

- Capítulo 1: Fundamentación teórica: Estudio del estado del arte de la Arquitectura de Software, principales conceptos de la disciplina, estudio de las tecnologías disponibles y la elaboración de una propuesta de arquitectura para la solución.

- Capítulo 2: Diseño Arquitectónico: Desarrollo los artefactos correspondientes al rol de arquitecto de software en el subsistema Gestión de Cuadros y Personal de Apoyo del proyecto Sistema de Gestión Fiscal.

- Capítulo 3: Análisis y validación de la Arquitectura: Validar el cumplimiento de las cualidades arquitectónicas más deseadas por los principales escenarios del sistema.

CAPÍTULO 1

1. FUNDAMENTACIÓN TEÓRICA

1.1. Introducción

En este capítulo se hace un estudio riguroso del los antecedentes de la Arquitectura de Software, su evolución y las tendencias actuales, se analizan los puntos de vista en cuanto a estilos y patrones para diseñar la arquitectura correcta. Teniendo en cuenta que una buena arquitectura condiciona una serie de atributos de calidad, los mismos se describirán brevemente. Por su importancia para la arquitectura se describen algunas características de las metodologías de desarrollo y las tecnologías más utilizadas para el desarrollo de aplicaciones web. Se define una propuesta de arquitectura basada en los requerimientos del cliente y los conocimientos aportados del estudio de las mejores prácticas para el desarrollo de sistemas de gestión.

1.2. Arquitectura de Software

Cuando se escucha la palabra arquitectura lo primero que se interpreta es el arte de construir, de edificar. Hace poco más de una década emergió una disciplina, previsto por las mentes más visionarias de entonces, fue un impacto en el diseño de sistemas de gran complejidad, la arquitectura de software, se adueñó de innumerables definiciones, pero su único objetivo es lograr una abstracción del sistema antes de comenzar a desarrollar, buscando un mejor entendimiento del mismo.

La Arquitectura de Software surge en 1992 en un artículo de *Dewayne Perry y Alexander Wolf* titulado "*Fundations for the Study of Software Architecture*" donde ellos plantean exactamente "La década de 1990, creemos, será la década de la Arquitectura de Software. Usamos el término arquitectura en contraste con diseño… Es tiempo de re-examinar el papel de la Arquitectura de Software en el contexto más amplio del proceso de software y de su administración" (*Fundations for the study of software*

architecture, 1992). En este artículo se deja claro el fundamento de la Arquitectura de Software construido sobre la base de diferentes disciplinas arquitectónicas bien definidas.

La década de 1990, es una época de significativos aportes para la Arquitectura de Software, donde surgen los patrones de diseño: GOF (Gamma, et al., 1995) como se les conoce popularmente; posteriormente en 1996 llegan los patrones arquitectónicos, en una serie de volúmenes que se conocen popularmente como POSA (*Patern Oriented Software Architecture*).

Muchos son los aportes en la definición de la Arquitectura de Software existentes en la literatura. Sin embargo al hacer un análisis de cada uno de los conceptos, resulta la existencia de ideas comunes entre los mismos, sin observarse planteamientos contradictorios, sino más bien complementarios.

La definición por la se rigen la mayoría de las instituciones fue dada por la IEEE:

> "*La Arquitectura del Software es la organización fundamental de un sistema formado por sus componentes*[1]*, las relaciones entre ellos y el contexto en el que se implantarán, y los principios que orientan su diseño y evolución".*
> [Std 1471-2000]:

1.2.1. Componentes, conectores y relaciones

Se entiende por componentes los bloques de construcción que conforman las partes de un sistema de software. A nivel de lenguajes de programación, pueden ser representados como módulos, clases, objetos o un conjunto de funciones relacionadas (Buschmann, et al., 1996). La noción de componente puede llegar a ser muy amplia, el término puede ser utilizado para especificar un conjunto de componentes. Se distinguen tres tipos de componentes, denominados también elementos, como:

- Elementos de Datos: contienen la información que será transformada.
- Elementos de Proceso: transforman los elementos de datos.

[1] Servidores, clientes, bases de datos, filtros, capas en un sistema jerárquico, etc.

- Elementos de Conexión: llamados también conectores, que bien pueden ser elementos de datos o de proceso, y mantienen unidas las diferentes piezas de la arquitectura.

(*Fundations for the study of software architecture*, 1992)

Una relación es la conexión entre los componentes. Puede definirse también como una abstracción de la forma en que los componentes interactúan en el sistema a través de los elementos de conexión. Es importante distinguir que una relación se concreta mediante conectores (Buschmann, et al., 1996).

Además de los componentes y conectores, la arquitectura contempla las propiedades externamente visibles de los componentes del software, y las relaciones entre estos. En este sentido, las propiedades externamente visibles hacen referencia a servicios que los componentes proveen, características de desempeño, manejo de fallas, uso de recursos compartidos, etc. (Kazman, et al., 2001).

1.2.2. Importancia de la Arquitectura de Software:

- La Arquitectura del Software aporta una visión abstracta de alto nivel, posponiendo el detalle de cada uno de los módulos definidos en pasos posteriores del diseño.
- Realiza un análisis riguroso de consistencia de la base antes de elaborar el diseño y escribir el código.
- Permite la reutilización a grande y pequeña escala de arquitecturas de sistemas ya realizados con funcionalidades similares.
- Responde a los atributos de calidad del sistema como son la eficiencia, la modificabilidad, la tolerancia a fallos, la seguridad, etc.
- Permite el estudio de alguna propiedad específica del dominio.
- Define un nivel de diseño global del sistema, relevante en etapas posteriores donde la aplicación adquiere una mayor complejidad.
- En etapas tempranas de la concepción del sistema, es un pasaporte para el entendimiento del problema, entre clientes y equipo de desarrollo.

1.2.3. ¿De qué se ocupa?:

- Diseño preliminar o de alto nivel, algunos autores dirían: una actividad previa al diseño.
- Organización del sistema a grandes rasgos, incluyendo aspectos como la descripción y análisis de propiedades relativas a su estructura y control global, los protocolos de comunicación y sincronización utilizados, la distribución física del sistema y sus componentes, etc.
- Aspectos relacionados con el desarrollo del sistema, su evolución y adaptación al cambio; como la composición, reconfiguración, reutilización, escalabilidad, mantenibilidad, etc.

1.2.4. ¿De qué no se ocupa? :

- Diseño detallado del sistema.
- Diseño de algoritmos.
- Diseño de estructuras de datos.

1.3. El rol de Arquitecto de Software

"(...) es una persona, equipo u organización responsable por la arquitectura del sistema"
[IEEE Std. 1471-2000].

De la misma manera que ocurre con la Arquitectura de Software, existen múltiples definiciones sobre el rol de arquitecto. Comúnmente se ubica a los mismos en el contexto de una Organización, con las necesidades y requerimientos propios de esa Organización. Es difícil establecer un estereotipo de arquitecto que especifique cuáles son sus responsabilidades y habilidades necesarias dentro de un proyecto.

Todos los proyectos de envergadura deberían tener un arquitecto de software. No se debe confundir un arquitecto de software con un experto en programación o un "gurú tecnológico". Un arquitecto debe

poseer experiencia en el dominio del problema, tener liderazgo, ser comunicativo, orientado a resultados, el arquitecto de software es la fuerza técnica dirigente detrás del proyecto.

A grandes rasgos las responsabilidades de un arquitecto de software, son las que siguen:

- Definición de las vistas de la arquitectura de una aplicación (o sea, crear la arquitectura, ya que la arquitectura, en pocas palabras es un conjunto de vistas de alto nivel).
- Dar soporte técnico-tecnológico a desarrolladores, clientes y expertos en negocios.
- Conceptualizar y experimentar con distintos enfoques arquitectónicos.
- Crear documentos de modelos y componentes, y especificaciones de interfaces.
- Validar la arquitectura contra requerimientos y suposiciones.
- Responde sobre las inquietudes relacionadas con la selección de herramientas y ambientes de desarrollo.
- Gerencia las estrategias de identificación y mitigación de los riesgos asociados con la arquitectura.

1.4. Atributos de calidad del software

"la calidad de software es la medida en que el software posee una combinación deseada de atributos (confiabilidad, interoperabilidad, etc.)"

[IEEE Std 610.12-1990]

La arquitectura de software debe tener en cuenta los atributos de calidad en todas las etapas del desarrollo de un sistema, sobre todo en la toma de decisiones objetivas sobre acuerdos de diseño, una combinación adecuada de estos atributos y la medida en que son logrados contribuirá a lograr un sistema que se adapte a las metas de la arquitectura propuesta.

Tales atributos son requerimientos adicionales del sistema, hacen referencia a características que debe satisfacer, diferentes a los requerimientos funcionales. Estas características o atributos se conocen con el nombre de atributos de calidad (Kazman, et al., 2001).

Cada decisión incorporada en una arquitectura de software puede afectar potencialmente los atributos de calidad (Bass, et al., 2000).

A grandes rasgos, se establece una clasificación de los atributos de calidad en dos categorías (Bass, et al., 1998):

- Observables vía ejecución: aquellos atributos que determinan del comportamiento del sistema en tiempo de ejecución. La descripción de algunos de estos atributos se presenta en la tabla 1-1.
- No observables vía ejecución: aquellos atributos que se establecen durante el desarrollo del sistema. La descripción de algunos de estos atributos se presenta en la tabla 1-2.

Tabla 1-1. Descripción de atributos de calidad observables vía ejecución.

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, 1990) 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).

Es importante destacar que el cumplimiento de los atributos observables, no necesariamente implica que se satisfacen los atributos no observables vía ejecución. Por ejemplo, un sistema que satisface todos los requerimientos observables puede o no, ser imposible de modificar.

Tabla 1-2. Descripción de atributos de calidad no observables vía ejecución.

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).

Al respecto, se afirma que la calidad del sistema debe ser considerada en todas las fases del diseño, pero los atributos de calidad se manifiestan de maneras distintas a lo largo de estas fases. De esta forma, establecen que la arquitectura determina ciertos atributos de calidad del sistema, pero existen otros atributos que no dependen directamente de la misma (Bass, et al., 1998).

Encontrar un atributo de calidad puede tener efectos positivos o negativos sobre otros atributos que, de alguna manera, también se desean alcanzar (Bass, et al., 1998).

1.5. Estilos y Patrones

El arquitecto de software al trazar la arquitectura de un sistema de software siempre tiene en cuenta los estilos arquitectónicos y patrones que puede aplicar al diseño de dicha arquitectura. Los estilos y patrones se basan para su utilidad en la reutilización de las mejores prácticas de diseño que han demostrado ser eficientes en la solución de un problema específico. Además, su aplicación en el diseño de la arquitectura del sistema es determinante para la satisfacción de los requerimientos de calidad del mismo.

La diferencia entre estilos y patrones arquitectónicos no ha sido aclarada formalmente. Bengtsson (Bengtsson, 1999) plantea la existencia de dos grandes vertientes, que surgen de la discusión de los

términos. Unos Shaw y Garlan (Shaw, 1996), y Bass, Clements y Kazman(Bass, et al., April 11, 2003) utilizan indistintamente los términos estilo y patrón arquitectónico. Por otro lado, (Buschmann, et al., 1996) establece diferencias sutiles entre ambos conceptos.

La diferencia entre los estilos y patrones arquitectónicos está en el nivel de abstracción en el que nos movemos. Los estilos se aplican a un nivel muy alto de abstracción, en el cual no interesa saber cuál es la semántica de los elementos que estamos utilizando, sólo hablamos de filtros, tubos u objetos sin interesarnos por lo que están representado. En el caso de los patrones, se tiene en cuenta el significado de los filtros, tubos u objetos, tal como se ha visto en los patrones de descomposición en capas (en el que se habla de presentación, dominio de aplicación y de datos).

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 propuestos al formalizar el proyecto.

1.5.1. Estilos Arquitectónicos

"*Una arquitectura puede ajustarse a un estilo arquitectónico*".
(Eeles, 2006)

La mayoría de las arquitecturas se derivan de sistemas que comparten cuestiones importantes similares, esta similitud puede ser descrita como un estilo arquitectónico. Mary Shaw (Shaw, 1996), estima que los estilos se pueden comprender como clases de patrones, o tal vez más adecuadamente como lenguajes de patrones.

Los estilos arquitectónicos han gozado de gran popularidad en los últimos años, y por una buena razón: representan el conocimiento acumulado de muchos arquitectos experimentados y guían a los arquitectos menos experimentados en el diseño de sus arquitecturas. Emplean argumentos cualitativos para determinar cuándo y en qué condiciones se deben utilizar.

Una definición acertada para los estilos arquitectónicos es la siguiente: *"Un conjunto de reglas de diseño que identifica tipos de componentes, los conectores utilizados para su composición en sistemas o subsistemas, y las restricciones locales y globales que describen la forma en que se lleva a cabo la composición"*. Esta definición implica los siguientes conceptos:

Componentes: unidades de procesamiento y almacenamiento.

- Ejemplos: objetos, *threads*, procesos, bases de datos, etc.
- Propiedades: funcionalidad proporcionada, puertos (o interfaces), etc.

Conectores*:* mecanismos de interacción entre componentes.

- Ejemplos: *procedure calls*, *sockets*, memoria compartida, RPC[2], *pipes*, *sql-link*, *middleware* (ORB), event multicast, etc.
- Propiedades*:* roles, protocolos, cardinalidades de los componentes conectados, etc.

La cantidad de estilos y sub-estilos puede variar según el nivel de detalle requerido, en tanto no se defina un estándar de clasificación de los estilos, muchos van a ser los aportes al área. La tabla a continuación resume los estilos arquitectónicos más generalizados, los sub-estilos que comprenden, y una descripción a grandes rasgos de cada estilo, según (Bass, et al., April 11, 2003):

Tabla 1-3. Estilos arquitectónicos y sub-estilos asociados.

Estilo	Descripción	Sub-estilos Asociados	Atributos Asociados
Datos Centralizados	Se estima apropiada para sistemas que se basan en el acceso y actualización de datos en estructuras de almacenamiento.	❖ **Pizarras** ❖ **Repositorio**	Integrabilidad Escalabilidad Modificabilidad

[2] RPC (*Remote Procedure Call*): es un protocolo que permite a un programa de ordenador ejecutar código en una máquina remota.

Flujo de Datos	El dato ingresa en el sistema, y fluye entre los componentes, de uno en uno, hasta que se le asigne un destino final (salida o repositorio). Es apropiada para sistemas que implementan transformaciones de datos en pasos sucesivos.	❖ **Tubería y filtros** ❖ **Procesamiento por lotes**	Reusabilidad Modificabilidad Mantenibilidad
Máquinas Virtuales	Simulan alguna funcionalidad que no es nativa al hardware o software sobre el que está implementado. Caracterizado por la traducción de una instrucción en alguna otra.	❖ **Intérpretes** ❖ **Sistemas basados en reglas**	Portabilidad
Llamada y Retorno	El sistema se constituye de un programa principal que tiene el control del sistema y varios subprogramas que se comunican con éste mediante el uso de llamadas. Son los estilos más generalizados en sistemas a gran escala.	❖ **Orientado a objetos** ❖ **Sistema en capas** ❖ **Programa principal y subrutinas**	Modificabilidad Escalabilidad Desempeño
Componentes Independientes (*Peer-to-Peer*)	Consiste en un número de procesos u objetos independientes que se comunican a través de mensajes. Cada entidad puede enviar mensajes a otras entidades, pero no controlarlas directamente. Los mensajes pueden ser enviados a componentes nominados o propalados mediante *broadcast*.	❖ **Sistemas basados en eventos** ❖ **Procesos de comunicación**	Modificabilidad Escalabilidad

1.5.2. Patrones

"*Los procesos involucrados en el diseño de estructuras físicas no siempre son iguales, pero siempre es posible encontrar invariantes comunes, que definen los principios del diseño y la construcción*".
(Alexander, 1979)

En términos generales, un patrón se define como la descripción de un problema particular y recurrente de diseño, que aparece en contextos de diseño específico, y presenta un esquema genérico demostrado con éxito para su solución.

En pocas palabras un patrón es una solución a un problema en un contexto, donde:

- Contexto*:* son las situaciones recurrentes a las que es posible aplicar el patrón.
- Problema*:* es el conjunto de metas y restricciones que se dan en ese contexto.
- Solución*:* es el diseño a aplicar para conseguir las metas dentro de las restricciones.

¿Por qué usar patrones?:

- Producción de Software más resistente al cambio.
- Establece parejas problema-solución.
- Ayudan a especificar interfaces.
- Promueve la reutilización de partes del diseño.
- Uso de documentación estándar.

En el libro "*Pattern Oriented Software Architecture, Volume 1*" (Buschmann, et al., 1996), se define una clasificación de los tipos de patrones (como se muestra en la Figura a continuación). Los patrones en cada grupo varían respecto a su nivel de detalle y abstracción.

Figura 1-1. Patrones según el nivel de detalle, adaptado de POSA.

1.5.2.1. Patrones Arquitectónicos

Así mismo, se plantea que los patrones arquitectónicos expresan el esquema de organización estructural fundamental para sistemas de software. Provee un conjunto de subsistemas predefinidos, especifica sus responsabilidades e incluye reglas y pautas para la organización de las relaciones entre ellos. Propone que son plantillas para arquitecturas de software concretas, que especifican las propiedades estructurales de una aplicación, con amplitud de todo el sistema y tienen un impacto en la arquitectura de subsistemas (Buschmann, et al., 1996).

La selección de un patrón arquitectónico es, por lo tanto, una decisión fundamental de diseño en el desarrollo de un sistema de software. Cada patrón arquitectónico a incluirse en el sistema debe seleccionarse teniendo en cuenta que estos respondan a los requerimientos de calidad, un sistema complejo por lo general incluye más de un patrón arquitectónico, donde cada uno es responsable de una o varias tareas, especificando las reglas y pautas para la organización de las relaciones entre ellos.

1.5.2.1.1. Patrón Arquitectónico 3-Capas

Es una implementación del estilo en capas, que se entiende como una organización jerárquica tal que cada capa proporciona servicios a la capa inmediatamente superior y se sirve de las prestaciones que le brinda la inmediatamente inferior.

Una restricción de este patrón es que las capas se diseñan para que interactúen con las capas adyacentes, cuando esto no ocurre el estilo deja de ser puro, se pierde la flexibilidad del conjunto y complica el mantenimiento.

Acceso a datos:

- Almacenamiento, actualización y consulta de los datos del sistema.
- Se agrega soporte para una nueva base de datos en período corto.
- Puede estar en el mismo servidor de la lógica de negocio o distribuida.

Lógica de negocio:

- Componentes que modelan la lógica de negocio.
- Reciben acciones de la capa de presentación y utilizan la capa de datos para manipular información.
- Integración sencilla y eficaz con sistemas externos.

Presentación:

- Parte del sistema con la que interactúa el usuario.

Desventajas:

- No es aplicable a todos los sistemas.
- Bajo rendimiento cuando existe un alto nivel de acoplamiento.

Ventajas:

- Reutilización.
- Soporta fácilmente la evolución del sistema.
- Los cambios sólo afectan a las capas vecinas (Modificabilidad).
- Uso eficiente del hardware.
- Encapsulación de datos.

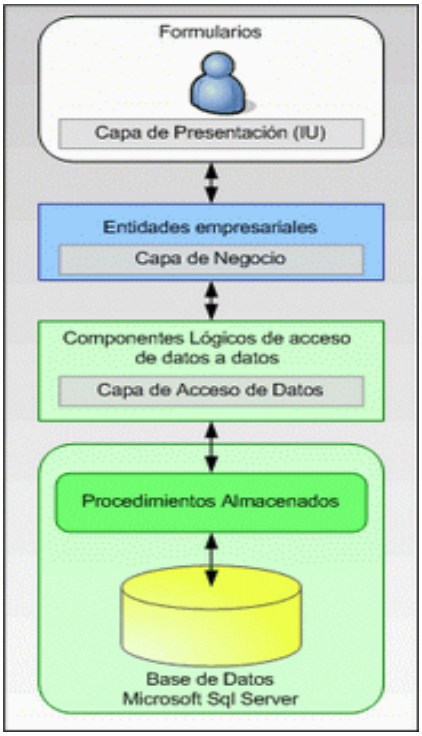


Figura 1-2. Arquitectura en capas

1.5.2.1.2. Patrón Arquitectónico MVC

La llegada de un evento del usuario arranca el controlador que se comunica con el modelo para pedir la actualización de los datos. Las vistas piden los datos al modelo y actualizan la representación de la información en cada una de las vistas.

En este patrón**,** el modelo, las vistas y los controladores se tratan como entidades separadas:

- Modelo: Es la información que manipula la aplicación, la representación de objetos reales, los maneja y controla sus transformaciones.
- Vista: Maneja la representación visual de los datos representados por el modelo, permite la interacción del usuario y proporciona la entrada de peticiones.
- Controlador: Gestiona los eventos del usuario, los recibe, los interpreta y decide que hacer con ellos.

Desventajas:

- Complejidad creciente.
- Si cambia la interfaz del modelo hay que cambiar todas las vistas y todos los controladores que hagan referencia a ella.
- Acceso ineficiente a los datos en la vista. Puede necesitar varias llamadas al modelo para actualizar todos los datos en la vista.

Ventajas:

- Múltiples vistas de un mismo modelo.
- Vistas sincronizadas.
- Flexibilidad para cambiar las vistas y los controladores.
- La aplicación puede soportar distintos tipos de interfaz de usuario.

