Universidad de las Ciencias Informáticas

FACULTAD 6

Título: "Sistema Informático para la Red Nacional de Genética Médica: Definición de la Arquitectura de Software"

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

Autores: Elvismary Molina de Armas

Eyleen Orosco Fonseca

Tutor: Ing. Yusdenis Sánchez Perodín

Co-tutor: Ing. Alfonso Claro Arceo

Ciudad de La Habana, Cuba.

Junio 2008

"Año 50 de la Revolución"

"Programar sin una arquitectura o diseño en mente es como explorar una gruta sólo con una linterna: no sabes dónde estás, dónde has estado, ni hacia dónde vas"

Danny Thorpe

DECLARACIÓN DE AUTORÍA

DECLARACIÓN DE AUTORÍA

Declaro que soy el único autor de este trabajo y autorizo al <nombre área> 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 _____ del año _____.

Elvismary Molina de Armas	Eyleen Orosco Fonseca
_____	_____
Firma del primer Autor	Firma del segundo Autor
Ing. Yusdenis Sánchez Perodín	Ing. Alfonso Claro Arceo
_____	_____
Firma del Tutor	Firma del Co-tutor

AGRADECIMIENTOS

AGRADECIMIENTOS

De Elvismary:
En primer lugar agradezco a mi familia, fuente de mi educación y aprendizaje. A mi abuela Mima, siempre pendiente de mi, y a mi abuela Elvia a quien llevo conmigo en el recuerdo. A mis tías y tío Melba, Maritza, Miriam, Mayra y Jesús, de quienes siempre he recibido todo el apoyo que me han podido brindar. A mis primas y primos, unos más grandes que me han enseñado, y otros a los que he visto nacer, y he tenido la posibilidad de enseñarles. A mi hermana y mi sobrino, por hacerme los momentos en que estoy con ellos muy felices; y a las personas que más amo en este mundo, mis padres, por haberme conducido en la vida hasta llegar a ser quien soy hoy.
Agradezco a mis familias de Guanabacoa Marta, Rigo y Aray, y Daysi, Jorge e Isyed, quienes siempre han permanecido junto a mi. A Angélica y a Eliades, quienes compartieron conmigo los momentos más duros. A Yuniel, por hacerme sentir siempre presente, por su paciencia y su amor.
Doy un agradecimiento especial a las dos personitas que me han hecho conocer la verdadera amistad: Odannis y Kenia; a Yariel por mostrarme lo que es la fe, y a todos mis compañeros de estudio.
Agradezco a Eyleen por ser mi dúo de tesis, y haber compartido estos días de stress y trabajo juntas. Además a los compañeros Alfredo y Juan Carlos de Softel, quienes fueron muy atentos a la hora de saldar mis dudas con respecto a la investigación.
Y finalmente agradezco a todos los profesores que han participado en mi formación profesional durante estos cinco años, y en especial, a la Universidad de las Ciencias Informáticas y a la Revolución, que me dio la oportunidad de cursar esta carrera, haciendo mis sueños realidad.

AGRADECIMIENTOS

De Eyleen:

Agradezco en primer lugar a mis padres por haberme guiado en la vida y dar todo de sí mismos para hacerme una persona preparada y de bien. Mami, Papi los quiero muchísimo.

A mi hermano que más que eso ha sido mi amigo y apoyo en los duros momentos y a Dany que ha sido como otra hermana para mí. A mis hermanos Pedro y Jorge.

A mis abuelos Margarita, Chuchi y también a los que lamentablemente ya no están conmigo.

A mis tíos y primos que hacen que nuestra familia sea tan unida: Papo, Loli, Raico, Cherley, Libia, Angelito, Lore, Daya, Noly, Marilín, Anet, Darwin, Anni y Jany. A todos los que quiero mucho.

No podía tampoco dejar de agradecer a Ivón que me vió crecer y que siempre tuvo una mano y una sonrisa para mí en mis primaros años de vida y también en mi adolescencia, así como a Elvita, Yusi y Pepe.

A mis amigos de la Universidad, pocos pero muy buenos: Odita, Blacky, Ema, Keku y Mary. Les agradezco la comprensión durante todos estos años de mi carrera, los momentos tensos de estudios donde siempre tomamos un descanso entre risas y apreciamos lo bueno de la vida y de la amistad, los consejos, el cariño y las buenas enseñanzas que me dieron y que no olvidaré.

A Andrés por alentarme a superarme como profesional y compartir conmigo su amor, cariño, inteligencia y las cosas lindas que hemos vivido juntos.

A mis amigas Marigerlys, Yohenia, Hildy y Leira, que en diferentes momentos de mi vida jugaron un papel importante y que siempre las llevo presente.

A los compañeros del MININT de la Isla que también me han atendido y seguido durante mi carrera.

A todos aquellos que me han formado como profesional durante estos cinco años y a esta Universidad que me ha dado la oportunidad de ser una profesional.

DEDICATORIA

DEDICATORIA

De Eyleen:

Dedico el presente Trabajo de Diploma:

A la Revolución y a Fidel por darme la oportunidad de hacerme una profesional.

A mis padres por ser la luz que siempre guió mis pasos.

Al MININT por acogerme entre sus filas y darme la carrera que culmino hoy.

De Elvismary:

Dedico el presente Trabajo de Diploma:

A mi madre por ser mi inspiración diaria, mi musa de la confianza, mi aliento para seguir siempre adelante.

A mi padre por exigirme cada día ser mejor, guiándome con su ejemplo de voluntad, empeño y sacrificio.

A mi familia, creadora de todo lo que soy hoy.

A la Revolución y a Fidel por permitirme cumplir este sueño.

RESUMEN

El presente trabajo de diploma surge a partir de la necesidad de integrar los módulos que gestionan los datos recogidos en los estudios de genética realizados en el país, en un único sistema, el Sistema Informático para la Red Nacional de Genética Médica. Para lograr este objetivo se hizo necesario definir una arquitectura de software que cumpliera con las normas establecidas por el Grupo de Arquitectura MINSAP-MIC, encargado de regir la estandarización de las aplicaciones informáticas destinadas al sector de la salud en Cuba.

Para la selección de las herramientas informáticas y tecnologías para el desarrollo del sistema se tuvo en cuenta que estuvieran bajo licencia libre, debido a las dificultades que presenta actualmente el país para acceder a las que son propietarias. Además se empleó el patrón arquitectónico Modelo-Vista-Controlador que está implementado por el framework aplicado en el desarrollo del sistema. La descripción y construcción de la solución propuesta se presenta a través de las vistas definidas por la metodología de desarrollo RUP, entre las que se encuentran la vista de casos de uso, la vista lógica, la vista de despliegue y la vista de implementación. Para validar la propuesta realizada se aplicó el método de evaluación ARID, que obtuvo resultados satisfactorios, ya que se cumplieron todos los escenarios definidos para cada atributo de calidad, demostrando la adecuación de la arquitectura propuesta.

La presente investigación constituye una guía para los desarrolladores que deseen emplear el framework Symfony.

PALABRAS CLAVES: Sistema Informático para la Red Nacional de Genética Medica, vista de casos de uso, vista lógica, vista de despliegue, vista de implementación

TABLA DE CONTENIDOS

TABLA DE CONTENIDOS

TABLA DE CONTENIDOS

INTRODUCCIÓN

Con más de 20 años de experiencia en el desarrollo de un programa de diagnóstico, manejo y prevención de enfermedades genéticas y defectos congénitos, Cuba ha producido una verdadera revolución en la extensión de los servicios de genética a la comunidad.

El Programa de la Revolución para la atención a personas con discapacidad y el desarrollo de la genética médica en el país, tiene como sede el Centro Nacional de Genética Médica (CNGM), rector además de la red nacional constituida por los centros de Genética Médica en los 169 municipios y las 14 provincias del país, en los cuales se desempeñan 453 másteres en Asesoramiento Genético.