1.5.2.2. Patrones de Diseño (GOF)

"*Un patrón de diseño es una solución probada que se puede aplicar con éxito a un determinado tipo de problemas que aparecen repetidamente en el desarrollo de software.*"

Los patrones de diseño son soluciones recurrentes a problemas que aparecen una y otra vez en el diseño de aplicaciones, tratan sobre el diseño e interacción de objetos, en tanto que proveen soluciones elegantes y reutilizables a los problemas comunes encontrados en la programación.

Características de los patrones de diseño:

- Permiten reutilizar soluciones a problemas comunes.
- Son flexibles para adaptarse a necesidades específicas.
- Su uso no se refleja en la estructura global del sistema.
- Indican resoluciones técnicas basadas en programación orientada a objetos.
- Permiten crear aplicaciones más flexibles y robustas.
- Se basan en el principio de programar para interfaces y no para una implementación.

En general un patrón de diseño tiene cuatro elementos esenciales:

- Nombre: describe el problema de diseño.
- Problema: describe cuando aplicar el patrón y su contexto.
- Solución: describe los elementos que componen el diseño, sus relaciones, responsabilidades y colaboración.
- Consecuencias: son los resultados, así como ventajas e inconvenientes de aplicar el patrón.

Los patrones de diseño varían en su granularidad y nivel de abstracción, los mismos están agrupados en familia de patrones relacionados. En la tabla a continuación se clasifica los patrones de diseño por dos criterios: uno es el propósito que puede ser de **creación**, de **estructura** o de **comportamiento**, y otro el ámbito que especifica si el patrón es aplicado a relaciones entre **clases** o a relaciones entre **objetos**.

Tabla 1-4. Patrones de diseño GOF (Gamma, et al., 1995).

		Propósito		
		De creación	Estructurales	De comportamiento
Ámbito	Clase	❖ Factory ❖ Method	❖ Adapter	❖ Interpreter ❖ Template Method
	Objeto	❖ Abstract ❖ Factory ❖ Factory ❖ Builder ❖ Prototype ❖ Singleton	❖ Adapter ❖ Bridge ❖ Composite ❖ Decorator ❖ Facade ❖ Proxy	❖ Chain of Responsibility ❖ Command ❖ Iterator ❖ Mediator ❖ Memento ❖ Flyweight ❖ Observer ❖ State ❖ Strategy ❖ Visitor
		Creación de Objetos	**Composición de objetos**	**Interacción de objetos**

1.5.2.3. Patrones para la Asignación de Responsabilidad (GRASP):

C. Larman propuso un conjunto de principios a los que les llamo GRASP (*General Responsability Assignment Software Patterns).* Larman eligió esta notación: GRASP[3], para sugerir la importancia de comprender estos principios como un paso clave para diseñar con éxito (Larman, 1999).

Los patrones GRASP describen los principios fundamentales de diseño de objetos para la asignación de responsabilidades a objetos, expresados en forma de patrones. A continuación se muestran cinco de los patrones *GRASP*, con una descripción breve de cada uno:

Tabla 1-5. Patrones para la Asignación de Responsabilidad (GRASP).

Patrones	Descripción	Ventajas
Experto	¿Cuál es el principio general del diseño orientado a objetos para la asignación de responsabilidades a las clases? Asignar una responsabilidad al experto en información (la clase que tiene la información necesaria para llevar a cabo la responsabilidad).	❖ Se conserva el encapsulamiento, esto soporta un **bajo acoplamiento**. ❖ Favorece una **alta cohesión**.
Creador	¿Quién debe ser el responsable de la creación de una nueva instancia de una clase? Asignar a la clase B la responsabilidad de crear una instancia de la clase A en uno de los siguientes casos: ❖ **B** agrega (compartidamente o no) a **A** ❖ **B** tiene los datos de inicialización de **A** ❖ **B** registra a **A** ❖ **B** utiliza 'estrechamente' a **A**	❖ Favorece un **bajo acoplamiento**.
Alta cohesión	¿Cómo mantener la complejidad dentro de límites manejables? Asignar a una clase responsabilidades moderadas en un	❖ Se facilita la comprensión y mantenimiento.

[3] GRASP: palabra del idioma ingles, que traducida al español quiere decir (comprender, entender,…).

	área funcional y de modo que colabore con otras para llevar a cabo las tareas.	❖ Favorece el bajo acoplamiento y la reutilización.
Bajo acoplamiento	¿Cómo dar soporte a una dependencia escasa y a una mayor reutilización? Asignar una responsabilidad para mantener bajo el acoplamiento, de manera que una clase con bajo acoplamiento no dependerá de muchas otras.	❖ Disminuye el impacto de los cambios. ❖ Se facilita el entendimiento y la reutilización.
Controlador	¿Quién debería encargarse de atender un evento del sistema? Asignar la responsabilidad del manejo de un mensaje de los eventos de un sistema a una clase que represente un controlador de fachada o de casos de uso.	❖ Posibilita la organización y claridad en el diseño. ❖ Favorece el bajo acoplamiento.

1.6. Lenguaje de modelado UML

Las herramientas que se han elaborado para representar la arquitectura de software son los Lenguajes de Descripción de Arquitecturas (ADL: *Architecture Description Language*). Sin embargo, son muy pocos quienes los utilizan como instrumento en el diseño arquitectónico de sus proyectos debido a que presentan diferentes problemas para su utilización, como:

- ❖ Requieren una extensa capacitación.
- ❖ No son amigables para presentar la arquitectura a personas ajenas a la construcción del software.
- ❖ No tienen herramientas ni metodologías de apoyo.
- ❖ Algunos se encuentran especializados solo en un tipo particular de sistemas.
- ❖ Sólo tienen en cuenta una sola estructura del sistema.

Las desventajas que se presentan en estos lenguajes, pueden ser superadas por un lenguaje de modelado como UML (*Unified Modeling Language*). A continuación se describen las principales características que sitúan UML como notación de referencia para representar la arquitectura de un sistema de software:

- Está soportado por un gran número de herramientas, las cuales están fácilmente disponibles.
- Está apoyado por metodologías de desarrollo, como RUP (*Rational Unified Process*).
- Es fácilmente comprensible tanto por las personas como por las máquinas.
- Permite modelar sistemas, desde los requisitos hasta los artefactos ejecutables, utilizando técnicas orientadas a objetos.

Esta unificación de los lenguajes de modelado fue promovida por la OMG[4] (*Object Management Group*) de tal manera que UML se convirtió en la notación estándar para la descripción de métodos software. Según su definición, UML es un lenguaje para visualizar, especificar, construir y documentar los artefactos (modelos) de un sistema que involucra una gran cantidad de software, desde una perspectiva orientada a objetos.

UML ha conseguido un rol importante en el proceso de desarrollo de software en la actualidad (Booch et al., 1999). La unificación del método de diseño y las notaciones, ha ampliado, entre otras cosas, el mercado de herramientas CASE que soportan el proceso de diseño de arquitecturas de software.

1.7. Metodologías de desarrollo

Todo desarrollo de software es riesgoso y difícil de controlar, pero si no se utiliza una metodología, se obtiene un desarrollo desorganizado y un nivel de integración muy pobre entre las partes involucradas.

Lo que se hace con los proyectos pequeños de dos o tres meses, es separar rápidamente el aplicativo en procesos, cada proceso en funciones, y por cada función determinar un tiempo aproximado de desarrollo. Cuando los proyectos que se van a desarrollar son de mayor envergadura, ahí si toma sentido, una metodología de desarrollo.

[4] OMG: organización de empresas informáticas que promueve el uso de técnicas orientadas a objeto en el desarrollo de software.

Teniendo en cuenta la importancia de las metodologías para desarrollar sistemas complejos y que están estrechamente ligadas a la arquitectura, a continuación se describirán tres de las más utilizadas en el proceso de desarrollo de software actual.

1.7.1. Proceso Unificado de Desarrollo (RUP)

El Proceso Unificado de Desarrollo (RUP) por sus siglas en ingles, es una metodología para el desarrollo de software orientado a objetos. Es un proceso de desarrollo de software, definido como un conjunto de actividades necesarias para transformar los requisitos de un usuario en un sistema software. Sin embargo, el proceso unificado es más que un proceso de trabajo, es un marco de trabajo genérico que puede especializarse para una gran variedad de sistemas software, para diferentes áreas de aplicación, diferentes tipos de organizaciones y diferentes niveles de aptitud. Está constituido por 5 flujos de trabajo fundamentales: requisitos, análisis, diseño, implementación y prueba, los cuales tienen lugar sobre 4 etapas o fases: inicio, elaboración, construcción y transición. Esta metodología es adaptable para proyectos a largo plazo y establece refinamientos sucesivos de una arquitectura ejecutable.

Principales características de RUP:

- Dirigido por casos de uso: Esto significa que el proceso de desarrollo sigue una trayectoria que avanza a través de los flujos de trabajo generados por los casos de uso. Los casos de uso se especifican y diseñan al principio de cada iteración, y son la fuente a partir de la cual los ingenieros de prueba construyen sus casos de prueba. Estos describen la funcionalidad total del sistema.

- Centrado en la arquitectura: Los casos de uso guían a la arquitectura del sistema y esta influye en la selección de los casos de uso. La arquitectura involucra los elementos más significativos del sistema y está influenciada entre otros por las plataformas de software, sistemas operativos, sistemas de gestión de bases de datos, además de otros como sistemas heredados y requerimientos no funcionales.

- Iterativo e incremental: RUP divide el proceso en cuatro fases, dentro de las cuales se realizan varias iteraciones en número variable según el proyecto y las cuales se definen según el nivel de madurez que alcanzan los productos que se van obteniendo con cada actividad ejecutada. La terminación de cada fase ocurre en el hito correspondiente a cada una, donde se evalúa que se hayan cumplido los objetivos de la fase en cuestión.

RUP está basado en componentes y utiliza UML para visualizar, especificar y documentar cada una de las partes que comprende el desarrollo de software.

1.7.2. Programación Extrema (XP)

La Programación Extrema, es una metodología ligera de desarrollo de software que se basa en la simplicidad, la comunicación y la reutilización del código desarrollado. La metodología consiste en una programación rápida o extrema, utilizada para proyectos de corto plazo.

La metodología se basa en:

- Pruebas unitarias: se basa en las pruebas realizadas a los principales procesos, de tal manera que se puede adelantar en algo hacia el futuro, se pueden hacer pruebas de las fallas que pudieran ocurrir. Es como si se obtuvieran los posibles errores.
- Refabricación: se basa en la reutilización de código, para lo cual se crean patrones o modelos estándares, siendo más flexible al cambio.
- Programación en pares: una particularidad de esta metodología es que propone la programación en pares, la cual consiste en que dos desarrolladores participen en un proyecto en una misma estación de trabajo. Cada miembro lleva a cabo la acción que el otro no está haciendo en ese momento. Es como el piloto y el copiloto: mientras uno conduce, el otro consulta el mapa.

El desarrollo bajo XP tiene características que lo distinguen claramente de otras metodologías:

- Los diseñadores y programadores se comunican efectivamente con el cliente y entre ellos mismos.
- Los diseños del software se mantienen sencillos y libres de complejidad o pretensiones excesivas.

- Se obtiene retroalimentación de usuarios y clientes desde el primer día gracias a las baterías de pruebas.
- El software es liberado en entregas frecuentes tan pronto como sea posible.
- Los cambios se implementan rápidamente tal y como fueron sugeridos.
- Las metas en características, tiempos y costos son reajustadas, permanentemente en función del avance real obtenido.

¿Qué es lo que propone XP? :

- Empieza en pequeño y añade funcionalidad con retroalimentación continua.
- El manejo del cambio se convierte en parte sustancial del proceso.
- El costo del cambio no depende de la fase o etapa.
- No introduce funcionalidades antes que sean necesarias.
- El cliente o el usuario se convierte en miembro del equipo. (Escribano, 2002)

1.7.3. Microsoft Solution Framework (MSF)

MSF tiene las siguientes características:

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

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

Visión general del MSF:

Fase 1 Estrategia y alcance:

- Elaboración y aprobación del documento de alcances del proyecto.
- Formación del equipo de trabajo y distribución de competencias y responsabilidades.
- Elaboración del plan de trabajo.
- Elaboración de la matriz de riesgos y plan de contingencia.

Fase 2 Planificación y prueba de concepto:

- Documento de planificación y diseño de arquitectura.
- Documento de plan de laboratorio (son las pruebas de conceptos).

Fase 3 Estabilización:

- Selección del entorno de pruebas piloto.
- Gestión de incidencias.
- Revisión de la documentación final de la arquitectura.
- Elaboración de plan de despliegue.
- Elaboración del plan de formación.

Fase 4 Despliegue:

- Registro de mejoras y sugerencias.
- Revisión de las guías y manuales de usuario.
- Entrega del proyecto y cierre del mismo.

Lo más importante antes de elegir la metodología que se usará para el desarrollo del software, es determinar el alcance que tendrá el software y luego determinar cuál es la que más se adapta.

1.8. Tecnología existente

En esta sección se describen aquellos componentes imprescindibles para el desarrollo y despliegue de la solución, posteriormente se hace un estudio de todas las herramientas utilizadas para seleccionar las que más se adaptan a la propuesta de arquitectura.

1.8.1. Lenguajes del lado del servidor

Los lenguajes del lado del servidor son aquellos, que se ejecutan en el servidor web. El código en el servidor se encarga de construir las páginas que serán enviadas de respuesta al cliente, estos lenguajes pueden acceder a bases de datos, ficheros en el servidor y recursos en la red. No son visibles por los clientes, ya que cuando se solicitan a través del servidor web lo que se genera es código HTML que es entendible y representable por un navegador web. Entre los principales lenguajes del lado del servidor encontraremos: PHP, las tecnologías *Active Server Pages* (ASP.Net) y *Java Server Pages* (JSP).

1.8.1.1. PHP5

PHP5 (acrónimo de "*PHP*: *Hypertext* Preprocessor") es un lenguaje de programación usado generalmente para la creación de aplicaciones Web. Lenguaje interpretado, multiplataforma y libre por lo que se ha convertido en uno de los más difundidos en Internet. Puede usarse para la creación de otro tipo de programas incluyendo aplicaciones con interfaz gráfica usando la biblioteca GTK+.

Desventajas:

- No está totalmente desarrollado en lo referente a la programación orientada a objetos.
- No brinda gran rendimiento en aplicación de complejidad como tratamiento de imágenes o de cálculos intensivos.

Ventajas:

- Es un lenguaje multiplataforma.
- Es libre, por lo que se presenta como una alternativa de fácil acceso para todos.
- No requiere de grandes recursos del sistema para su funcionamiento.
- Capacidad de conexión con la mayoría de los manejadores de base de datos que se utilizan en la actualidad.
- Capacidad de expandir su potencial utilizando la enorme cantidad de módulos (llamados extensiones).
- Posee una amplia documentación en su página oficial, destacando que todas las funciones del lenguaje están explicadas y ejemplificadas en un único archivo de ayuda.
- Permite técnicas de Programación Orientada a Objetos, mejorado en su versión 5.

- Biblioteca nativa de funciones sumamente amplia e incluida.
- La sencillez de su sintaxis facilita su aprendizaje por lo que se ha difundido mucho en Internet.

1.8.1.2. ASP.NET

ASP.NET es un conjunto de tecnologías de desarrollo de aplicaciones web comercializado por Microsoft. Es usado por programadores para construir sitios web domésticos, aplicaciones web y servicios XML. Forma parte de la plataforma .NET de Microsoft y es la tecnología sucesora de la tecnología *Active Server Pages* (ASP). ASP.NET es la plataforma unificada de desarrollo web que proporciona a los desarrolladores los servicios necesarios para crear aplicaciones web empresariales.

1.8.1.3. Java Server Pages (JSP)

Es una tecnología para crear aplicaciones web. Es un desarrollo de la compañía Sun Microsystems, y su funcionamiento se basa en *scripts*, que utilizan una variante del lenguaje Java. Permite a los programadores generar contenido dinámico para la web, en forma de documentos HTML y/o XML, e incrustar en el contenido estático de las páginas web, código Java y comandos predefinidos. Con JSP se pueden crear aplicaciones que se ejecuten en variados servidores web, de múltiples plataformas.

1.8.2. Servidores web

1.8.2.1. Apache

En lo que respecta a las tecnologías por parte del servidor sobresale Apache. El servidor HTTP Apache es un producto libre, y multiplataforma.

Presenta entre otras características un diseño modular, que se extiende hasta las funciones básicas de un servidor web, como servir peticiones, conectar con los puertos, etc. Lo cual ofrece dos beneficios importantes:

- Apache puede soportar de una forma más fácil y eficiente una amplia variedad de sistemas operativos. Tiene Módulos de Multiprocesamiento (MPM) que aprovechan las funcionalidades de cada sistema operativo que hacen que sea mucho más eficiente.
- El servidor puede personalizarse mejor para las necesidades de cada sitio web. Por ejemplo, los sitios web que necesitan más que nada escalibildad pueden usar el MPM worker[5], mientras los que requieran estabilidad o compatibilidad MPM prefork[6].

Apache aprovecha eficientemente los recursos del servidor ya que una vez que se haya iniciado, crea una serie de subprocesos "*children processes*", para poder gestionar las solicitudes.

El servidor HTTP Apache, en marzo del 2009 se utilizaba en el 66% de los sitios más activos de internet y en abril ocupaba el 45.95% de todos los sitios publicados sobre internet (fuentes *Netcraft.com*).

1.8.2.2. Internet Information Services (IIS)

Es el servidor de aplicaciones web de Microsoft. Sus principales funcionalidades son la publicación de sitios y aplicaciones web, sitios FTP, SMTP y servicios de noticias. Dispone de soporte necesario para crear páginas en ASP. Su principal problema radicaba en la pobre seguridad de sus primeras versiones, aspecto que fue resuelto en la versión 6.0, en la que Microsoft ha cambiado el comportamiento de los controles ISAPI[7] pre-instalados. Es un sistema esencial destinado a la utilización de los servicios de Internet basado en la plataforma Windows.

1.8.3. Gestores de Base de Datos

Existen muchos Sistemas de Gestión de Bases de Datos Relacionales (RDBMS) en la actualidad, los más recomendados por su garantía para aplicaciones reales son: Oracle, MySQL, Microsoft SQL Server, Sybase, Interbase, PostgreSQL y DB/2.

[5] Módulo de Multiprocesamiento que implementa un servidor hibrido, multihilo y multiproceso.
[6] Módulo de Multiprocesamiento que implementa un servidor que separa cada petición como un proceso independiente y no es multihilo.
[7] ISAPI: Es una interfaz de programación de aplicaciones (API) para el servidor web IIS.

1.8.3.1. PostgreSQL

PostgreSQL es un sistema de gestión de bases de datos objeto-relacional (RDBMS) de software libre, publicado bajo la licencia BSD[8], característica que lo sitúa por delante de otros productos competitivos como Oracle o Microsoft SQL Server 2000, ambos; software propietarios.

Es un sistema objeto-relacional, ya que incluye características de la orientación a objetos, como puede ser la herencia, tipos de datos, funciones, restricciones, disparadores, reglas e integridad transaccional. A pesar de esto no es un sistema de gestión de bases de datos puramente orientado a objetos.

El servidor de BD de código abierto más grande del mundo. Presenta algunas **características** como:

- Integridad Referencial**:** PostgreSQL soporta integridad referencial, la cual es utilizada para garantizar la validez de los datos de la base de datos.
- API Flexible**:** La flexibilidad del API de PostgreSQL ha permitido a los vendedores proporcionar soporte al desarrollo fácilmente para el RDBMS PostgreSQL. Estas interfaces incluyen Object Pascal, Python, Perl, PHP, ODBC, Java/JDBC, Ruby, TCL, C/C++, Pike y TCL.
- Funciones y Triggers**:** Bloques de código que se ejecutan en el servidor y pueden ser escritos en varios lenguajes, incluyendo el nativo denominado PL/pgSQL similar al PL/SQL de Oracle.
- Arquitectura Cliente/Servidor**:** PostgreSQL usa una arquitectura cliente/servidor y proceso-por-usuario. Hay un proceso maestro que se ramifica para proporcionar conexiones adicionales para cada cliente que intente conectar a PostgreSQL.
- Alta Concurrencia**:** La tecnología MVCC (Multi-Version Concurrency Control) permite que mientras un proceso escribe en una tabla, otros accedan a la misma tabla sin necesidad de bloqueos.
- Write-Ahead Logging (WAL)**:** Tecnología para asegurar la integridad de los datos. Registra de cambios antes de que estos sean escritos en la base de datos. Esto garantiza que en caso de que la BD se corrompa, existirá un registro de las transacciones que permitirá restaurar la BD.
- Tiene herramientas libres que soportan la réplica maestro-maestro y maestro-esclavo sincrónica y asincrónicamente como son el PyReplica, el Slony, el PgCluster, etc.

[8] BSD: licencia que pertenece al grupo de licencias de software libre, pero al contrario de GPL permite el uso del código fuente en software no libre.

- Soporta los tipos de datos base, así como: tipo fecha, monetarios, elementos gráficos, datos sobre redes (MAC, IP...), cadenas de bits, array, etc. También permite la creación de tipos propios.

1.8.3.2. Microsoft SQL Server

Producido por Microsoft. Su principal lenguaje de consulta es Transact SQL, una implementación del lenguaje de consulta estructurado ANSI/ISO usado tanto por Microsoft como por Sybase. SQL Server es comúnmente usado por empresas que necesitan de pequeñas a medianas bases de datos, aunque en los últimos años ha sido ampliamente adoptado por empresas que poseen grandes bases de datos empresariales. Microsoft SQL Server constituyó la entrada de Microsoft hacia el mercado de bases de datos de nivel empresarial, compitiendo con Oracle, IBM, y luego, el propio Sybase. Su última versión SQL Server 2005, es un potente gestor de bases de datos, que incorpora numerosas características completamente novedosas, lo que lo hace un producto extremadamente costoso.