Estos nuevos avances, dirigidos a ofrecer una mayor seguridad de salud y felicidad a la familia cubana, tuvieron su punto de partida hace cinco años luego del estudio nacional realizado a personas con retraso mental y otras discapacidades, que por orientaciones de Fidel se realizó entre julio del 2001 y abril del 2003 insertado dentro de la Batalla de Ideas. Entre otros estudios que fueron desarrollándose paulatinamente también se encuentra el Registro Cubano de Gemelos del país, que permite hacer estudios sobre cómo contribuyen los factores genéticos al origen de enfermedades como el asma, la hipertensión, la diabetes, el cáncer; la investigación sobre la prevalencia de la demencia, y en particular del Alzheimer, así como los estudios de longevidad en personas de más de 99 años.

Para dar soporte a estos estudios se hace necesaria una aplicación informática de gestión que brinde soporte de almacenamiento a estos datos, garantizando el acceso a los mismos desde cualquier punto del país a través de la red de computadores, brindando un flujo de información rápido y seguro, y donde se listen de forma automatizada diferentes reportes estadísticos de importancia para el investigador.

Dicho sistema, nombrado Sistema Informático para la Red Nacional de Genética Médica (SIGM), se ha concebido bajo las pautas de integración de varios registros, que recogen los datos de los diferentes estudios genéticos realizados en el país. El mismo debe además insertarse en la Red Telemática de Salud en Cuba, Infomed, integrándose específicamente con el Sistema de Información para la Salud, SISalud.

INTRODUCCIÓN

Teniendo en cuenta los argumentos anteriormente expuestos, el problema científico a resolver en la presente investigación consiste en *¿cómo definir la organización estructural del Sistema Informático para la Red Nacional de Genética Médica, teniendo en cuenta los componentes del sistema, las relaciones entre ellos, y el ambiente donde se implantarán?*

Ante la interrogante planteada como problema científico, se identifica como **objeto de estudio** a *la arquitectura de software,* y dentro de este se selecciona como **campo de acción** la *arquitectura de software de aplicaciones para la Red Nacional de Salud.*

Para dar respuesta al problema científico, la investigación propone como **objetivo general** *definir la arquitectura del Sistema Informático para la Red Nacional de Genética Médica,* del cual se derivan los siguientes **objetivos específicos:**

- Caracterizar al sistema definiendo los requerimientos arquitectónicos, el patrón de arquitectura, así como las tecnologías y herramientas informáticas para dar soporte al desarrollo.
- Describir la arquitectura del sistema.
- Evaluar la arquitectura propuesta.

Para darle cumplimiento a estos objetivos específicos se definen las siguientes tareas:

1. Análisis de los elementos fundamentales, tecnologías y herramientas informáticas para el desarrollo de la arquitectura.
2. Análisis de la arquitectura de software establecida para el desarrollo de sistemas informáticos para el MINSAP.
3. Descripción de los requerimientos arquitectónicos.
4. Definición y descripción de las tecnologías y herramientas informáticas a utilizar.
5. Definición y descripción del patrón arquitectónico a utilizar.
6. Diseño de las vistas de casos de uso, lógica, despliegue e implementación.
7. Análisis de las técnicas y métodos de evaluación de la arquitectura de software.
8. Evaluación de la arquitectura definida.

INTRODUCCIÓN

El presente trabajo de diploma se estructura en cuatro capítulos:

- En el Capítulo 1: FUNDAMENTACIÓN TEÓRICA, son analizados el concepto de arquitectura de software y los elementos estrechamente vinculados con la misma. Además se describe la arquitectura del Sistema Informático para la Salud.
- En el Capítulo 2: CARACTERÍSTICAS DEL SISTEMA, se analizan los requisitos del sistema y se definen las tecnologías y herramientas informáticas propuestas para dar soporte al desarrollo. Además se selecciona y describe el patrón arquitectónico a utilizar.
- En el Capítulo 3: DESCRIPCIÓN DE LA ARQUITECTURA, se presentan las diferentes vistas arquitectónicas que componen la descripción de la arquitectura planteada por la metodología de desarrollo RUP.
- En el Capítulo 4: EVALUACIÓN DE LA ARQUITECTURA PROPUESTA, se identifican los atributos de calidad y el método para realizar la evaluación de la arquitectura propuesta, describiéndose posteriormente los resultados.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

Dentro de las diferentes metodologías de desarrollo del software, se encuentra un elemento fundamental para el éxito del producto: la arquitectura de software. Esta es considerada como una radiografía del sistema que se está desarrollando, lo suficientemente completa como para que todos los implicados en el desarrollo tengan una idea clara de qué es lo que se está construyendo.

En el presente capítulo se realiza un análisis de los elementos fundamentales de la arquitectura de software entre los que se encuentran los estilos, patrones y lenguajes de descripción arquitectónicos. Se describe además de forma general la metodología de desarrollo RUP, mostrando sus fases y flujos, y las actividades que desempeña el rol de arquitecto en la misma. Finalmente se presentan las normas que establece el Grupo de Arquitectura MINSAP – MIC para el desarrollo estandarizado de las aplicaciones informáticas en el sector de la salud, profundizándose en la descripción de la arquitectura de SISalud, uno de los sistemas ya implantados dentro de este sector.

1.1 ARQUITECTURA DE SOFTWARE

La arquitectura de software es un tema recurrente para muchos estudiosos de las metodologías de desarrollo de software. Algunos con criterios abstractos, otros con pensamientos más concretos y cercanos a la implementación, han abordado los aspectos fundamentales que influyen en una arquitectura de software. A continuación se toman en cuenta ambos criterios y se estudia el marco específico donde será implantado el sistema.

1.1.1 DEFINICIÓN DE ARQUITECTURA DE SOFTWARE

Casi todos han observado alguna vez la construcción de un edificio. Comienza por los cimientos, luego las columnas y vigas, las distintas plantas, hasta tener un esqueleto de soporte. Primero se crea la estructura o esqueleto del edificio que aportan la mayoría de las funcionalidades básicas del inmueble, y luego se ensamblan las distintas partes. Esta forma de proceder es una estrategia general de solución de problemas en multitud de disciplinas sociales y técnicas, siendo una de ellas la creación de software.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

Desde el punto de vista de qué debe hacer el software, la arquitectura "se define a partir de un conjunto de requisitos críticos funcionales, de rendimiento, o de calidad" (1). Desde otro punto de vista más tangible, "la arquitectura se materializa en el conjunto de componentes de código fuente y ejecutables que implementan dicho esqueleto, lo que posibilita demostrar y evaluar en qué medida el diseño da solución a aquellos requisitos críticos" (1)

Existen variados criterios sobre qué es la arquitectura de software. A continuación se exponen algunos de ellos:

Clements considera que "La arquitectura de software es, a grandes rasgos, una vista del sistema que incluye los componentes principales del mismo, la conducta de esos componentes según se la percibe desde el resto del sistema y las formas en que los componentes interactúan y se coordinan para alcanzar la misión del sistema. La vista arquitectónica es una vista abstracta, aportando el más alto nivel de comprensión y la supresión o diferimiento del detalle inherente a la mayor parte de las abstracciones" (2).

Kazman define la Arquitectura de Software como "la estructura de estructuras de un sistema, la cual abarca componentes de software, propiedades externas visibles de estos componentes y sus relaciones" (3).
Jacobson plantea que "la arquitectura abarca decisiones importantes sobre:

- La organización de un sistema de software
- Los elementos estructurales que compondrían el sistema y de sus interfaces, junto con sus comportamientos
- La composición de los elementos estructurales y del comportamiento en subsistemas progresivamente más grandes
- El estilo de la arquitectura que guía esta organización: los elementos y sus interfaces, sus interfaces, sus colaboraciones y su composición" (4)

La IEEE Std 1471-2000 define que "la Arquitectura del Software es la organización fundamental de un sistema formada por sus componentes, las relaciones entre ellos y el contexto en el que se implantarán, y los principios que orientan su diseño y evolución". (5) Este concepto es asumido internacionalmente como el oficial.