1.8.3.3. Oracle

Se considera como uno de los sistemas de BD más completos, destacando su soporte de transacciones, estabilidad, escalabilidad y ser multiplataforma. En 1992 Oracle versión 7h (h significaba *data warehouse*) apareció con soporte para la integridad referencial, procedimientos almacenados y triggers. En 2001 Oracle 9i apareció en el mercado con 400 nuevas características, incluyendo la posibilidad de leer y escribir documentos XML. 9i también brindaba una opción para Oracle RAC (Real Application Clusters), una base de datos de clústeres de computadoras, como reemplazo a Oracle Parallel Server (OPS) que había lanzado antes.

Desventajas:

- Es un software propietario, sumado que tiene un elevado precio de adquisición y de soporte técnico.
- Requiere elevado conocimiento para su mantenimiento en entornos de producción.
- La seguridad de la plataforma, y las políticas de suministro de parches de seguridad, incrementan el nivel de exposición de los usuarios. En los parches de actualización provistos durante el primer semestre de 2005 fueron corregidas 22 vulnerabilidades públicamente conocidas, algunas de ellas con una antigüedad de más de 2 años.

Ventajas:

- Hace uso de los recursos del sistema en todas las arquitecturas de hardware, para garantizar su aprovechamiento al máximo en ambientes cargados de información.
- Corre automáticamente en más de 80 arquitecturas de hardware y software distintas sin tener la necesidad de cambiar una sola línea de código.
- Soporta datos alfanuméricos, y también soporta textos sin estructura, imágenes, audio y video.
- Incluye mejoras de rendimiento y de utilización de recursos.
- Soporta aplicaciones de procesamiento de transacciones online (OLTP) y de *data warehousing* mayores y más exigentes.

1.8.3.4. MySQL

Es un sistema gestor de base de datos cliente/servidor, multihilo y multiusuario. Se puede obtener también como una biblioteca multihilo que se puede enlazar dentro de otras aplicaciones para obtener un producto más pequeño, más rápido, y más fácil de manejar. Es la base de datos de código abierto más popular en internet, que tiene entre sus usuarios a gigantes como Google, Yahoo!, YouTube, MySpace, etc.

MySQL es desarrolla por la compañía MySQL AB en un modelo de licenciamiento dual. Cualquier empresa o persona interesada puede descargar el software de MySQL de Internet y usarlo sin pagar por ello. En Febrero del 2008, MySQL AB es adquirida por la empresa Sun Microsystems, y en el 2009 la corporación Oracle compra Sun Microsystems, lo cual crea cierta incertidumbre entre la comunidad sobre el futuro de MySQL.

Ventajas:

- Mejores utilidades de administración (*backup*, recuperación de errores, etc.), contando con un sistema de replicación multihilo en los servidores esclavos.
- Soporta cinco tipos de tablas: MyISAM, ISAM, HEAP, BDB (Base de Datos Berkeley), e InnoDB. InnoDB y BDB son tablas transaccionales.

- Recuperación automática ante fallas: Si MySQL se cuelga inesperadamente, no suele perder información ni corromper los datos, InnoDB automáticamente completará las transacciones que quedaron incompletas.
- Integridad referencial: Ahora se pueden definir llaves foráneas entre tablas InnoDB relacionadas para asegurarse de que un registro no puede ser eliminado de una tabla si aún está siendo referenciado por otra tabla.
- SELECTs sin bloqueo: El motor InnoDB usa una técnica conocida como *multi-versioning* que elimina la necesidad de hacer bloqueos en consultas SELECT muy simples.
- Puede trabajar en distintas plataformas y S.O. distintos, además es multiproceso, es decir puede usar varias CPU si éstas están disponibles y consume muy pocos recursos, tanto de CPU como memoria.

1.8.4. Herramientas CASE

Las herramientas CASE (*Computer Aided Software Engineering*) Ingeniería de Software Asistida por Computadoras, son diversas aplicaciones informáticas destinadas a aumentar la productividad en el desarrollo de software. Algunas de las cuales se describen a continuación:

1.8.4.1. Rational Rose Enterprise Suite

El *Rational* es una herramienta CASE basada en UML que permite crear los diagramas que se van generando durante el proceso de ingeniería en el desarrollo del software. Es completamente compatible con la metodología RUP, brinda muchas facilidades en la generación de la documentación del software que se está desarrollando, además posee un gran número de estereotipos predefinidos que facilitan el proceso de modelación del software. Es capaz de generar el código fuente de las clases definidas en el flujo de trabajo de diseño, pero tiene la limitación de que aún hay varios lenguajes de programación que no soporta o que sólo lo hace a medias. Por otra parte, una vez que se tiene el diagrama de clases persistentes a partir del cual se genera la base de datos del sistema, no existe la posibilidad de exportar ese modelo hacia algún sistema gestor de bases de datos. Permite además generación de código PHP mediante la utilización de RosePhpTool; *plugins* incorporado de manera independiente durante el desarrollo del proyecto.

1.8.4.2. Visual Paradigm – UML

Visual Paradigm para UML es una de las herramientas UML CASE del mercado, considerada como muy completa y fácil de usar, con soporte multiplataforma y que proporciona excelentes facilidades de interoperabilidad con otras aplicaciones. Fue creada para el ciclo de vida completo del software que lo automatiza permitiendo la captura de requisitos, análisis, diseño e implementación. Visual Paradigm para UML también proporciona características tales como generación del código, ingeniería inversa y generación de informes. Tiene la capacidad de crear el esquema de clases a partir de una base de datos y crear la definición de base de datos a partir del esquema de clases. Permite invertir código fuente de programas, archivos ejecutables y binarios en modelos UML al instante, creando de manera simple toda la documentación. Está diseñada para usuarios interesados en sistemas de software de gran escala con el uso del diseño orientado a objeto, además apoya los estándares más recientes de las notaciones de Java y de UML. Incorpora el soporte para trabajo en equipo, integrándose fácilmente con Subversion.

1.9. Propuesta Arquitectónica

Teniendo en cuenta que los fiscales se organizan por esferas de trabajo y que cada una de estas esferas tiene bien definidos sus procesos. Se decide realizar la estructura del proyecto acorde a su forma de trabajo, de ahí que el sistema se dividió en varios subsistemas los cuales responden a las esferas de trabajo, y por cada subsistema se creó un conjunto de módulos correspondientes a los procesos que se llevan a cabo es esa esfera (ver Anexo 1).

Se propone utilizar una arquitectura en capas común para todos los subsistemas y módulos del proyecto, ya que favorece la reutilización y disminuye el impacto de los cambios. Dada la complejidad del sistema, que cuenta con 41 módulos se propone utilizar la metodología de desarrollo RUP, la cual se adapta muy bien a proyectos de largo plazo, además que se ha probado en numerosos proyectos en la universidad, siendo la metodología que se imparte en la carrera por lo existe un amplio dominio de ella, y propone una documentación detalla de cada artefacto generado.

Se seleccionaron un conjunto de herramientas que contribuirán a obtener un sistema flexible, escalable, reusable y mantenible. A continuación se listan las de mayor peso para la arquitectura:

- Herramienta CASE *(Computer Aide Software Engineering)*: Visual Paradigm Suite 3.0.
- Plataforma de Desarrollo: Linux - Apache 2 - PostgreSQL 8.2 – PHP 5 (LAPP).
- Framework de Desarrollo: Symfony 1.0.17.
- Herramienta para la generación de reportes personalizados: Sistema de Gestión de Reportes "Olympia".
- Herramienta para replicar bases de datos: PgCluster 1.5.0.
- La herramienta IDE de desarrollo: Eclipse 3.2.2 + PDT.
- Para el Sistema de Control de Versiones: Subversion 1.4.2.

Para la selección de dichas herramientas se tuvo en cuenta principalmente que fueran software libre y las mejores para su funcionalidad, priorizando aquellas que estuvieran soportadas por una gran comunidad de usuarios, como una alternativa para evacuar dudas, ya que la mayoría de los desarrolladores no contaba con experiencia en el desarrollo de aplicaciones de este tipo.

1.10. Conclusiones

Se realizó un estudio del estado del arte de la arquitectura de software, que comprendió una breve introducción a la arquitectura, además se describieron estilos, patrones, y atributos de calidad de software. Se analizaron las metodologías, y herramientas a utilizar durante el ciclo de desarrollo de software. Se propuso la base arquitectónica para la construcción de la aplicación, a partir de la selección de tecnologías libres y/o multiplataforma, siempre que hubo oportunidad.

CAPÍTULO 2

2. DISEÑO ARQUITECTÓNICO

"El aprendizaje más importante proviene de la experiencia directa"

2.1. Introducción

En el presente capítulo se realiza la descripción del diseño arquitectónico del Sistema de Gestión Fiscal, ilustrando de manera sencilla los módulos que lo conforman y el objetivo que persigue la arquitectura dentro del proyecto. Para ello se describe el sistema en diferentes vistas arquitectónicas según el modelo de Kruchten vinculado al RUP (*Rational Unified Process*). También se define qué variantes utilizar en temas relacionados con estilos y patrones de arquitectura, así como las posibilidades que brinda la utilización del *framework* Symfony; como estructura dentro de la cual el sistema puede ser organizado y desarrollado. Se tienen en cuenta las restricciones de hardware o software de la arquitectura y se describe las tecnologías y herramientas que se utilizarán en el desarrollo del sistema.

2.2. Visión del Sistema

El sistema consiste en un sitio web a través del cual el fiscal desde la esfera de trabajo a la que pertenece puede realizar todas sus actividades o procesos pertinentes, los cuales se encuentran digitalizados en una secuencia lógica de proceder según lo establecido y asistidos por las leyes que en cada caso correspondan. Este portal constituye una completa ayuda a la toma de decisiones de los fiscales el cual viabiliza y humaniza el trabajo de los mismos, permitiendo reducir al máximo el margen de errores.

El sistema proporcionara entre otras funcionalidades la automatización de los procesos fundamentales en lo que interviene la Fiscalía General de la República:

- Procesos Penales (PP)
- Verificación Fiscal (VF)
- Protección a los Derechos Ciudadanos (PDC)
- Gestión de Cuadros y Personal de Apoyo (GCPA)
- Relaciones Internacionales (RI)
- Herramientas Comunes a Todas las Áreas (HCTA)
- Control de la Legalidad en Establecimientos Penitenciarios (CLEP)
- Dirección General de Control (DGC)

Desde el punto de vista físico cada uno de los subsistemas y funcionalidades que componen la aplicación residirán en un mismo servidor, el acceso a cada subsistema será restringido solo al personal que pertenece dicha esfera. Cada uno de los subsistemas es independiente, solo comparten la información del modelo de datos de la organización.

2.3. Alcance

El diseño arquitectónico comprende la primera iteración de la arquitectura del proyecto, que abarca hasta la fase de despliegue del subsistema GCPA, que está compuesto por 6 módulos (Gestión de Cuadros, Gestión del Personal de Apoyo, Capacitación, Reportes y Búsquedas, Generalidades, y Captación de Plantillas). No obstante la línea base de la arquitectura se define al principio y es común para sistema completo.

2.4. Representación Arquitectónica

La representación de la arquitectura define los elementos imprescindibles para el desarrollo del sistema. El hecho de que sea una aplicación web, brindara las siguientes ventajas:

- Agilidad y sencillez en el despliegue y actualización de la aplicación.
- Disminuye los requerimientos de hardware en las PC clientes y por tanto el costo de inversión.
- Ganancia en la seguridad de la aplicación al estar localizada en un servidor alejado del acceso

directo del usuario.

- Amplio soporte de *frameworks* que apoyan el desarrollo en la plataforma.
- Mayor experiencia del equipo de desarrollo en estas aplicaciones.
- Mayor accesibilidad para los usuarios, al estar disponible desde cualquier navegador.

2.4.1. Estilos Arquitectónicos

Para separar la lógica de presentación, la lógica de datos, y la lógica de negocio, se decidió utilizar el estilo arquitectónico en capas, el cual contribuye a alcanzar ciertos atributos de calidad deseados del sistema como son la flexibilidad, la mantenibilidad y la reusabilidad. Una correcta aplicación del estilo en capas garantizará que los cambios que sean necesarios aplicar a una capa solo afectaran a las capas vecinas, no a la aplicación completa, facilitando el mantenimiento y la flexibilidad del sistema.

2.4.2. Patrones

2.4.2.1. Patrones arquitectónicos

Como parte del *framework* de desarrollo se utilizan varios patrones arquitectónicos que permiten entre otras ventajas lograr un sistema más seguro, flexible, y mantenible. A continuación se listan estos patrones y una breve descripción de cada uno:

- Modelo Vista Controlador: También conocido como arquitectura MVC. Divide la interacción con la interfaz de usuario en tres elementos diferentes. Es un patrón clásico del diseño web, implementado por todos los *framework* de PHP conocidos.
- Data Mapper (Mapeo de Datos): La capa *Mapper* (Mapeo) es la encargada de convertir la base de datos relacional a objetos y viceversa, para hacer uso de las ventajas del diseño orientado a objetos. Este patrón esta implementado por el ORM (*object relational mapping*) Propel.

2.4.2.2. Patrones de diseño

Las mejores prácticas de diseño son incluidas a través del *framework*, el cual implementa internamente los patrones de diseño comúnmente utilizados para el desarrollo de aplicaciones web, a continuación se describen brevemente algunos de ellos:

- *Singleton* (Instancia única)**:** Garantiza la existencia de una instancia única a los objetos del núcleo del *framework*, permitiendo acceso a los mismos desde cualquier parte de la aplicación.

- *Abstract Factory* (Fábrica abstracta)**:** Permite trabajar con objetos de distintas clases, de manera independiente y haciendo transparente la clase concreta que se esté usando. Cuando el *framework* necesita crear un nuevo objeto para una petición, busca en la definición de la factoría el nombre de la clase que se debe utilizar para esta tarea.
  Factoría:
  sfContext::getInstance()->getI18N()
  sfContext::getInstance()->getRouting()
  sfContext::getInstance()->getLogger()

- *Decorator* (Envoltorio)**:** La plantilla global *layout.php*, almacena el código HTML que es común a todas las páginas de la aplicación, para no tener que repetirlo en cada una. El contenido de la plantilla se integra en el *layout*, o se puede ver como que el *layout* decora la plantilla.

2.4.2.3. Patrones GRASP

Los patrones de asignación de responsabilidades GRASP basan su utilidad en definir principios fundamentales del diseño orientado a objetos y las responsabilidades de estos objetos de acuerdo a su comportamiento. Los GRASP utilizados a través del *framework* son:

- Alta Cohesión: Symfony permite asignar responsabilidades con una alta cohesión, por ejemplo la clase *sfContext* colabora con las demás clases del *framework* para realizar diferentes operaciones, instanciar objetos y acceder a las *properties*, está formada por diferentes

funcionalidades que se encuentran estrechamente relacionadas proporcionando que el software sea flexible frente a grandes cambios.

- Bajo Acoplamiento: La clase de las acciones hereda solamente de *sfActions*, y las clases del modelo heredan exclusivamente de las clases bases correspondientes, contribuyendo al bajo acoplamiento entre las clases.

- Controlador: Todas las peticiones web son manejadas por un solo controlador frontal *index.php*, el cual se encarga entre cosas de localizar las librerías del *framework*, cargar e inicializar las clases del núcleo, cargar la configuración y pasar la petición al objeto *sfController*, el cual utiliza el sistema de enrutamiento para determinar la acción a ejecutar y los parámetros de la petición.

2.4.3. Framework de Desarrollo

Un *framework* simplifica el desarrollo de una aplicación mediante la automatización de algunos de los patrones utilizados para resolver las tareas comunes, permitiendo al desarrollador dedicarse a aspectos específicos de la aplicación. Además, el *framework* proporciona estructura al código fuente, forzando al desarrollador a crear código más legible y más fácil de mantener. El cual facilita la programación de aplicaciones, ya que encapsula operaciones complejas en instrucciones sencillas.

2.4.3.1. Symfony

A partir de la estructura de organización del sistema en capas se decide la utilización de un *framework* de PHP5 el cual entre otras ventajas divide el desarrollo de aplicaciones en capas. Después de hacer un estudio de los principales *frameworks* de desarrollo para PHP disponibles (Kumbia, CakePHP, ZendFramework, Symfony, CodeIgniter, entre otros), se llegó a la conclusión que la mayoría poseían las mismas características: uso de MVC, mapeo de BD, mecanismos de caché, manejo de validación, de autenticación, en fin las principales funcionalidades de un *framework*. Se decidió escoger Symfony, teniendo en cuenta los atributos que presenta:

- Escalable: Symfony es infinitamente escalable si se disponen de los recursos necesarios. Yahoo utiliza Symfony para programar aplicaciones con 20 millones de usuarios y 12 idiomas.
- Soporte: Symfony sigue una política de tipo LTS (*long term support*), soporte a largo plazo. Las versiones estables se mantienen durante 3 años sin cambios pero con una continua corrección de los errores conocidos.
- Código: desde su primera versión Symfony ha sido creado para PHP5, desechando la versión PHP4 (que ha sido declarada obsoleta recientemente). Lo que le permite una mejor utilización de las técnicas orientadas a objetos.
- Seguridad: se puede controlar hasta el último acceso a la información e incluye por defecto protección contra ataques XSS[9] (*cross-site scripting*) y CSRF[10] (*Cross Site Request Forgery*).
- Documentado: se trata del *framework* PHP mejor documentado: miles de páginas en el wiki oficial, tutoriales de hasta 250 páginas y un libro gratuito de casi 500 páginas. Además, el libro está completamente traducido al español.
- Calidad: su código fuente incluye más de 8.000 pruebas unitarias y funcionales.
- Garantía: tiene una de las comunidades más activas, que es una garantía para la evacuación de dudas.
- Extensible: a través de una multitud de *plugins* disponibles, que ya supera la cifra de 500 *plugins*.
- Reusabilidad: se publican algunas partes del *framework* en forma de componentes independientes, que puedes usar independientemente del *framework*.

2.4.3.1.1. Características del *framework*

Symfony ha sido probado en numerosos sitios web de primer nivel. Symfony es compatible con la mayoría de gestores de bases de datos, como MySQL, PostgreSQL, Oracle y Microsoft SQL Server. Se puede ejecutar tanto en plataformas Unix (Linux, FreeBSD, etc.) como en plataformas Windows. A continuación se muestran algunas de sus características:

[9] XSS: tipo de ataque que consiste en insertar código *Javascript* en los formularios para que sea ejecutado junto al código HTML.
[10] CSRF: tipo de ataque que consiste en sitios web maliciosos que al visitarlos envían peticiones invisibles con tú usuario a sitios confiables, aprovechando una sesión abierta.

- La autenticación y la gestión de credenciales simplifican la creación de secciones restringidas y la gestión de la seguridad de usuario.
- Soporte de e-mail incluido.
- Independiente del sistema gestor de bases de datos.
- La capa de presentación utiliza plantillas y *layouts* que pueden ser creados por diseñadores HTML sin ningún tipo de conocimiento del *framework*.
- Los *helpers* incluidos permiten minimizar el código utilizado en la presentación, ya que encapsulan grandes bloques de código en llamadas simples a funciones.
- Los formularios incluyen validación automatizada y relleno automático de datos ("*population*"), lo que asegura la obtención de datos correctos y mejora la experiencia de usuario.
- La gestión de la caché reduce el ancho de banda utilizado y la carga del servidor.
- Los *plugins*, las factorías (patrón de diseño "*Factory*") y los "*mixin*[11]" permiten realizar extensiones a medida de Symfony.
- Las interacciones con *Ajax* son muy fáciles de implementar mediante los *helpers* que permiten encapsular los efectos *Java Script* compatibles con todos los navegadores en una única línea de código.
- Los datos incluyen mecanismos de escape que permiten una mejor protección contra los ataques producidos por datos corruptos.

Symfony puede ser completamente personalizado para cumplir con los requisitos de las empresas que disponen de sus propias políticas y reglas para la gestión de proyectos y la programación de aplicaciones, e incluye varias herramientas que permiten automatizar las tareas más comunes de la ingeniería del software:

- Las herramientas que generan automáticamente código han sido diseñadas para hacer prototipos de aplicaciones y para crear fácilmente la parte de gestión de las aplicaciones.
- El *framework* de desarrollo de pruebas unitarias y funcionales proporciona las herramientas ideales para el desarrollo basado en pruebas ("*test-driven development*").
- La barra de depuración web simplifica la depuración de las aplicaciones, ya que muestra toda la información que los programadores necesitan sobre la página en la que están trabajando.
- La interfaz de línea de comandos automatiza la instalación de las aplicaciones entre servidores.

[11] Un *mixin* es un grupo de métodos o funciones que se juntan en una clase para que otras clases hereden de ella.

- El completo sistema de log permite a los administradores acceder hasta el último detalle de las actividades que realiza la aplicación.

Implementación del patrón MVC en Symfony:

Symfony separa la lógica de negocio (el modelo) y la presentación (la vista) por lo que se consigue un mantenimiento más sencillo de las aplicaciones, y además proporciona grandes ventajas, como la organización del código, la reutilización, la flexibilidad, entre otras. MVC está formado por tres niveles a continuación se explica cómo Symfony implementa cada uno de los niveles de este patrón.

- La Vista

  La vista se encarga de producir las páginas que se muestran como resultado de las acciones. La vista en Symfony está compuesta por diversas partes, las plantillas (que son la presentación de los datos de la acción que se está ejecutando) y el *layout* (que contiene el código HTML común a todas las páginas). Estas partes están formadas por código HTML que contiene y trozos de código PHP, que normalmente son llamadas a los diversos *helpers* disponibles.

  Helpers: son funciones de PHP que devuelven código HTML y que se utilizan en las plantillas. Que tiene como ventajas el ahorro de tiempo, la reutilización y reducción del código, ya que encapsulan grandes bloques de código, utilizados habitualmente en la vista. La función *link_to()* de la Fig. 2-1 es un ejemplo de *helper* que crea un enlace al módulo: artículo, acción: leer.

  Plantilla*:* Su contenido está formado por código HTML, y código PHP sencillo que normalmente son llamadas a las variables definidas en la acción (mediante la instrucción *$this->nombre_variable = 'valor';*). A continuación se muestra el código típico de una plantilla.

```
<h1>Bienvenido</h1>
    <p>¡Hola de nuevo, <?php echo $nombre ?>!</p>
    <ul>¿Qué es lo que quieres hacer?
        <li><?php echo link_to('Leer los últimos artículos', 'articulo/leer') ?></li>
    </ul>
```

Figura 2-1. Plantilla de ejemplo *indexSuccess.php*

Layout de las páginas*:* La figura anterior no es un documento XHTML válido, le faltan la definición del *DOCTYPE* y las etiquetas <html> y <body>. Estos elementos se encuentran definidos en una plantilla global, que almacena el código HTML que es común a todas las páginas de la aplicación. El contenido de la plantilla se integra en el *layout* como se muestra en la figura a continuación. Este comportamiento es una implementación del patrón de diseño "*decorator*".

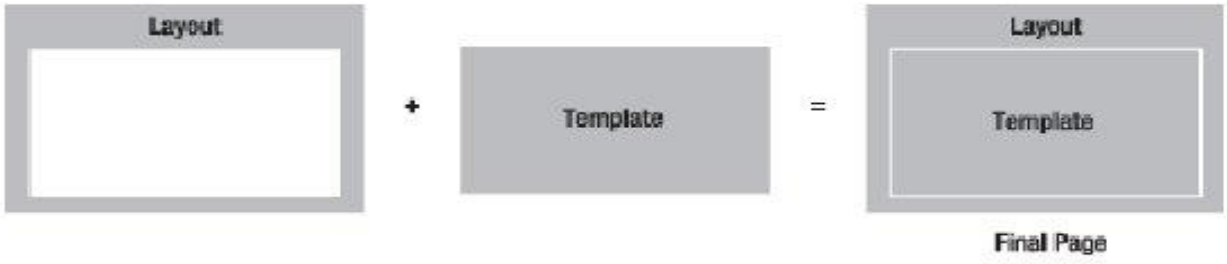


Figura 2-2. Plantilla decorada con un layout.

Figura 2-3.

- El Controlador

Todas las peticiones web son manejadas por un solo controlador frontal, que es el punto de entrada único de toda la aplicación. El controlador normalmente se divide en un controlador frontal, que es único para cada aplicación, y las acciones, que incluyen el código específico del controlador de cada página.

El controlador carga la configuración y determina el módulo y la acción a ejecutarse, por medio de un sistema de enrutamiento. Además maneja automáticamente las sesiones del usuario y es capaz de almacenar datos de forma persistente entre peticiones. Mejora el mecanismo de manejo de sesiones incluido en PHP5 para hacerlo más configurable y más fácil de usar.

En Symfony, el controlador, está dividido en varios componentes que se utilizan para diversos propósitos: los objetos *request*, *response* y *session* dan acceso a los parámetros de la petición, las cabeceras de las respuestas y a los datos persistentes del usuario.

❖ El Modelo

El componente que se encarga por defecto de gestionar el modelo en Symfony es una capa de tipo ORM realizada mediante el ORM Propel. En las aplicaciones Symfony, la gestión de los datos almacenados en la base de datos se realiza mediante objetos. Este comportamiento permite un alto nivel de abstracción y permite una fácil portabilidad.

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

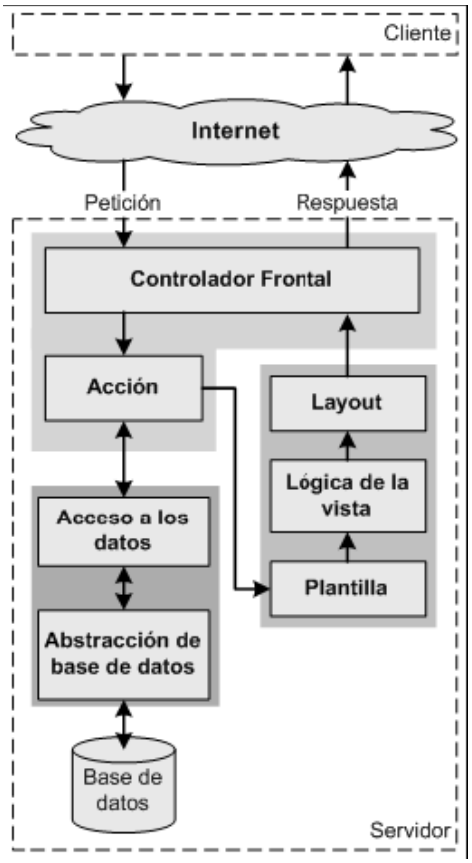


Figura 2-4. Patrón arquitectónico MVC que implementa Symfony.

Mapeo de Objetos a Bases de datos (ORM)**:** Las bases de datos siguen una estructura relacional. PHP5 y Symfony por el contrario son orientados a objetos. Por este motivo, para acceder a la base de datos como si fuera orientada a objetos, es necesaria una interfaz que traduzca la lógica de los objetos a la lógica relacional.

Symfony gracias a su política de no reinventar la rueda, incorpora algunas librerías de terceros como es el caso del ORM Propel, que trae un conjunto de ventajas como:

- Reutilización: permitiendo llamar a los métodos de un objeto de datos desde varias partes de la aplicación e incluso desde diferentes aplicaciones.
- Evita utilizar una sintaxis específica de un sistema de bases de datos concreto.
- Genera automática las clases de la capa del modelo, en función de la estructura de datos de la aplicación.

Abstracción de la base de datos**:** La abstracción de la base de datos es completamente invisible al programador, ya que la realiza otro componente específico llamado Creole. Así, que si se cambia el sistema gestor de bases de datos en cualquier momento, no se debe reescribir ni una línea de código, ya que tan sólo es necesario modificar un parámetro en un archivo de configuración.

2.4.4. Plataforma de Desarrollo

El sistema se desarrollará en una plataforma libre formada por *Linux-Apache-PostgreSQL-PHP* (LAPP) el principal motivo de esta decisión es la política de migración hacia software libre de la Fiscalía General de la República y del País. Esta plataforma tiene gran aceptación a nivel global, en ella se desarrollan la mayoría de las aplicaciones web hoy en día.

2.4.4.1. Porque se decidió escoger esta plataforma?

Se decidió escoger como sistema operativo a **Linux**, primeramente por ser libre, considerando que la fiscalía no dispone de muchos recursos, además tiene excelente rendimiento para aplicaciones de servidor ya que puede ejecutarse sin el servidor gráfico *X-Windows*, su diseño modular permite ajustar el servidor solo a las funcionalidades que se necesitan, existe una menor cantidad de *malwares* que en

Windows, entre otras causas por el control sobre los permisos de ejecución de los archivos por usuarios normales.

Se escogió **Apache2** como servidor web por varias características que hacen que sea el servidor web más usado actualmente. Apache2 es un producto libre de código abierto, con soporte multiplataforma, entre sus mejores características presenta un diseño modular, lo que permite que pueda personalizarse mejor para las necesidades del sitio web. Es en gran medida escalable ya que es multiproceso y multihilo, lo que le permite aprovechar mejor los recursos de memoria y procesador.

Como Sistema Gestor de Base de Datos se decidió escoger ***PosgreSQL 8.3*** por las siguientes razones:

- Existe en el país y en nuestra universidad una comunidad de PostgreSQL, lo cual es una fuente de conocimiento.
- Es una variante libre, ante otros productos competitivos como Oracle o Microsoft SQL Server 2000.
- Es un Sistema de Bases de Datos Objeto-Relacional que facilita usar técnicas orientadas a objetos como son la herencia entre tablas, los tipos de datos y las funciones.
- Cuenta con un sistema de respaldos para la protección de los datos ante un fallo del sistema.
- Restringe el acceso a la base de datos a un nivel de seguridad tal, que es obligatorio definir desde qué equipos se pueden conectar a la base de datos, cuales usuarios y a qué bases de datos se pueden conectar.

El lenguaje de programación seleccionado fue **PHP5**, pensado específicamente para el desarrollo web desde su creación, las ventajas de este lenguaje son las siguientes:

- Es un lenguaje con soporte multiplataforma.
- No requiere grandes recursos del sistema para su funcionamiento, por la principal razón de ser un lenguaje interpretado.
- Tiene capacidad de conexión con la mayoría de los manejadores de base de datos que se utilizan en la actualidad, destaca su conectividad con MySQL y PostgreSQL.
- Posee una amplia documentación en su página oficial (*http://www.php.net/*), destacando que todas las funciones del lenguaje están bien explicadas y con varios ejemplos de su uso.
- Es libre, por lo que se presenta como una alternativa de fácil acceso para todos.

- Permite las técnicas de Programación Orientada a Objetos, mejorado en su versión 5.
- Biblioteca nativa de funciones sumamente amplia e incluida.
- No requiere definición de tipos de variables.
- Tiene manejo de excepciones.

2.4.5. Herramientas de Desarrollo

Herramienta CASE: Visual Paradigm Suite 3.0

Visual Paradigm aunque no es libre, tiene soporte multiplataforma, utiliza UML 2.1, creado para el ciclo de vida completo del software que lo automatiza permitiendo la captura de requisitos, análisis, diseño e implementación. Tiene la capacidad de crear el esquema de clases a partir de una base de datos y crear la definición de base de datos a partir del esquema de clases. Genera código para varios lenguajes entre ellos PHP y hacer ingeniería inversa a partir de diagramas UML, creando de manera simple toda la documentación. Se integra con varios IDEs, entre ellos Eclipse.

Herramienta IDE de desarrollo: Eclipse 3.2.2 + PDT

Eclipse es una plataforma de software de código abierto con soporte multiplataforma. Es una plataforma, que ha sido usada para desarrollar un IDE, como el llamado *Java Development Toolkit* (JDT) y el compilador (ECJ) que se enmarca como parte del proyecto Eclipse y que son usados también para desarrollar el mismo Eclipse. El entorno integrado de desarrollo (IDE) de Eclipse emplea módulos (*plug-in*) para proporcionar toda su funcionalidad al frente de la plataforma de cliente rico. Este diseño modular permite la integración de componentes de software. Adicionalmente Eclipse permite extenderse para usar lenguajes de programación como PHP, C/C++ y Phyton, y lenguajes de sistemas de gestión de base de datos.

2.5. Principales restricciones arquitectónicas

2.5.1. Distribución geográfica

La FGR tiene distribuidas sus sedes geográficamente en todo el país, una sede nacional, 14 sedes provinciales y 169 sedes municipales, desde donde se accederá al sistema mediante una red dedicada ADSL con velocidades de hasta 2Mbps. Cuenta con servidores dedicados y *Frame Relay*[12] en todas sus sedes.

Teniendo en cuenta que el sistema está previsto a ser utilizado por todas las fiscalías del país, se decidió no utilizar un sistema centralizado debido a que la concurrencia de usuarios conectados seria alta, a través de un ancho de banda bajo sobre una infraestructura de red no confiable. Para solventar este problema se decidió implementar una arquitectura distribuida del sistema, siendo una forma eficiente de disminuir la carga de usuarios, para esto se utilizó un servidor web y uno de base de datos por cada sede. Aunque aún no se cuenta de toda la tecnología necesaria.

2.5.2. Crecimiento de los datos

La FGR genera anualmente gran cantidad de información, se promedia que anualmente se tramitan 80000 expedientes, a pesar de que no hay que digitalizar estos expedientes, sí se genera gran cantidad de documentación a partir de ellos.

La información no se almacena en objetos con lo que se disminuye el crecimiento de los datos. Además se diseñó la base de datos de forma que no se guarde en ella información redundante. Como característica de SGBD se tiene un tamaño máximo de base de datos ilimitado, en el caso que los dispositivos de almacenamiento físico de un nodo no satisfagan la demanda de capacidad entonces se crearán clústeres de base de datos en nodos auxiliares.

[12] Arquitectura de red de transporte orientada a conexión que mejora las funciones del protocolo de red X.25.

2.5.3. Sistema auditable

El sistema tiene que ser obligatoriamente auditable[13], o sea todo cambio o modificación en el sistema debe ser atribuible a un usuario particular según sus datos de autenticación. Para darle solución a este requerimiento se implementó una variante del patrón de diseño de seguridad *Logger* con lo cual se registran todas las acciones realizadas por el usuario en el sistema, en forma de *logs* y se almacenan en una tabla de la base de datos con la siguiente estructura: usuario, fecha en que realizó la acción y un mensaje que describe la acción.

2.5.4. Integridad de la base de datos

Dada a la importancia que representa la base de datos para el sistema siendo considerada la vida de la organización, se previó la implantación de varias medidas encaminadas a garantizar la disponibilidad y la seguridad de la base de datos. El sistema debe mantener en todo momento la seguridad de la información y la protección contra acciones no autorizadas que puedan afectar la integridad de los datos, asegurando su restauración en caso de catástrofes. Para contribuir a la seguridad del servidor de base de datos se hará uso de las funcionalidades que provee PostgreSQL que se materializa en 2 aspectos:

2.5.4.1. Control de acceso

El gestor utilizado permite restringir el acceso a la base de datos a un nivel de seguridad especificando desde qué equipos se pueden conectar a la base de datos, así como definir qué usuarios y a qué bases de datos se pueden conectar (Esta configuración se realiza en los ficheros *pg_hba.conf* y *pg_ident.conf*). Además se debe asignar privilegios mínimos a los roles y usuarios del clúster de base de datos.

2.5.4.2. Copias de seguridad y recuperación

Las copias de seguridad son esenciales para las recuperaciones frente a fallos, se deben determinar la frecuencia conveniente, que será en función del tamaño de los datos y de la frecuencia con que son

[13] Se registran todas las trazas de las acciones de los usuarios en el sistema.

modificados, lo cual marcará la estrategia de copias de seguridad. Las copias se automatizarán mediante el uso de herramientas del SO como el *cron* y/o *scripts*. Las formas para realizar las copias no son más que herramientas del SGBD ProsgreSQL. A continuación se describen los tipos de copias de seguridad establecidas:

- Réplica:

  Mediante la implementación de un sistema de replicación *PgCluster*[14] entre los servidores de base de datos, se garantizará[15] que todos tengan la misma información, convirtiéndose en puntos de restauración para en caso que sea necesario restablecer servidores caídos. Aquí juega un papel fundamental el servidor de replicación que sería el responsable de actualizar las bases de datos en ambos sentidos, chequeando que todos los clústeres posean los mismos datos (ver Anexo2).

- Recuperación a un punto del tiempo (*PITR*):

  PostgreSQL almacena todos los ficheros WAL (Write Ahead Log - información sobre las transacciones realizadas) generados por el sistema, los cuales describen cada cambio en los archivos de datos. El propósito principal de estos *logs* es la seguridad ante catástrofes, gracias a ellos la base de datos puede ser restaurada a un estado consistente y sin pérdida de datos. Simultáneamente cada cierto tiempo se realizará una copia de seguridad base (copia total de cluster). Esta copia se realiza a nivel del sistema de ficheros, y alguna inconsistencia en la misma será corregida por el reemplazo de los ficheros de *log* (*WAL - Write Ahead Log*).

  En caso de catástrofe, se utilizará la copia de seguridad base más todos los ficheros WAL archivados desde el término de la última copia de seguridad hasta el momento del fallo, para restaurar la base de datos a un estado consistente y sin pérdida de datos.

  Este tipo de *backup* avanzado *PITR - Point in Time Recovery* permite hacer la recuperación de la base de datos al estado deseado. La administración de PITR se automatizará mediante tareas programadas utilizando el CRON de Linux.

[14] Aplicación de Linux para hacer réplica de bases de datos entre servidores PostgreSQL
[15] Juega un papel fundamental la disponibilidad de las conexiones de red.

2.5.5. Mecanismos de seguridad de la aplicación

Dada las características de privacidad del sistema, la información no debe ser accedida por personal que no esté previamente registrado con los permisos requeridos. Para lo cual se aprovechó la infraestructura proporcionada por el *framework,* donde seguridad es gestionada por la capa del controlador a través de filtros. Antes de ser ejecutada, cada acción pasa por un filtro de seguridad que verifica si el usuario actual tiene privilegios de acceder a la acción requerida.

Añadir esta seguridad a la aplicación requiere dos pasos: declarar los requerimientos de seguridad para cada acción y autenticar a los usuarios con los privilegios para que puedan acceder estas acciones seguras. Los privilegios no son más que permisos asociados a los cargos de los fiscales y el rol que ocupan en el sistema, agrupados bajo un nombre y que permiten organizar la seguridad en grupos.
Dado los requerimientos del cliente fue necesario la creación de un filtro propio para restringir el acceso a la aplicación solo desde direcciones de red (MAC) autorizadas.

Además se hará uso del protocolo de comunicación HTTPs para garantizar que los datos viajen encriptados, habilitando el módulo *mod_ssl* de Apache, adicionalmente se cuenta con una aplicación de firma digital desarrollada en el proyecto para verificar la integridad de los documentos que lo requieran.

2.6. Requisitos no funcionales del sistema

1. RNF de Usabilidad

1.1. El software tendrá siempre la posibilidad de ayuda disponible para cualquier tipo de usuario, lo que le permitirá un avance considerable en la explotación de la aplicación en todas sus funcionalidades.
1.2. Existirán servidores locales con capacidad necesaria para el procesamiento de las solicitudes del conjunto de aplicaciones de las diferentes oficinas.
1.3. Las aplicaciones siempre solicitarán los datos a través del servidor local.
1.4. Desde cada servidor local se establecerá la conexión con servidores centrales para mantener la actualización de los datos en ambos sentidos.

1.5. El tiempo de entrenamiento requerido para que usuarios normales y avanzados sean productivos operando el sistema es de 15 días.

1.6. Debe poseer una interfaz agradable para el cliente.

2. RNF de Confiabilidad

2.1. El sistema estará disponible 24 horas al día, 7 días a la semana.

2.2. Disponibilidad de los casos asignados desde cualquier parte del país.

2.3. Tiempo medio entre fallos es de 1 mes.

2.4. El tiempo permitido para que el sistema quede fuera de operación luego de haber fallado es de 2 días.

2.5. La precisión y exactitud requerida en las salidas del sistema o sea el máximo de errores, es de 5 errores/MLC.

2.6. La herramienta de implementación a utilizar tiene soporte para recuperación ante fallos y errores.

3. RNF de Eficiencia

3.1. Tiempo de respuesta promedio de las peticiones que se realizan al servidor no deberá ser mayor de 3 segundos.

3.2. El número de clientes o transacciones que el sistema puede alojar es de 2000.

4. RNF de Soporte

4.1. Soporte para grandes volúmenes de datos y velocidad de procesamiento.

4.2. Tiempo de respuesta rápido en accesos concurrentes.

4.3. El sistema debe ser multiplataforma.

5. **RNF de Interfaz de Usuario**

5.1. El sistema tiene que ofrecer una interfaz amigable, fácil de operar.

5.2. El sistema tiene que mantener la línea de diseño establecida para la institución que mantiene la uniformidad y representatividad de la misma.

5.3. Diseño sencillo, con pocas entradas, permitiendo que no sea necesario mucho entrenamiento para que los usuarios puedan utilizar el sistema.

6. RNF de Seguridad

6.1. Protección contra acciones no autorizadas o que puedan afectar la integridad de los datos.

6.2. El sistema debe mantener en todo momento la seguridad de la información asegurando la autenticidad de la misma.

6.3. La seguridad se establecerá por roles que se le asignarán a los usuarios que interactúen con el sistema.

6.4. El software brindará solamente aquellas funcionalidades que competen a la Unidad Ejecutora donde este implantado.

6.5. El sistema mantendrá en todo momento las trazas que se corresponden con las diferentes situaciones críticas que se puedan ocurrir.