Después de un análisis de los conceptos anteriores, para la presente investigación se asume la definición de arquitectura de software propuesta por la IEEE Std 1471-2000.

1.1.2 ESTILOS ARQUITECTÓNICOS

Al desarrollar una arquitectura de software se deben crear y representar componentes que interactúen entre ellos y tengan asignadas tareas específicas, además de organizarlos de forma tal que se logren los requerimientos establecidos. Se puede comenzar aplicando patrones de soluciones ya probados, con la intención de no comenzar de cero las propuestas, y utilizar modelos que han funcionado durante la práctica agrupando ciertas regularidades de configuración como respuesta a problemas recurrentes. Estas soluciones probadas se conocen como estilos y patrones arquitectónicos, y van de lo general a lo particular.

Los estilos arquitectónicos surgieron como analogía al término que se utiliza en la arquitectura de edificios, definiendo una colección de tipos de componentes y un conjunto coordinado de restricciones que enmarcan los roles de dichos componentes y de sus relaciones.

Las primeras definiciones explícitas de estilo fueron propuestas por Dewayne Perry de AT&T Bell Laboratories de New Jersey y Alexander Wolf de la Universidad de Colorado. Ellos definieron un estilo arquitectónico como “una abstracción de tipos de elementos y aspectos formales a partir de diversas arquitecturas específicas. Un estilo arquitectónico encapsula decisiones esenciales sobre los elementos arquitectónicos y enfatiza restricciones importantes de los elementos y sus relaciones posibles.” (6)

Posteriormente Mary Shaw y Paul Clements conceptualizan los estilos arquitectónicos como un “conjunto de reglas de diseño que identifica clases de componentes y conectores que se pueden manejar para componer el sistema, junto con las restricciones locales o globales que determinan como se lleva a cabo la composición” (7).

Luego, Roy Thomas Fielding sintetiza la definición de estilo, exponiendo que un estilo arquitectónico es un “conjunto coordinado de restricciones arquitectónicas que restringe los roles de los elementos arquitectónicos y las relaciones permitidas entre esos elementos” (8).

La definición de los estilos y su asignación a determinada clase de estilo, que los agrupan por similitud, es susceptible a realizarse de múltiples formas, conforme a los criterios que se apliquen en la constitución de los ejemplares y a la valoración muy particular de cada autor. En la presente investigación se adopta la clasificación propuesta por Shaw y Garlan (9) quienes define las siguientes clases de estilos:

Estilos de Flujo de datos

- ✓ Tuberías y Filtros

Estilos centrados en datos

- ✓ Arquitecturas de Pizarra o repositorio

Estilos de Llamada y Retorno

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

Estilo de Código Móvil

- ✓ Arquitectura de Máquinas Virtuales

Estilos Peer – To – Peer

- ✓ Arquitectura basadas en Eventos
- ✓ Arquitecturas Orientas a Servicios (SOA)

A continuación se describen algunos de los estilos arquitectónicos más usados en la actualidad:

Arquitectura en Capas:

Estilo arquitectónico que establece una organización jerárquica de capas, las cuales agrupan componentes. “Cada capa proporciona servicios a la capa inmediatamente superior y se sirve de las prestaciones que le brinda la inmediatamente inferior” (10). El objetivo principal de este estilo es la separación de la lógica de negocio de la lógica de diseño.

Una aplicación básica de este estilo es segmentar la lógica de la solución en tres capas: la capa de presentación, la de negocio y la de datos. Seguidamente son especificadas dichas capas.

Capa de presentación: Constituye la interfaz del usuario con el sistema. Esta se encarga de visualizar la información del sistema al usuario, y a su vez de capturar los datos introducidos por este que son importantes para realizar las operaciones de la aplicación. Esta capa se comunica únicamente con la capa de negocio.