7. RNF de Hardware

Cliente:

7.1. Memoria RAM de 128 Mb o superior.

7.2. Pentium a 200 MHz de velocidad de procesamiento o superior.

7.3. Con tarjeta de red y puerto USB 2.0.

7.4. Dispositivos de impresión conectado (IMPRESORA HP 1018 LASERJET).

7.5. Dispositivos de escaneo conectado (SCANNER CANON LIDE 90).

Servidores:

7.6. Procesadores Quad-Core Intel® Xeon® E5430 (2.66 GHz, 80 Watts, 1333 FSB).

7.7. Mínimo de capacidad de almacenamiento 32GB.

8. **RNF de Software**

Cliente:

8.1. PC con sistema operativo Windows o Linux.

8.2. PC con navegador Mozilla Firefox instalado tanto en Windows como en Linux.

Servidor:

8.3. Un servidor dedicado en debían 4.0 con Apache2 instalado.

8.4. Un servidor dedicado en debían 4.0 con PostgreSQL 8.3 instalado.

8.5. Un servidor dedicado en debían 4.0 con SMTP instalado.

8.6. Un servidor con DNS instalado.

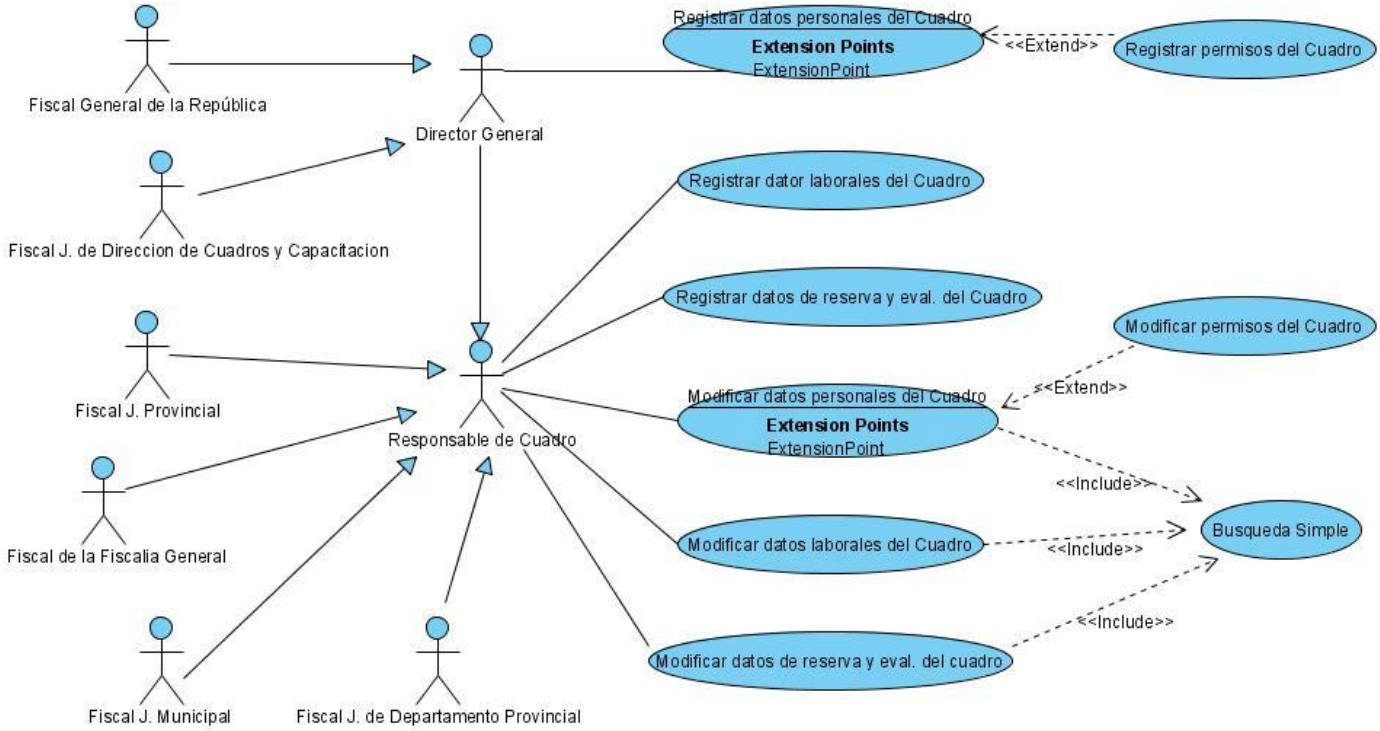
2.7. Vista de Casos de Uso por módulo

Para la realización de la Vista de Casos de Uso, RUP establece que se deben seleccionar los casos de uso más significativos para la arquitectura, que comprende aquellos casos de uso que dan cumplimiento a los requisitos que más desea el cliente, y los que influyen en requisitos no funcionales de gran impacto en la arquitectura, esta selección es propia de cada sistema.

2.7.1. Casos de uso arquitectónicamente significativos del módulo Gestión de Cuadros:

En este módulo se gestiona toda la información personal y laboral del cuadro o fiscal, además se registran las resoluciones de nombramiento, movimiento y cese de funciones del cuadro para asignarle un cargo y quitarlo de sus funciones.

Teniendo en cuenta la descripción del módulo y el documento de Casos de Uso del Sistema del módulo Gestión de Cuadros, resultan los siguientes Casos de Usos significativos para la Arquitectura:



Registrar datos personales de Cuadro: En este caso de uso se introducen los datos personales de los cuadros para la creación del expediente de los mismos. Luego de introducir los datos del cuadro se guarda la información.

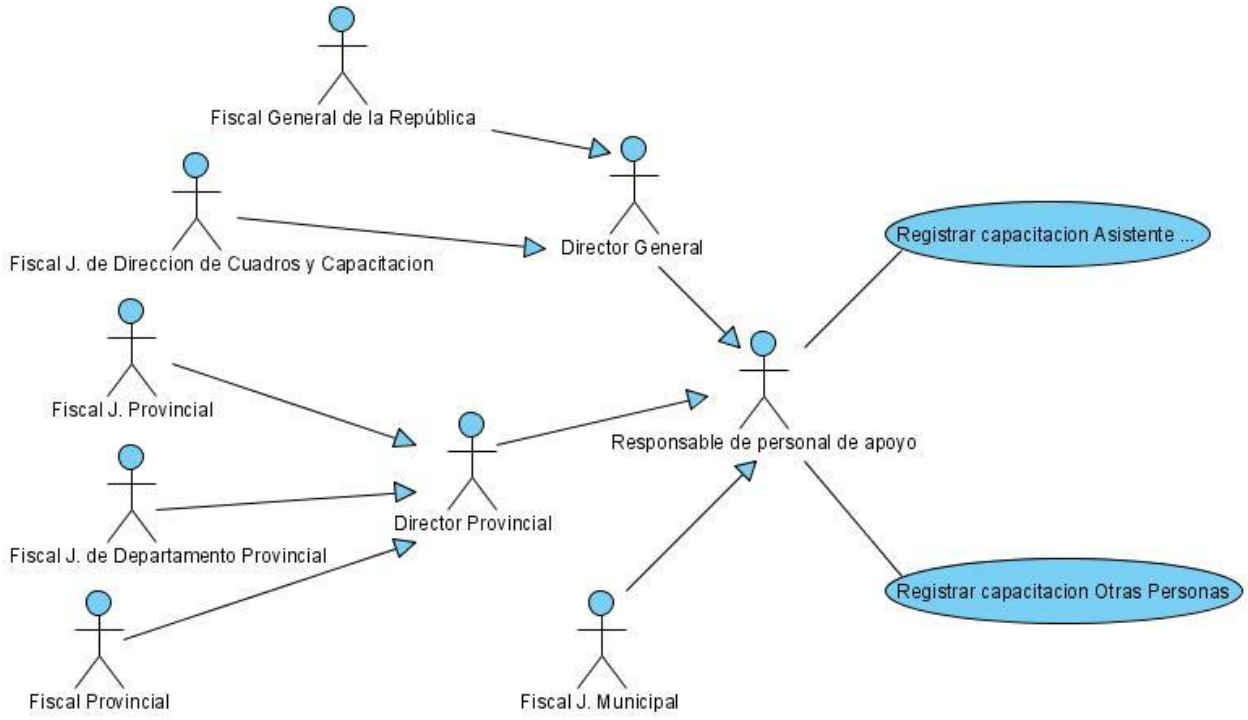
Registrar datos laborales del Cuadro: En este caso de uso se introducen los datos laborales de los cuadros para la creación del expediente de los mismos, pero además estos datos pueden ser modificados.

Registrar datos de reserva y evaluación del Cuadro: En este caso de uso se introducen los datos de reserva y evaluación de los cuadros, los cuales pueden ser modificados. Además se pueden gestionar los datos de los cuadros que constituyen la reserva. Además se da una evaluación anual del cuadro.

Registrar Permisos: Permite al Responsable de Cuadro del sistema registrar los permisos de los usuarios.

Búsqueda simple: En este caso de uso se realiza la búsqueda de los cuadros que se encuentran registrados en el sistema, a través del nombre, apellido, CI o ambos.

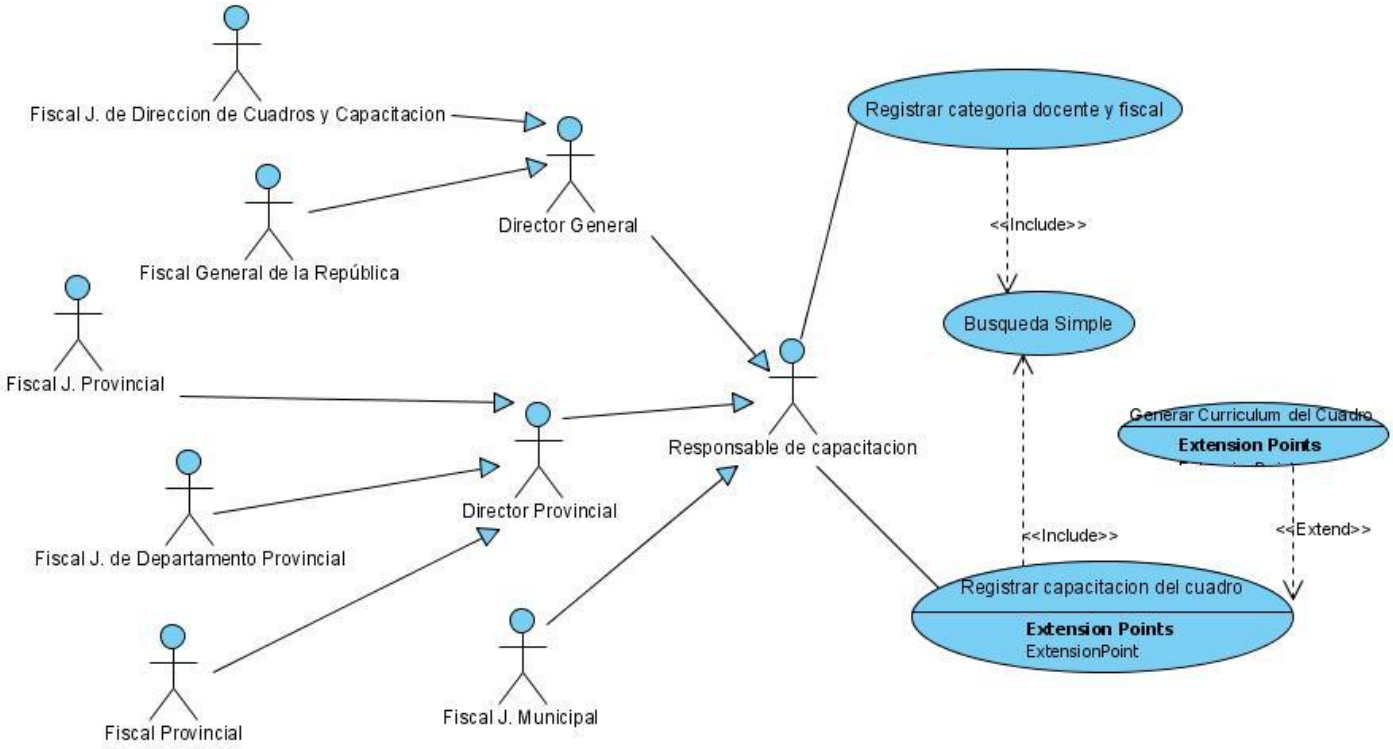
2.7.2. Casos de uso arquitectónicamente significativos del módulo Personal de Apoyo:



Registrar datos de captación de asistente fiscal: En este caso de uso se registran los datos personales y datos relacionados con la carrera universitaria, de los asistentes fiscales.

Registrar datos de captación de otras personas: En este caso de uso se registran los datos de otras personas no relacionadas con las leyes que trabajan en la fiscalía.

2.7.3. Casos de uso arquitectónicamente significativos del módulo Capacitación:

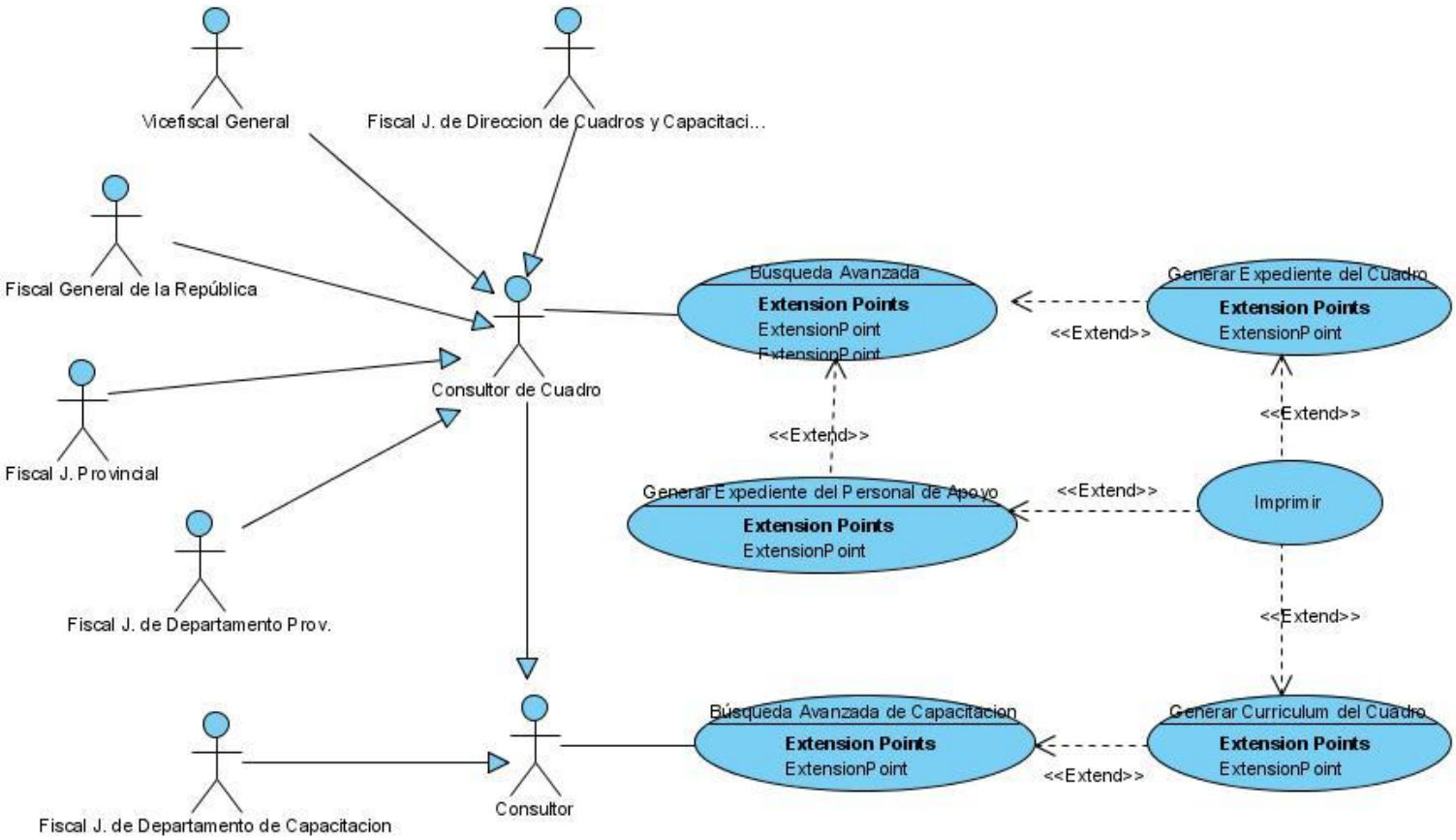


Registrar categoría docente y fiscal: En este caso de uso de registran los datos relacionados con la categoría docente y categoría fiscal. Luego se registra si el cuadro pertenece a la Unión de Juristas de Cuba, y la responsabilidad que ocupa.

Registrar capacitación del cuadro: En este caso de uso se introducen los datos relacionados con la capacitación del cuadro. Componente de preparación y superación, acciones de capacitación correspondientes y se introducen los datos de las mismas. Por último se da la opción de generar un currículo del cuadro.

Generar currículo del cuadro: En este caso de uso se genera un reporte con los principales componentes de la capacitación del cuadro y algunos datos personales del mismo.

2.7.4. Casos de uso arquitectónicamente significativos del módulo Reportes y Búsquedas:



Generar expediente del personal de apoyo: En este caso de uso se genera un reporte con alguno de los datos del personal de apoyo. Se puede imprimir el mismo en caso de ser necesario.

Generar expediente del cuadro: En este caso de uso se genera un reporte con los datos personales, datos laborales, datos de reserva y evaluación del Cuadro, registrados con anterioridad. Además se muestran los aspectos relacionados con la capacitación del mismo. Luego se puede imprimir el reporte.

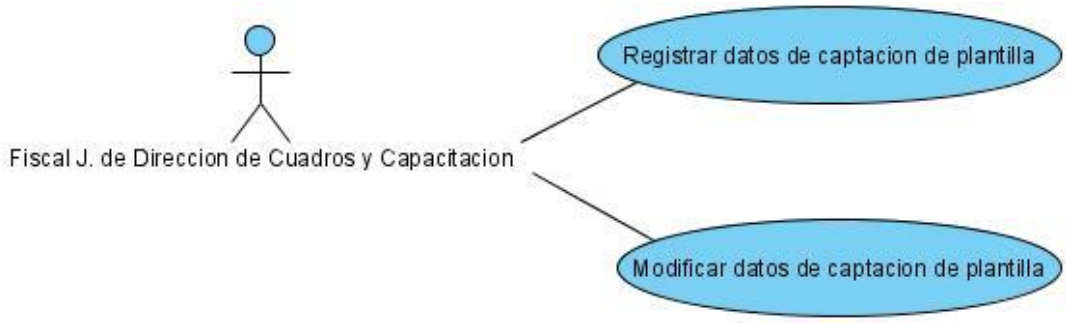
Búsqueda avanzada: Se pueden realizar búsquedas personalizadas relacionadas con los datos del cuadro ó personal de apoyo que se encuentran registrados en el sistema. Luego el sistema muestra un listado con las personas que coinciden y da la posibilidad de acceder al expediente.

Búsqueda avanzada de Capacitación: Se pueden realizar las búsquedas personalizadas relacionadas con los datos de la capacitación de los Fiscales. Luego el sistema muestra un listado con las personas que coinciden y da la posibilidad de acceder al currículo de los mismos.

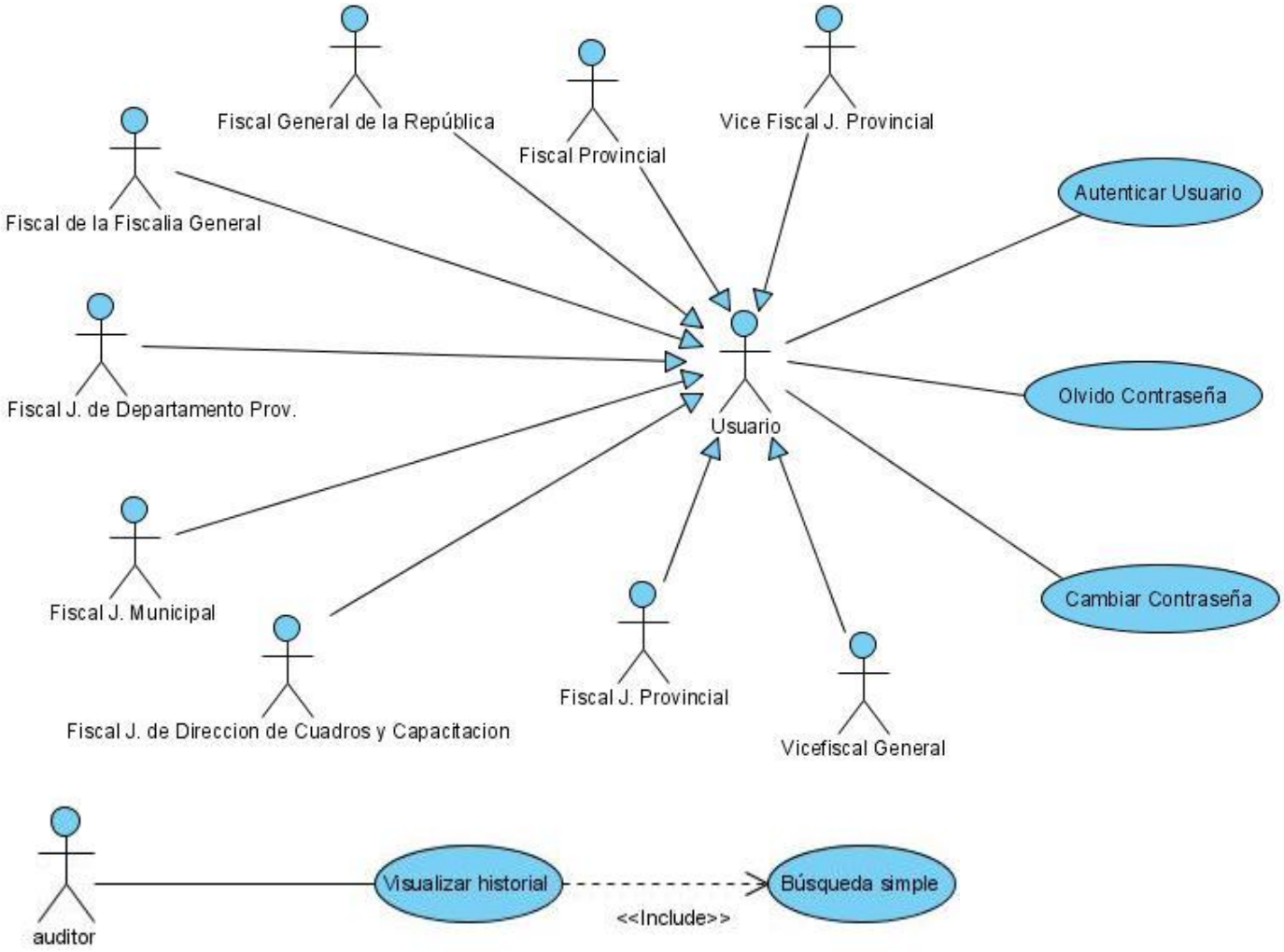
Generar currículo del Cuadro: Se genera un reporte con los principales elementos de la capacitación del cuadro y algunos datos personales del mismo. Se puede imprimir el reporte en caso de ser necesario.

Imprimir: En este caso de uso se imprimen los reportes de la aplicación, que incluye currículos, expedientes del cuadro, del personal de apoyo, y además otros reportes de interés.

2.7.5. Casos de uso arquitectónicamente significativos del módulo Captación de Plantilla:



2.7.6. Casos de uso arquitectónicamente significativos del módulo Generalidades:



2.8. Vista de Implementación

Según RUP esta vista se realiza a través del diagrama de componentes el cual muestra los componentes software que constituyen una parte reusable, sus interfaces y sus interrelaciones. No se pretende con el diagrama especificar todos los componentes que forman parte de la aplicación solo aquellos que de manera general engloban un conjunto de componentes que son imprescindibles para el desarrollo del sistema.

Para la realización de dicho diagrama se asignaron responsabilidades de modo que la cohesión se mantenga alta y bajo el acoplamiento para así obtener un diseño más reutilizable y mantenible. La selección de los componentes empezó con los elementos y subsistemas más significativos encontrados en el diseño, además se agregaron otros componentes de interés para la arquitectura como el *framework*, y algunas librerías que este utiliza.

Todos los subsistemas y módulos que componen la aplicación comparten la misma estructura, a continuación se muestra el diagrama de componentes genérico, que representa las relaciones entre los componentes básicos de la aplicación, aunque existen específicos para algunos módulos que son los menos, como es el caso del módulo Reportes y Búsquedas que además utiliza componentes para exportar a formato PDF los reportes y expedientes.

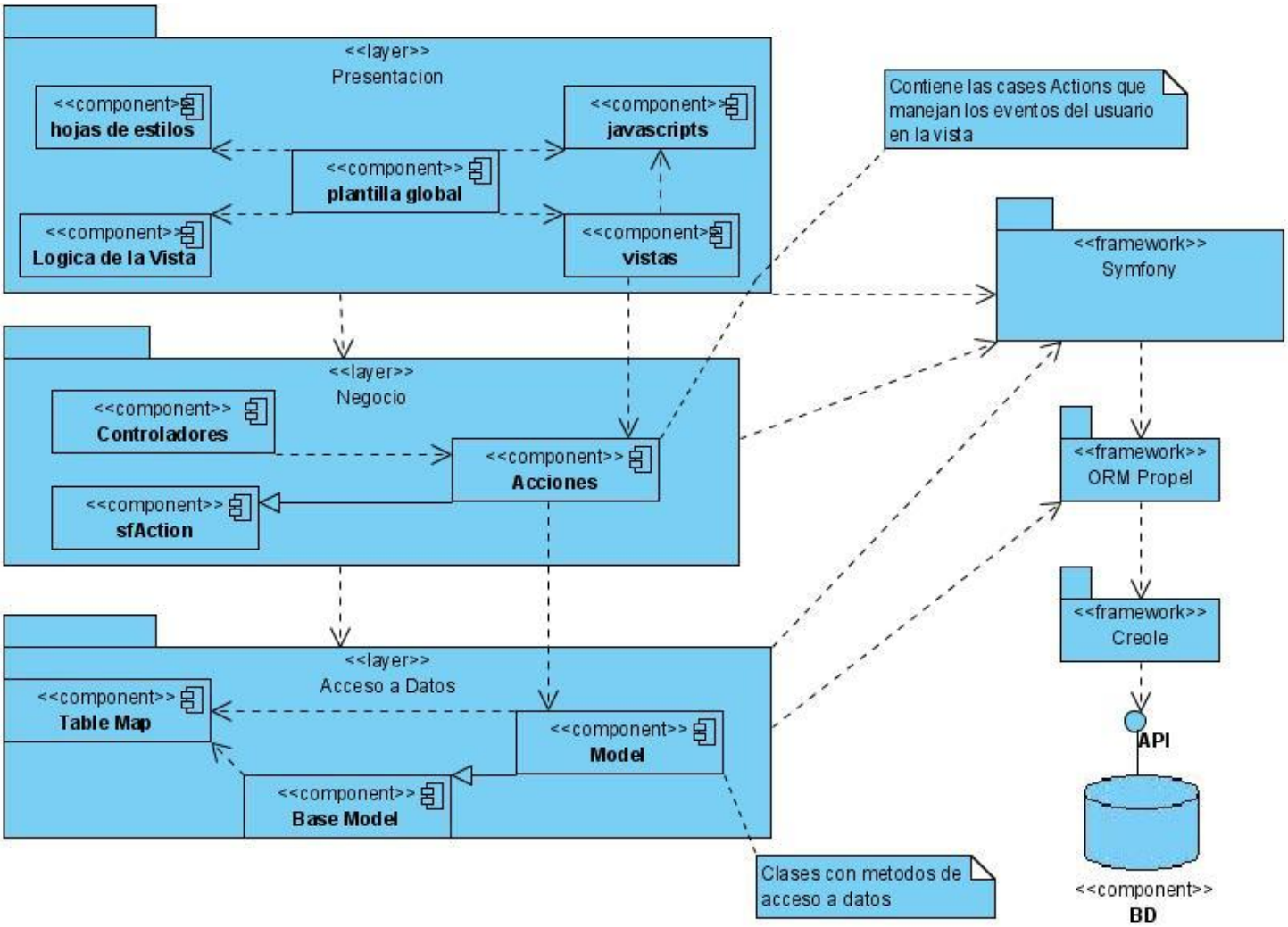


Figura 2-5. Diagrama de componentes del subsistema GCPA.

El módulo Reportes y Búsquedas hace uso exclusivo de componentes para la realización del Caso de Uso Imprimir, dichos componentes influyen en gran medida en el rendimiento del sistema, por interés para la arquitectura se mostrará en el diagrama a continuación la relación de dichos componentes con la aplicación para un mejor entendimiento.

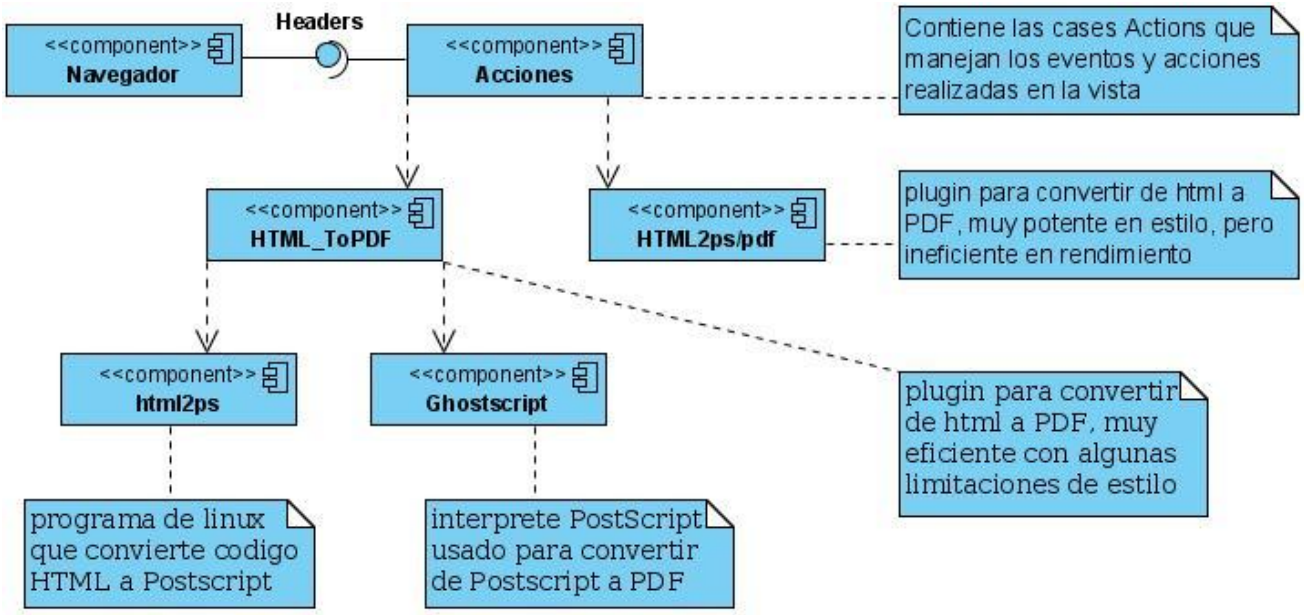


Figura 2-6. Vista de Implementación de los elementos arquitectónicamente significativos del caso de uso Imprimir.

2.9. Vista lógica

RUP define una vista lógica para comprender la estructura y organización del diseño del sistema que representa las realizaciones de los casos de usos, subsistemas, paquetes y clases arquitectónicamente significativas. La vista lógica se perfecciona durante las iteraciones sucesivas.

Debido a la gran envergadura del proyecto el cual está compuesto por 41 módulos, se decidió hacer una representación lógica de alto nivel y agrupar las clases en paquetes y a su vez estos paquetes en subsistemas funcionales.

El sistema está dividido en subsistemas que representan las esferas de trabajo en las que se organizan los fiscales, y en cada una de estas esferas están definidos los procesos que le competen, estos procesos representan los módulos que son los que se relacionan brindando interfaces bien definidas, ya que los subsistemas no se relacionan entre ellos. Para una mejor comprensión de la vista lógica se hará

referencia a los módulos como subsistemas funcionales indistintamente en los casos que se crea necesario.

Esta vista abarca la primera iteración del proyecto, que incluye el desarrollo del subsistema GCPA y parte del subsistema HCTA, los mismos fueron considerados por la FGR de mayor interés para su inmediata realización.

2.9.1. Dependencia entre los subsistemas funcionales

Gestión de Cuadros y Personal de Apoyo (GCPA): Garantiza la gestión de los cuadros (fiscales) y el personal de apoyo (trabajadores no fiscales y estudiantes de derecho que hacen las prácticas en la fiscalía). Implica la gestión de datos de cuadros, gestión de la capacitación de cuadros, movimiento de cuadros dentro de la fiscalía. Además de la gestión de los datos del personal de apoyo que están vinculado a la fiscalía. Garantiza los niveles de seguridad y acceso tanto de los cuadros como del personal de apoyo.

El Subsistema GCPA está formado por los siguientes módulos:

- Gestión de Cuadros.
- Gestión del Personal de Apoyo.
- Capacitación.
- Reportes y Búsquedas.
- Generalidades.
- Captación de Plantillas.

Herramientas Comunes a Todas las Áreas (HCTA): Organiza los subsistemas de la aplicación, es el punto de partida de la aplicación, es el módulo que va a engranar todos los subsistemas, a partir de él los usuarios pueden acceder a los demás subsistemas. Engloba toda una serie de herramientas comunes que estarán inmersas en el resto de los subsistemas como son el Diccionario Jurídico, Gestor de Normativas Jurídicas.

El Subsistema HCTA está formado por los siguientes módulos:

- Módulo Diccionario Jurídico.
- Módulo Gestor de Nomencladores.
- Módulo Motor de Dictados.
- Módulo Herramientas de Análisis de Contenido.
- Módulo Cliente de Correo.

A continuación se muestra la vista lógica de los subsistemas, es una representación de subsistemas que a su vez contienen otros subsistemas o módulos y las relaciones entre estos últimos.

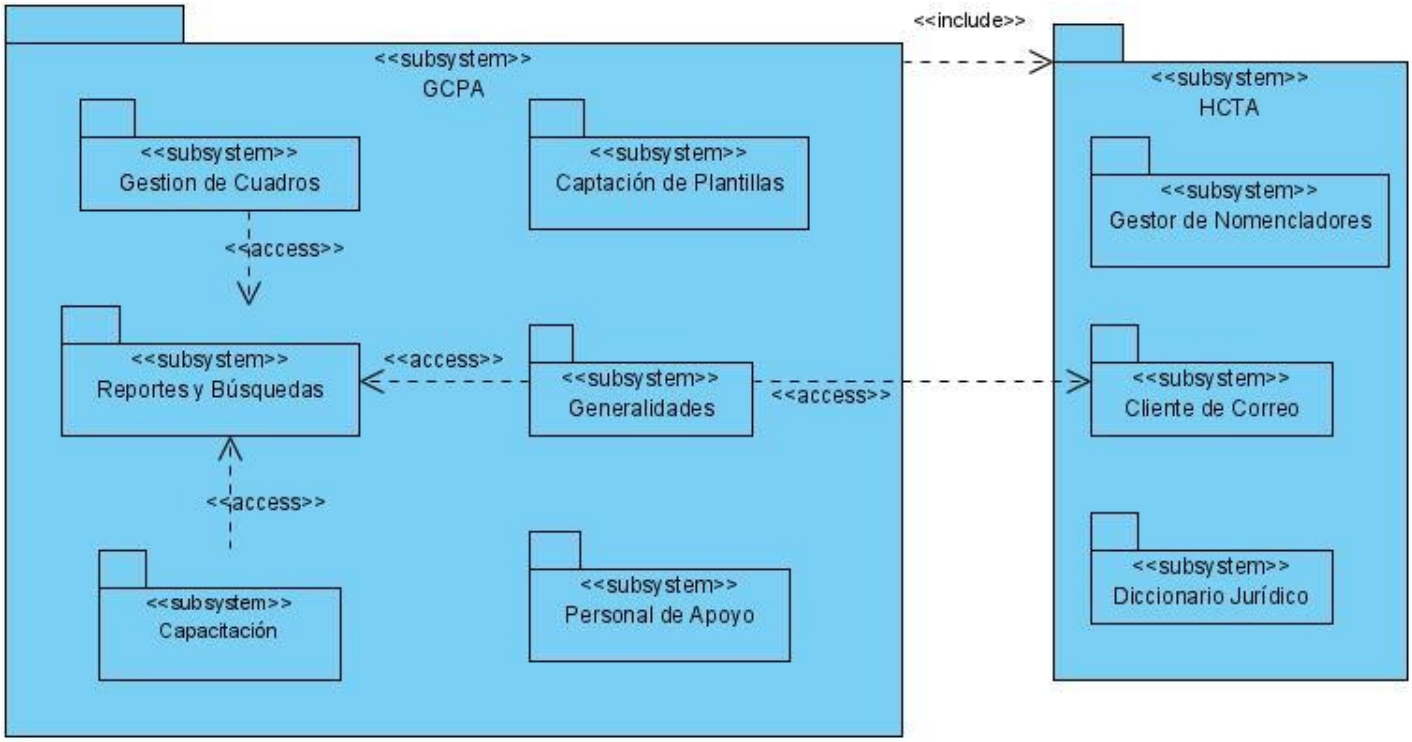


Figura 2-7. Vista de Lógica de los subsistemas funcionales correspondientes a la primera iteración.

2.9.2. Distribución física de los paquetes

Cada uno de los subsistemas comparte la misma estructura de paquetes e implementaran una arquitectura en capas. Esta estructura en forma de árbol es proporcionada por el framework para la organización de los componentes dentro de la aplicación, está pensada para fomentar la reutilización y evitar las duplicaciones del código. En la siguiente figura se muestra la distribución física de los paquetes y algunos componentes que conforman el sistema.

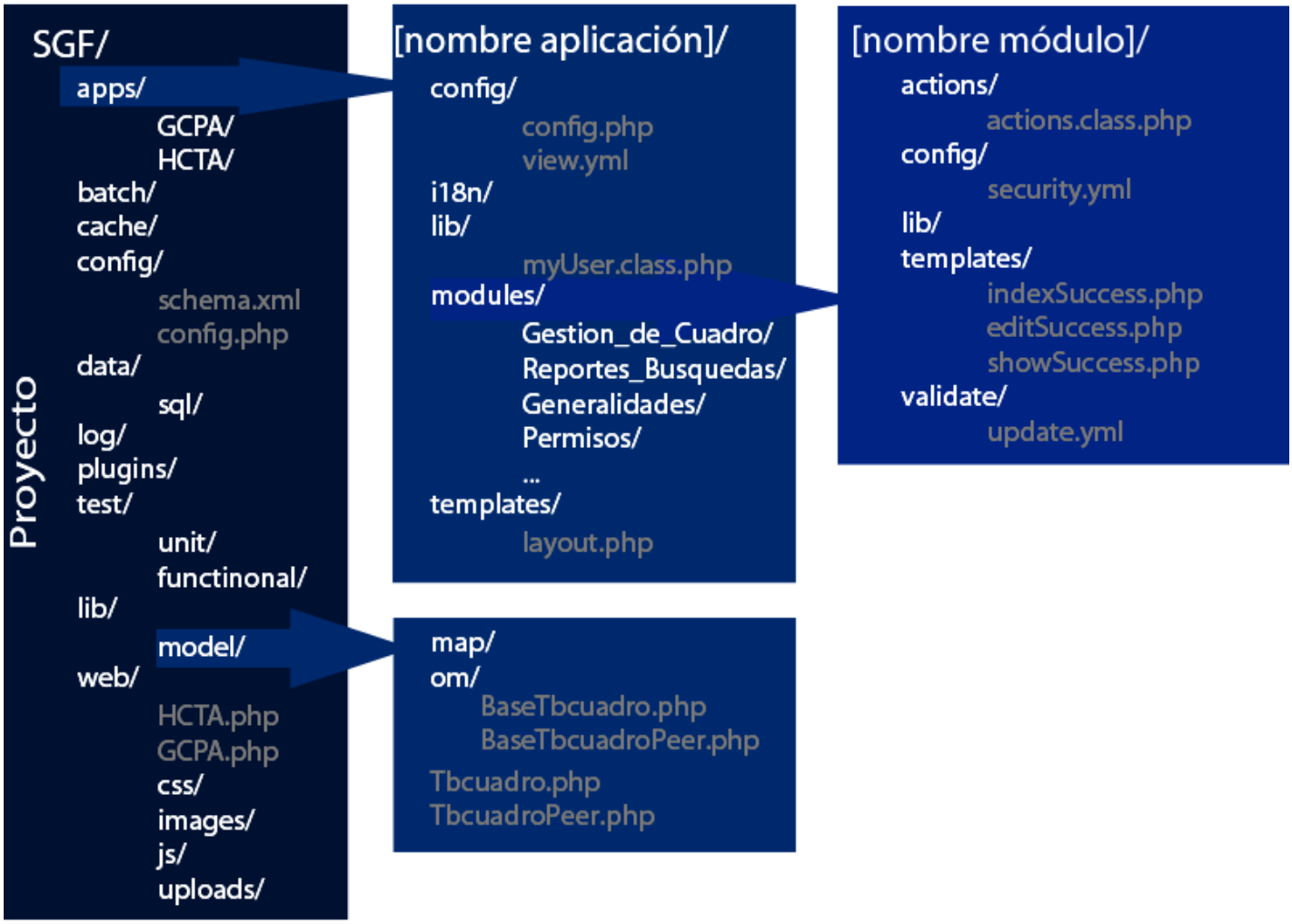


Figura 2-8. Vista de física de los paquetes que forman la estructura del proyecto.

En los proyectos de Symfony las operaciones se agrupan de forma lógica en aplicaciones. Cada aplicación está formada por uno o más módulos. Un módulo normalmente representa un grupo de páginas con un propósito relacionado. Los módulos almacenan las acciones, que representan cada una de las operaciones que se puede realizar en un módulo. Generalmente una acción representa una página. A continuación se describen en forma de tablas los paquetes que forman el proyecto:

Tabla 2-1. Directorios o paquetes en la raíz del proyecto.

Directorio	Descripción
apps/	Contiene un directorio por cada aplicación del proyecto (normalmente, *frontend* y *backend* para la parte pública y la parte de gestión respectivamente).
batch/	Contiene los scripts de PHP que se ejecutan mediante la línea de comandos o mediante la programación de tareas para realizar procesos en lotes (*batch processes*)
cache/	Contiene la versión cacheada de la configuración y (si está activada) la versión cacheada de las acciones y plantillas del proyecto. El mecanismo de caché utiliza los archivos de este directorio para acelerar la respuesta a las peticiones web. Cada aplicación contiene un subdirectorio que guarda todos los archivos PHP y HTML preprocesados.
config/	Almacena la configuración general del proyecto en archivos con extensión *.yml*
data/	En este directorio se almacenan los archivos relacionados con los datos, como por ejemplo el esquema de una base de datos, el archivo que contiene las instrucciones SQL para crear las tablas e incluso un archivo de bases de datos de SQLite.
lib/	Almacena las clases y librerías externas. Se suele guardar todo el código común a todas las aplicaciones del proyecto. El subdirectorio *model/* guarda el modelo de objetos del proyecto.
log/	Guarda todos los archivos de log generados por Symfony. También se puede utilizar para guardar los *logs* del servidor web, de la base de datos o de cualquier otro componente del proyecto. Symfony crea un archivo de log por cada aplicación y por cada entorno.
plugins/	Almacena los plugins instalados en la aplicación.
test/	Contiene las pruebas unitarias y funcionales escritas en PHP y compatibles con el *framework* de pruebas de Symfony. Cuando se crea un proyecto, Symfony crea algunas pruebas básicas.
web/	La raíz del servidor web. Los únicos archivos accesibles desde Internet son los que se

	encuentran en este directorio.

Tabla 2-2. Subdirectorios de cada subsistema o aplicación.

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

Tabla 2-3. Subdirectorios de cada módulo o subsistema.

Directorio	Descripción
actions/	Normalmente contiene un único archivo llamado *actions.class.php* y que corresponde a la clase que almacena todas las acciones del módulo. También es posible crear un archivo diferente para cada acción del módulo.
config/	Puede contener archivos de configuración adicionales con parámetros exclusivos del Módulo.
lib/	Almacena las clases y librerías utilizadas exclusivamente por el módulo.
templates/	Contiene las plantillas correspondientes a las acciones del módulo. Cuando se crea un nuevo módulo, automáticamente se crea la plantilla llamada *indexSuccess.php*
validate/	Contiene archivos de configuración relacionados con la validación de formularios.

2.10. Vista de Despliegue

El objetivo principal de esta vista es la descentralización de la aplicación, por varias razones: primero le permite a la FGR aprovechar mejor el ancho de banda, ya que solo se cuenta con 2 MB/s, y segundo aumenta la disponibilidad del sistema, teniendo en cuenta que la entidad ETECSA que es la que provee la conectividad no asegura la confiabilidad de la infraestructura de red, lo cual no es factible ya que casi todo el trabajo de los fiscales se debe hacer sobre la aplicación.

Para garantizar que la información sea la misma para todas las sedes se diseño un sistema de replicación multimaestro asíncrono *PgCluster*[16] , que replicará los datos modificados en una instancia hacia todas las bases de datos, a una determinada hora del día, preferiblemente de noche (ver Anexo2).

A pesar de las innumerables ventajas de la descentralización también trae consigo el problema de la seguridad en cada uno de estos servidores, problema que no depende totalmente de la arquitectura del sistema, aquí juega un papel fundamental la configuración de estos servidores, la utilización de un firewall como el *iptables* y la seguridad del local.

.

[16] Aplicación de Linux para hacer réplica de bases de datos entre servidores PostgreSQL

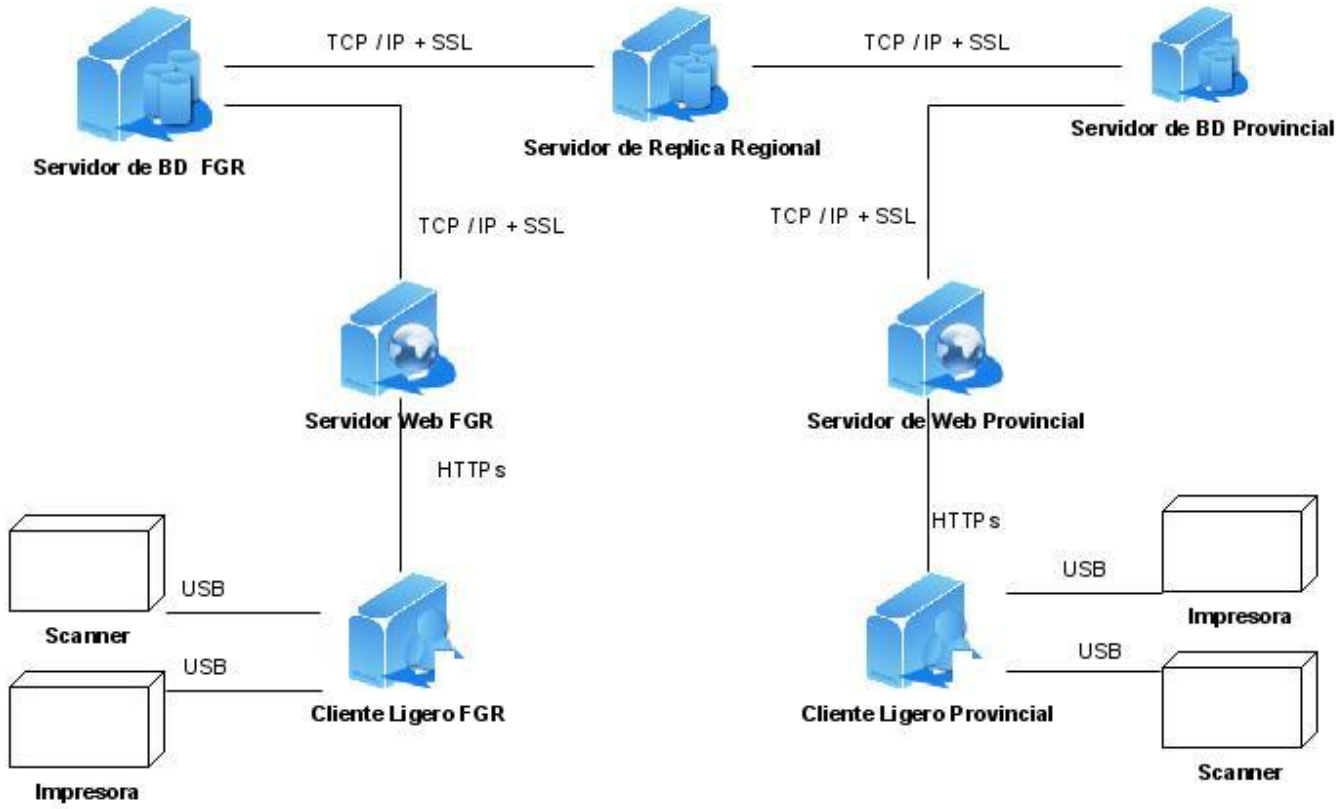


Figura 19. Diagrama de despliegue del Sistema de Gestión Fiscal.

A continuación se describen los diferentes nodos y dispositivos que contiene la figura anterior:

- **Nodos:**

Todos los Nodos Servidores que se utilizarán son de fabricación *Hewlett-Packard* (HP) y todos poseen las mismas características técnicas, son servidores profesionales corriendo es sistema operativo Debian GNU/Linux 4.0r7, con capacidades de hardware de 4GB de Memoria RAM, 2.66 GHz de velocidad de procesamiento y 3x146GB de almacenamiento.

Servidor de Réplica Regional: Nodo que se encarga de la réplica de los datos entre los servidores de base de datos provinciales en una región del país, uno por cada región: occidental, central y oriental. Cada nodo utiliza un sistema replicador multi-maestro PgCluster.

Servidor de BD Provincial y FGR: Nodo que se encarga en cada provincia incluyendo la FGR, de servir la base de datos, a través del SGBD PostgreSQL 8.3, el cual va a permitir acceso solo de la aplicación provincial. La comunicación con la aplicación es cifrada utilizando el puerto 5432.

Servidor de Web Provincial y FGR*:* Nodo que se encarga en cada provincia incluyendo la FGR, de servir la aplicación, a través de Apache2, que proveerá acceso a todos los clientes de dicha provincia a través del protocolo seguro HTTPs.

Cliente Ligero Provincial y FGR: Nodo que se encarga de conectarse con la aplicación, el cual provee las siguientes características de Hardware: Intel Pentium 3 a 1.13GHz, memoria RAM de 256MB y adaptador Ethernet 100MBps. Estos se conectan a la aplicación por medio de protocolo seguro HTTPs.

- **Dispositivos**:

Impresora: Este dispositivo garantiza las funcionalidades de impresión de reportes y documentos que genera la aplicación, también de tecnología Hewlett-Packard.

Scanner: Este dispositivo garantiza las funcionalidades para escanear documentos que deben guardarse digitalmente en la aplicación, también de tecnología Hewlett-Packard.

2.11. Conclusiones

En el presente capítulo se describió la arquitectura del Sistema de Gestión Fiscal, que es la entrada para el artefacto documento de arquitectura de software, donde se seleccionaron las tecnologías para el desarrollo del sistema, se propuso la metodología de desarrollo, se definió el estilo arquitectónico del sistema, así como también la plataforma y el framework de desarrollo a utilizar. Se elaboraron un conjunto de vistas que ayudan a una mejor representación de la arquitectura, las cuales se irán refinando en iteraciones posteriores.

CAPÍTULO 3

3. ANÁLISIS Y VALIDACIÓN DE LA ARQUITECTURA

3.1. Introducción

La calidad del software depende de los resultados de la evaluación de la arquitectura. Cuya función es determinante para la selección entre las distintas arquitecturas candidatas y la certificación de la factibilidad de la línea base, siendo el componente clave de las iteraciones arquitectónicas exitosas.

Existen técnicas para evaluar una arquitectura las cuales analizan y prueban el diseño enfocándose en los atributos de calidad o casos de uso arquitectónicamente significativos. Para llevar a cabo la evaluación de la arquitectura propuesta se analizarán varias técnicas y métodos que permiten validar el diseño contra las cualidades deseadas. Esta evaluación permitirá mitigar riesgos en etapas tempranas del diseño, como estrategia para la reducción de costos de mantenimiento en etapas posteriores.

3.2. Objetivos

El objetivo de la validación de la arquitectura del Sistema de Gestión Fiscal es demostrar el cumplimiento de los requerimientos y cualidades arquitectónicas, asegurando que el sistema a ser construido cumple con las necesidades de los clientes. Algunas de estas cualidades se refieren al diseño, sin embargo tienen su repercusión en la arquitectura.

3.3. ¿Por qué evaluar una Arquitectura?

Dado que la arquitectura es un producto temprano de la fase de diseño, su evaluación permitirá verificar el cumplimiento de los requisitos no funcionales y detectar errores que en etapas posteriores resultarían

más costosos en tiempo y recursos. Determina como se deben cumplir los atributos de calidad del sistema, lo cual deviene en un marcado interés la validez de la propuesta arquitectónica.

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

3.4. ¿Cuándo una Arquitectura puede ser evaluada?

En general, una evaluación debe realizarse cuando hay suficiente de la arquitectura como para justificarlo. La arquitectura de software puede ser evaluada en cualquier etapa del desarrollo del software. Una buena estrategia es efectuar una evaluación cuando el equipo de desarrollo comienza a tomar decisiones que dependen de la arquitectura y el costo de deshacerlas sobrepasa al costo de realizar una evaluación.

Existen dos variantes que abarcan todas las etapas en que una arquitectura puede ser evaluada, estas son evaluación temprana y evaluación tardía.

Evaluación temprana: Para la evaluación no es necesario que la arquitectura este totalmente especificada. Esta puede ser realizada en cualquier etapa del proceso de creación de la arquitectura, para examinar las decisiones arquitectónicas ya tomadas y decidir entre las opciones que están pendientes. Mientras mayor sea el nivel de especificación, mejores serán los resultados arrojados por la evaluación.

Evaluación tardía: Consiste en realizar la evaluación una vez que la arquitectura esta terminada y se ha completado la implementación. Por lo general tiene lugar cuando se desea comprobar que el sistema ya desarrollado cumple con los requerimientos de calidad propuestos y su comportamiento es el deseado.

3.5. ¿Qué resultado produce la evaluación de una Arquitectura?

La evaluación de la arquitectura produce un informe, la forma y contenido del mismo varía según la etapa y método utilizado. Una evaluación define y prioriza los objetivos que debe cumplir la arquitectura para poder considerarse adecuada. Por lo general genera respuestas a las siguientes interrogantes:

- ¿Es la arquitectura adecuada para el sistema para el cual fue diseñada?
- ¿Cuál de dos o más arquitecturas propuestas es la más adecuada para el sistema?

Una arquitectura adecuada es aquella donde el sistema resultante de esta cumple con los objetivos de calidad que dependen de la arquitectura, y si el sistema puede ser construido con los recursos disponibles: equipo de trabajo, presupuesto, etc.

Por lo general no produce resultados cuantitativos, mas identifica las debilidades y propone pasos para mitigarlas, la evaluación determina como un atributo de calidad es afectado por una decisión de diseño arquitectónico.

3.6. Atributos de calidad

Según la estandarización de la [IEEE Std 610.12-1990] la calidad del software es la medida en que se logra la combinación deseada de atributos (Interoperabilidad, Confiabilidad, etc.) en los procesos del software. Dado que muchos de estos atributos son responsabilidad de la arquitectura a continuación se describen aquellos que forman parte de los objetivos inmediatos de la arquitectura del Sistema de Gestión Fiscal en su primera iteración.

Seguridad: Podemos entender como seguridad un estado de cualquier tipo de información (informático o no) que nos indica que ese sistema está libre de riesgo. Para la mayoría de los expertos el concepto de seguridad en la informática es utópico porque no existe un sistema 100% seguro. Para que un sistema se pueda definir como seguro debe tener estas cuatro características:

- Integridad: La información sólo puede ser modificada por quien está autorizado y de manera controlada.
- Confidencialidad: La información sólo debe ser legible para los autorizados.
- Disponibilidad: Debe estar disponible cuando se necesita.
- Irrefutabilidad (No repudio): El uso y/o modificación de la información por parte de un usuario debe ser irrefutable, es decir, que el usuario no puede negar dicha acción.

Flexibilidad: Es la capacidad de un sistema para adaptarse a diferentes entornos y condiciones, y hacer frente a los cambios en las reglas del negocio. Un sistema flexible es uno que pueda ser fácilmente modificado para dar respuesta a nuevos requerimientos del usuario o del sistema.

Mantenibilidad: Es la capacidad de un sistema de resistir a los cambios en sus componentes, servicios, características e interfaces que puedan ser necesarios al añadir o cambiar una funcionalidad, reparar un error, o encontrar nuevos requerimientos del negocio. Puede ser medida en términos del tiempo necesario para restaurar el sistema a su estado operacional después de un fallo producido por una actualización del sistema.

Escalabilidad: A diferencia de los tres atributos anteriores este influye en la ejecución del sistema. Es la facilidad con la que un sistema o componente puede modificarse para aumentar su capacidad funcional o de almacenamiento. El propósito es mantener el rendimiento, disponibilidad y confiabilidad del sistema aun cuando aumente la carga. Existen dos métodos para mejorar la escalabilidad: escalabilidad vertical y escalabilidad horizontal.

- Escalabilidad vertical: Se logra adicionando más recursos de procesamiento, memoria, disco, etc. a un único sistema para escalar verticalmente.
- Escalabilidad horizontal: Se logra adicionando más servidores para la aplicación.

3.7. Técnicas de Evaluación de Arquitectura de Software

Con el objetivo de tomar decisiones desde el punto de vista arquitectónico y principalmente en fases tempranas de desarrollo, se hace necesario implementar técnicas que requieran poca información y a su vez, posibiliten la obtención de resultados relativamente precisos.

Evaluación basada en escenarios: Un escenario es una breve descripción de la interacción de un actor del sistema con éste y cuenta de tres partes (estímulo, contexto y respuesta). El estímulo describe la interacción con el sistema (ejecución de tareas, cambio de configuración). El contexto describe qué sucede en el sistema ante un determinado estímulo. La respuesta describe a través de la arquitectura, cómo debe responder el sistema ante el estímulo, lo que posibilita asociarlo con determinado atributo de calidad.

Evaluación basada en simulación: Consiste en la evaluación del comportamiento de la propuesta arquitectónica bajo determinadas circunstancias. Para ello se utilizan componentes implementados con cierto nivel de abstracción, dentro del contexto en el cual se espera sean ejecutados. Los pasos que sigue este proceso son:

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

Evaluación basada en modelos matemáticos: Permite la evaluación de atributos de calidad operacionales y los modelos de diseño arquitectónico. De igual manera puede utilizarse para la evaluación por simulación, ya que ambas evalúan el mismo tipo de atributo por lo que ambos enfoques pueden encontrase combinados. Los pasos que sigue esta técnica para la evaluación son los siguientes:

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

Evaluación basada en experiencia: Como su nombre lo indica, la técnica para la evaluación arquitectónica basada en la experiencia, se centra en el conocimiento que pueden aportar los especialistas durante el desarrollo del proyecto. Estos conocimientos constituyen por lo general valiosas herramientas en la adopción de decisiones aceptadas.

3.8. Métodos de Evaluación de Arquitecturas de Software

SAAM (*Software Architecture Analysis Method*): El Método de Análisis de Arquitecturas de Software es el primer método basado en escenarios que surgió. Originalmente fue creado para el análisis de modificabilidad de la arquitectura, pero ha demostrado ser muy útil para evaluar ciertos atributos de calidad, tales como modificabilidad, portabilidad, escalabilidad e integrabilidad (Clements, et al., Oct 22, 2001). Este consiste en la enumeración de escenarios que representarán los probables cambios a los que el sistema estará sometido en el futuro. Como entrada principal necesitará la descripción de la arquitectura de dicho sistema, la cual será evaluada.

La evaluación de este método está compuesta por seis (6) pasos, los cuales son:

1. Desarrollo de escenarios.
2. Descripción de la arquitectura.
3. Clasificación y asignación de la prioridad de los escenarios.
4. Evaluación individual de los escenarios indirectos.
5. Evaluación de la interacción entre los escenarios.
6. Creación de la evaluación global.

ATAM (*Architecture Trade-off Analysis Method*): El Método de Análisis de Acuerdos de Arquitectura revela la forma en que una arquitectura satisface ciertos atributos de calidad, provee una visión de cómo estos atributos interactúan con otros. ATAM está centrado en tres aéreas distintas, estilos arquitectónicos, el análisis de los atributos de calidad y el método SAAM.

El método de evaluación ATAM comprende nueve pasos, agrupados en cuatro fases:

Fase 1: Presentación.

1. Presentación del ATAM.
2. Presentación de las metas del negocio.
3. Presentación del negocio.

Fase 2: Investigación y análisis.

4. Identificación de los enfoques arquitectónicos.
5. Generación del *Utility Tree*[17].
6. Análisis de los enfoques arquitectónicos.

Fase 3: Pruebas.

7. Tormenta o lluvia de ideas y establecimientos de la prioridad de los escenarios.
8. Análisis de los enfoques arquitectónicos.

Fase 4: Reportes.

9. Presentación de los resultados.

ARID (*Active Reviews for Intermediate Designs*): El método de Revisiones Activas para Diseños Intermedios permite realizar evaluaciones a diseños parciales en etapas tempranas dentro del desarrollo del proyecto. ARID es un híbrido entre *Active Design Review* (ADR) y ATAM. Tanto ADR (*Active Design Review*) como ATAM proveen características útiles para el problema de la evaluación de diseños preliminares.

En el caso de ADR, proporciona una documentación detallada del diseño y completan cuestionarios, cada uno por separado. En el caso de ATAM, está orientado a la evaluación de toda una arquitectura. Ante la necesidad de evaluación en las fases tempranas del diseño, expertos proponen la utilización de ARID como una combinación de estos métodos.

[17] Utility tree es un esquema en forma de árbol que representa los atributos de calidad de un sistema de software, refinados hasta el establecimiento de escenarios que especifican suficiente detalle la prioridad de cada uno

El método ARID comprende nueve pasos agrupados en dos fases:

Fase 1: Actividades previas.

1. Identificación de los encargos de la revisión.
2. Preparar el informe de diseño.
3. Preparar los escenarios bases.
4. Preparar los materiales.

Fase 2: Revisión.

5. Presentación del método ARID.
6. Presentación del diseño.
7. Lluvia de ideas y establecimiento de prioridad de escenarios.
8. Aplicación de los escenarios.
9. Resumen.

3.9. Evaluación de la Arquitectura del Sistema de Gestión Fiscal

No viene al caso comparar dichos método en una escala de mejor o peor, en cambio si se puede afirmar que en determinadas condiciones y para determinados atributos de calidad existen métodos que evalúan la arquitectura de manera más eficiente que otros.

Teniendo en cuenta que la arquitectura es un producto temprano del diseño, se selecciona ARID como método para evaluar la arquitectura ya que permite realizar evaluaciones a diseños parciales en etapas tempranas del desarrollo, además aprovecha las mejores características de otros métodos, ya que es un híbrido entre ATAM y ADR.

Para la evaluación se seleccionaron los escenarios claves para la arquitectura, aunque existen muchos más escenarios que también aportan cualidades a la arquitectura, los cuales serán refinados en iteraciones posteriores. A continuación se evalúan las decisiones arquitectónicas con respecto a su impacto sobre los atributos de calidad en los principales escenarios de la aplicación:

Atributos de calidad	Perfil	Escenario

Reusabilidad	Reutilización	Acceso a datos
Relación atributo-escenario		
El *framework* Symfony utiliza el ORM Propel que realiza el acceso y la modificación de los datos de la aplicación mediante objetos, permitiendo llamar a los métodos de un objeto de datos desde varias partes de la aplicación e incluso desde diferentes aplicaciones.		

Atributos de calidad	**Perfil**	**Escenario**
Portabilidad	Portabilidad	Acceso a datos
Relación atributo-escenario		
ORM Propel permite crear las consultas mediante una sintaxis independiente de la base de datos y un componente externo, en este caso Creole, se encarga de traducirlas y optimizarlas al lenguaje SQL concreto de la base de datos.		

Atributos de calidad	**Perfil**	**Escenario**
Mantenibilidad	Mantenimiento	Abstracción de la bases de datos
Relación atributo-escenario		
ORM Propel utiliza la librería Creole, que se encarga de la abstracción de la base de datos, posibilitando la migración a los gestores de base de datos más usados sin adicionar ni una línea de código, solo se necesita cambiar un parámetro en un archivo de configuración.		

Atributos de calidad	**Perfil**	**Escenario**
Seguridad Interna	Seguridad	Interacción con la aplicación
Relación atributo-escenario		
Symfony provee mecanismos para la restricción de acceso al nivel más mínimo, donde cada acción antes de ser ejecutada pasa por un filtro especial que verifica si el usuario tiene privilegios para acceder a las misma. Estos privilegios están compuestos por dos partes: ❖ Las acciones seguras requieren que los usuarios estén autenticados. ❖ Las credenciales son privilegios de seguridad agrupados bajo un nombre y permiten organizar la seguridad en grupos.		

Atributos de calidad	Perfil	Escenario
Reusabilidad	Reutilización	Reutilización de componentes
Relación atributo-escenario		
El framework Symfony permite convertir acciones en **componentes** (Un componente es como una acción, solo que mucho más rápido, ya que no manejan la seguridad ni la validación) y trozos de plantillas en **elementos parciales** (Un elemento parcial es un trozo de código de plantilla que se puede reutilizar entre acciones de un módulo o de varios módulos de una aplicación), los mismos podrán ser reutilizados por otros módulos.		

Atributos de calidad	Perfil	Escenario
Escalabilidad	Ampliación	Base de Datos
Relación atributo-escenario		
El SGBD ProstgreSQL brinda la posibilidad de crear tantos clústeres de bases de datos como se necesario. Esto permite aumentar la capacidad de almacenamiento y disminuir la sobrecarga del proceso servidor en caso que la concurrencia o el espacio sean insuficientes.		

Atributos de calidad	Perfil	Escenario
Flexibilidad	Flexibilidad	Comunicación entre Capas
Relación atributo-escenario		
Se implementó un sistema en capas que proporciona como principal ventaja la abstracción de las capas lógicas del sistema, restringiendo cada capa a interactuar solo con las adyacentes, para que los cambios solo afecten la que se sirve de esta. Logrando así mayor flexibilidad del conjunto.		

Como resultado de la evaluación cualitativa de la arquitectura propuesta, se evidenció que la gestión de la capa de acceso a datos favorece la reusabilidad y la portabilidad de la base de datos, permitiendo usar las técnicas orientadas a objetos en el modelo como principal ventaja ya que Symfony y PHP5 son orientados a objetos. Una de las características importantes que incluye el *framework* es la posibilidad de la abstracción del SGBD a través de una capa implementada por el componente Creole, consiguiendo un mantenimiento más sencillo de la aplicación, específicamente de la capa de acceso a datos. Con vistas al

crecimiento de los datos, se validó la posibilidad de crear clústeres de base de datos en caso que sea necesario debido a un incremento de la información en el servidor.

Después de analizar los resultados alcanzados con la evaluación de la arquitectura se llegó a la conclusión que la arquitectura es adecuada, proporcionando seguridad, escalabilidad, mantenibilidad y flexibilidad al sistema. Además provee otros atributos de calidad que son importantes para la aplicación como la reusabilidad y la portabilidad. Se debe tener en cuenta que el éxito de la arquitectura depende de factores como el hardware y las características del entorno de ejecución.

3.10. Conclusiones

Es este capítulo se expusieron las principales técnicas y métodos usados para evaluar una arquitectura de software. Se realizó la evaluación del diseño arquitectónico del Sistema de Gestión Fiscal de manera parcial, demostrando que la arquitectura condiciona en gran medida los atributos de calidad más deseados. Teniendo en cuenta que estos atributos no solo dependen de la arquitectura se concluye que la arquitectura cumple con los requerimientos no funcionales previstos para la primera iteración.

CONCLUSIONES GENERALES

A partir del acuerdo tomado entre la FGR y la UCI de crear un Sistema de Gestión Fiscal surge la necesidad de establecer una arquitectura confiable, flexible y óptima para el sistema que se necesitaba. Con vista a lograr un diseño arquitectónico que cumpliera con los requerimientos deseados del sistema, tuvo especial significación el estudio del estado del arte de la arquitectura de software, resultando los siguientes conceptos y tecnologías, como los más importantes:

- Arquitectura de software.
- Atributos de calidad arquitectónica.
- Estilos y patrones de arquitectura.
- Patrones de diseño y patrones GRASP.
- Tecnologías (servidores de aplicaciones, lenguajes de programación, *frameworks* de desarrollo, etc.).

La representación de la Arquitectura del Sistema de de Gestión Fiscal posibilitó llegar a un entendimiento entre todos los involucrados en el proceso de desarrollo, determinado entre otros factores por las siguientes decisiones arquitectónicas:

- Se seleccionó la metodología de desarrollo RUP.
- Se utilizó la arquitectura en capas y el patrón MVC, para todos los subsistemas de la aplicación.
- Se escogió la plataforma de desarrollo Linux-Apache-PostgreSQL-PHP (LAPP).
- De las cinco vistas arquitectónicas que abarca la metodología RUP se representaron cuatro (vista de casos de uso, vista lógica, vista de implementación, vista de despliegue) que muestran la solución técnica a diferentes niveles.

A raíz del estudio realizado se demostró que el diseño arquitectónico cumple con los objetivos propuestos, mediante la evaluación de los atributos de calidad más deseados, en los principales escenarios definidos por el arquitecto, a través del método ARID seleccionado.

RECOMENDACIONES

- Darle seguimiento al presente trabajo en futuras iteración del desarrollo del proyecto.
- Continuar refinando el documento de arquitectura de software, para tratar puntos que se hayan quedado débiles o se hayan olvidado.
- Actualizar la versión del *framework* de desarrollo utilizado ya que las últimas versiones mejoran mucho su rendimiento.
- Revisión más detallada de cada uno de los escenarios de la aplicación para evaluar a fondo las restricciones que se le imponen a la arquitectura.
- Optima configuración de los servidores Apache2 y PostgreSQL para aprovechar mejor sus características de balanceo de carga, seguridad y capacidad de procesamiento.

BIBLIOGRAFÍA

1. **Alexander, Christopher. 1979.** *A Timeless Way of Building.* s.l. : Oxford University Press, 1979.

2. **Allen, Robert. 1997.** "A formal approach to Software Architecture". s.l. : Technical Report, CMU-CS-97-144, 1997.

3. **Barbacci, Mario, et al. 1995.** Quality Attributes. [Online] Diciembre 1995. [Cited: Abril 06, 2009.] http://www.sei.cmu.edu/publications/documents/95.reports/95.tr.021.html. CMU/SEI-95-TR-021.

4. **Bass, L. and Clements, P. & Kazman R. 1998.** *Software Architecture in Practice.* s.l. : Addison-Wesley Longman, 1998.

5. **Bass, L., Klein, M. and Bachmann, F. 2000.** *Quality Attribute Design Primitives.* s.l. : Carnegie Mellon University, 2000. CMU/SEI-2000-TN-017.

6. **Bass, Len, Clements, Paul and Kazman, Rick. April 11, 2003.** *Software Architecture in Practice, Second Edition.* s.l. : Addison Wesley, April 11, 2003. ISBN : 0-321-15495-9.

7. **Bengtsson, P. 1999.** *Design and Evaluation of Software Architecture.* Department of Software Engineering and Computer Science, University of Karlskrona/Ronneby. Karlskrona : s.n., 1999. http://www.ipd.hk-r.se/pob/archive/thesis.pdf.

8. **Bosch, J. 2000.** *Design & Use of Software Architectures.* s.l. : Addison-Wesley, 2000.

9. **Buschmann, F., et al. 1996.** *Pattern Orinented Software Architecture, Volumen 1: A System of Patterns.* Inglaterra : John Wiley & Sons Ltd, 1996.

10. **Clements, Paul, Kazman, Rick and Klein, Mark. Oct 22, 2001.** *Evaluating Software Architectures: Methods and Case Studies.* s.l. : Addison-Wesley Professional, Oct 22, 2001.

11. *Comentario en discusión sobre teoría y práctica de la ingeniería de software en conferencia de OTAN Science Committee.* **Sharp, P. I. 1969.** Roma : s.n., 1969. Software Engineering Concepts and Techniques: Proceedings of the NATO conferences.

12. **Coplien, James O. and Harrison, Neil B. Julio 16, 2004.** *Organizational Patterns of Agile Software Development.* s.l. : Prentice Hall, Julio 16, 2004. ISBN: 0131467409 .

13. **Dijkstra, Edsger. Enero de 1968.** *The Structure of the THE Multiprogramming system.* s.l. : Communications of the ACM, Enero de 1968. pp. pp. 49-52. 26(1).

14. **Eeles, Peter. 2006.** International Business Machines Corp. . [Online] Febrero 15, 2006. [Cited: 03 22, 2009.] http://www.ibm.com/developerworks/rational/library/feb06/eeles/.

15. **Frederick Jr., Brooks. 1975.** *The mythical man-month. Reading.* s.l. : Addison-Wesley, 1975.

16. *Fundations for the study of software architecture.* **Elwood Perry, Dewayne and Wolf, Alexander L. 1992.** n. 4, New York, USA : ACM, 1992, Vol. 17. ISSN:0163-5948.

17. **Gamma, Erich, et al. 1995.** *Design Patterns: Elements of reusable object-oriented software.* s.l. : Addison-Wesley, 1995. ISBN 0-201-63361-2.

18. **IEEE. 1990.** *IEEE Standard Glossary of Software Engineering.* New York : s.n., 1990. Institute of Electrical and Electronics Engineers. IEEE Std 610.12-1990.

19. **Kazman, R., Clements, P. and Klein, M. 2001.** *Evaluating Software Architectures. Methods and case studies.* s.l. : Addison Wesley, 2001.

20. **Kruchten, Philippe. 1999.** *The Rational Unified Process.* s.l. : Addison Wesley Longman, Inc., 1999.

21. **Larman, Craig.** *UML y Patrones.* 2da edición. Madrid : Prentice Hall.

22. **—. 1999.** *UML y Patrones. Introducción al análisis y diseño orientado a objetos.* México : PRENTICE HALL, 1999. p. 536. ISBN: 970-17-0261-1.

23. **Monroe, Robert and Andrew Kompanek, Ralph Melton y David Garlan. Enero de 1997.** *"Architectural Styles, design patterns, and objects".* s.l. : IEEE Software, Enero de 1997. pp. pp. 43-52.

24. **Parnas, David. Diciembre de 1972.** *On the Criteria for Decomposing Systems into Modules.* s.l. : Communications of the ACM, Diciembre de 1972. pp. 1053-1058. Vol. 15. No. 12.

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

26. **Rumbaugh, James, et al. 1991.** *Object-oriented modeling and design.* Englewood Cliffs : Prentice Hall, 1991.

27. **Schmidt, D. C., Stal, M. and Buschmann, H. Rohnert & F. 2000.** *Patterns for Concurrent and Distributed Objects.* s.l. : J. Wiley and Sons Ltd, 2000. Pattern-Oriented Software Architecture.

28. **Shaw, Clements Paul & Mary. 1997.** *A field guide to boxology: Preliminary classification of architectural styles for software systems. In COMPSAC '97: Proceedings of the 21st International Computer Software and Applications Conference.* s.l. : Society, IEEE Computer, 1997. pp. p. 6–13.

29. **Shaw, M. and Garlan, D. 1996.** *Software Architecture: Perspectives on an Emerging Discipline.* s.l. : Addison-Wesley, 1996.

30. **Shaw, Mary. Octubre 1984.** *"Abstraction Techniques in Modern Programming Languages".* s.l. : IEEE Software, Octubre 1984. pp. pp. 10-26.

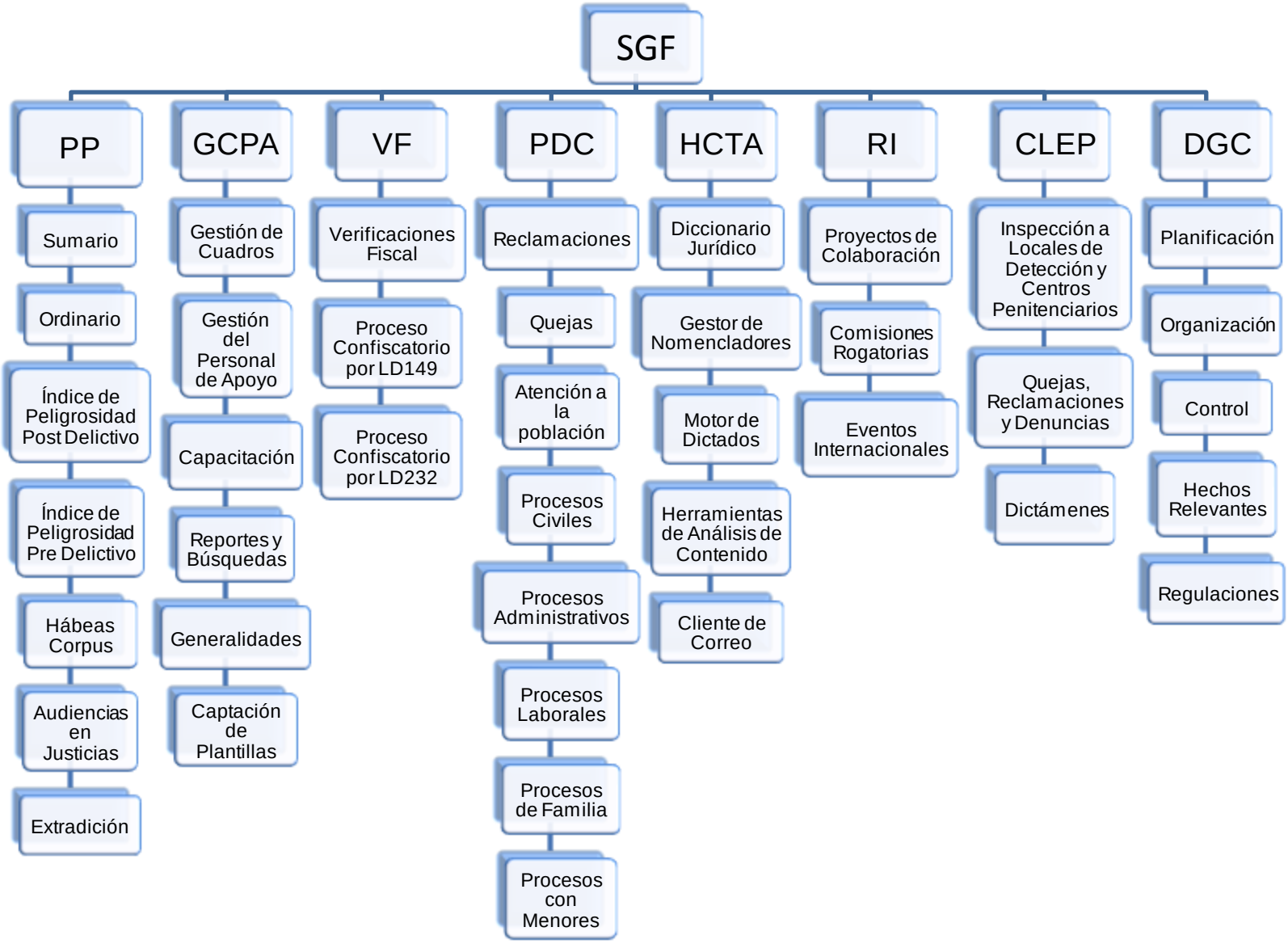
31. **—. 1989.** *“Large Scale Systems Require Higher - Level Abstraction”.* s.l. : IEEE Computer Society, 1989. pp. pp. 143-146.

32. **—. 1996.** “Some Patterns for Software Architecture”. [book auth.] J. Vlissides, J. Coplien and & N. Kerth. *Pattern Languages of Program Design.* s.l. : Addison-Wesley, 1996, Vol. Vol. II, pp. pp. 255-269.

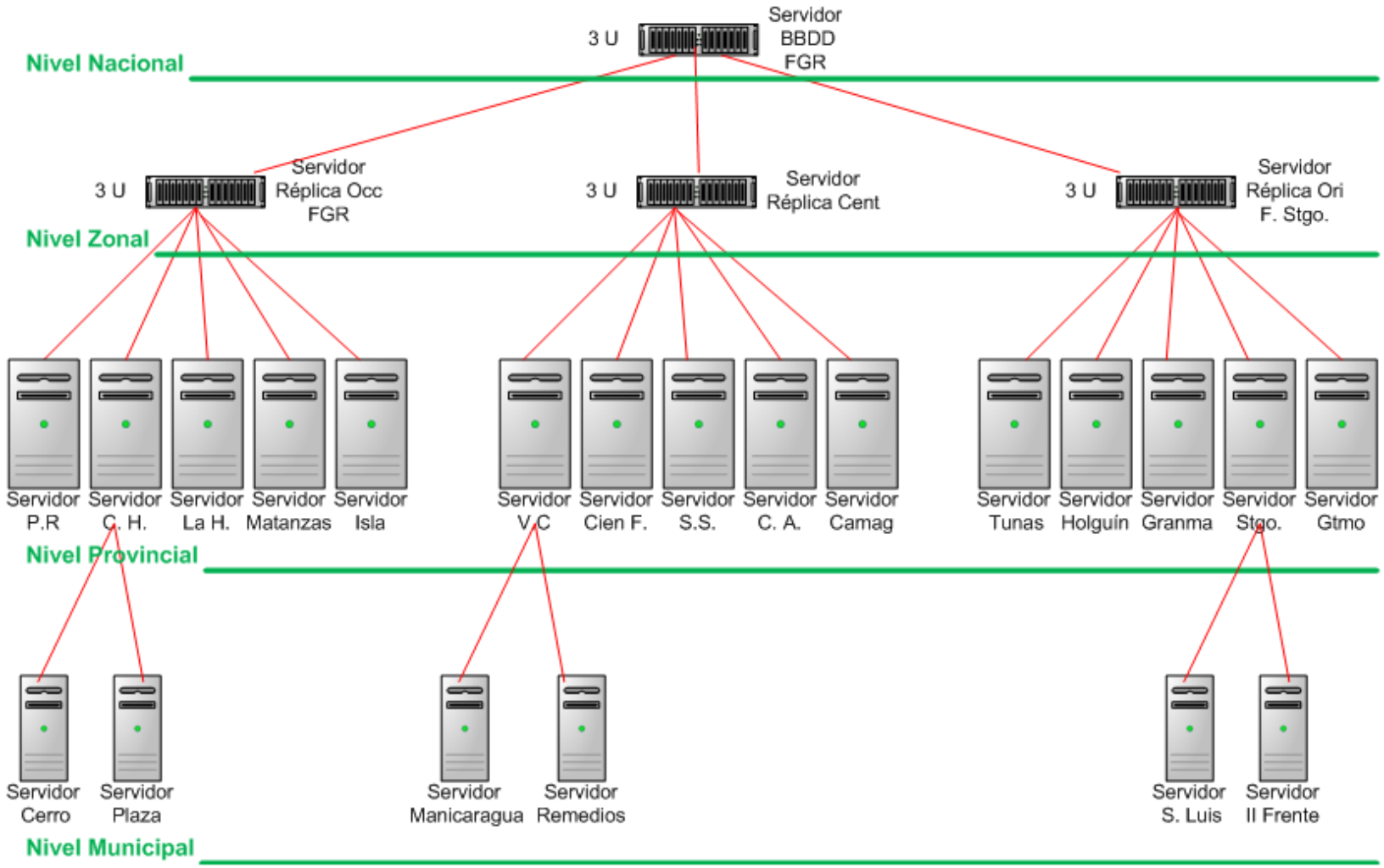
33. **Shaw, Mary and Garlan, David. Enero 1994.** *An introduction to software architecture.* s.l. : CMU SEI Technical Report, Enero 1994. CMU/SEI-94-TR-21.

34. **Wang, Guijun & Casey Fung. 2004.** “Architecture paradigms and their influences and impacts on componente-based software systems”. [Online] 2004.

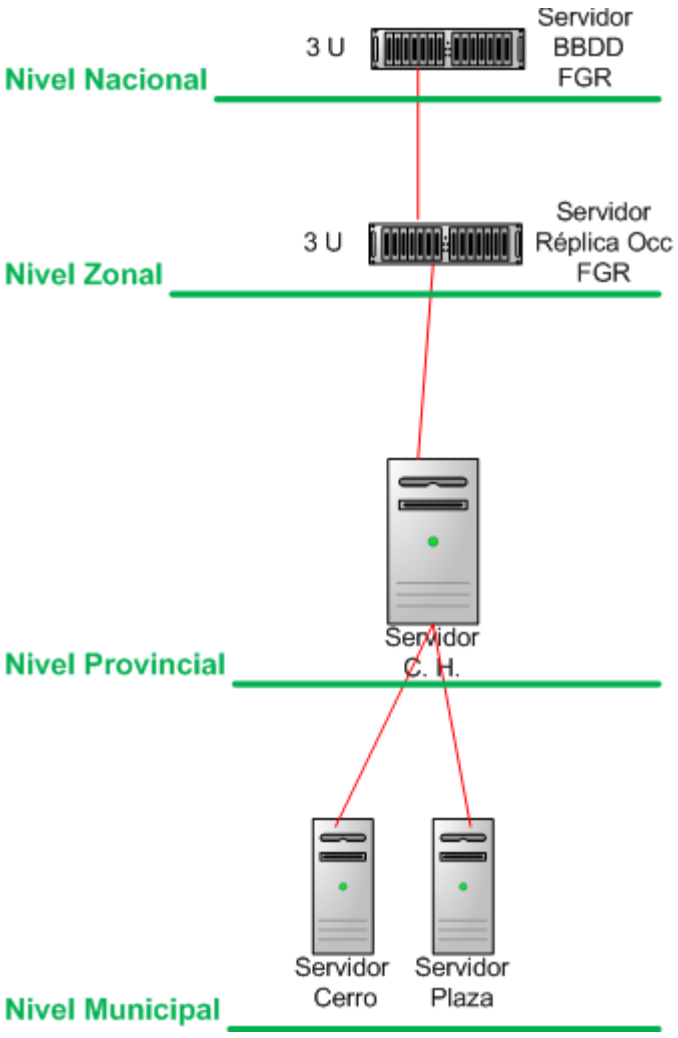
ANEXOS



Anexo 1. Muestra la estructura lógica del proyecto, organizada por niveles, un primer nivel corresponde al sistema, en un segundo nivel están representados los subsistemas y en los niveles posteriores los módulos de cada subsistema.



Anexo 2. Representación lógica final de la réplica de los servidores de base de datos a nivel nacional.



Anexo 3. Esquema jerárquico actual de la réplica de bases de datos.