Capa de negocio: Es el contenedor de procesos del sistema, donde se ejecutan los programas que dan soporte a las funcionalidades y reglas que deben cumplirse. Esta capa se comunica con la capa de presentación, para recibir las solicitudes y presentar los resultados de las peticiones del usuario; y con la capa de datos, para consultar la información o guardar la misma.

Capa de datos: En ella reside la información persistente del sistema. Constituida por uno o más gestores de bases de datos que realizan todo el almacenamiento de datos, y responden a diferentes solicitudes de almacenamiento o consultas desde la capa de negocio.

Todas estas capas pueden residir en un único ordenador o estar desplegadas en varios de ellos, en dependencia de la complejidad y extensión de cada una. La representación física de las capas, o sea, la cantidad de ordenadores en que están distribuidas es a lo que se le denomina niveles.

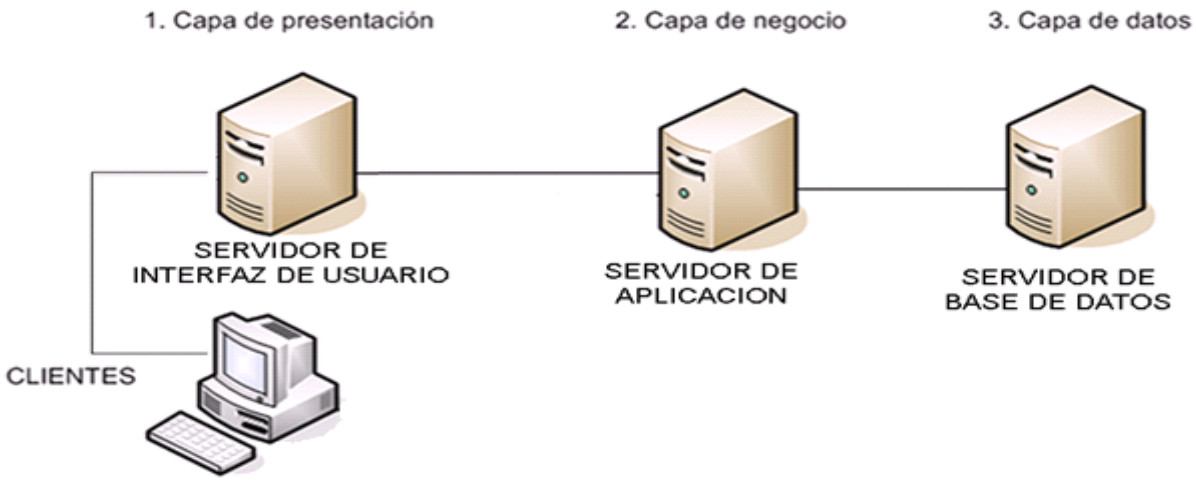


Figura 1. Representación del estilo Arquitectura en Capas. Ejemplo de tres capas y tres niveles.

La ventaja principal de este estilo, es que "soporta un diseño basado en niveles de abstracción crecientes, lo cual a su vez permite a los implementadores la partición de un problema complejo en una secuencia de pasos incrementales" (10). En caso de algún cambio solo se ataca la capa defectuosa sin necesidad de revisar en el código mezclado.

Arquitectura Orientada a Servicios (SOA)

La Arquitectura Orientada a Servicios (SOA, por sus siglas en inglés) "define la utilización de servicios para dar soporte a los requerimientos de software del usuario" (11). En un ambiente SOA, los nodos de la red hacen disponibles sus recursos a otros participantes como servicios independientes, a los que tienen acceso de un modo estandarizado. Para comunicarse entre sí, estos servicios se basan en una definición formal independiente de la plataforma subyacente y del lenguaje de programación. Para ello se utilizan tecnologías basadas en servicios, la más extendida hasta hoy son los web services, que contienen una colección de protocolos y estándares (XML, SOAP, WSDL y UDDI) que sirven para intercambiar datos entre aplicaciones".

Para mayor comprensión se ilustra en la siguiente figura una representación de este tipo de arquitectura.

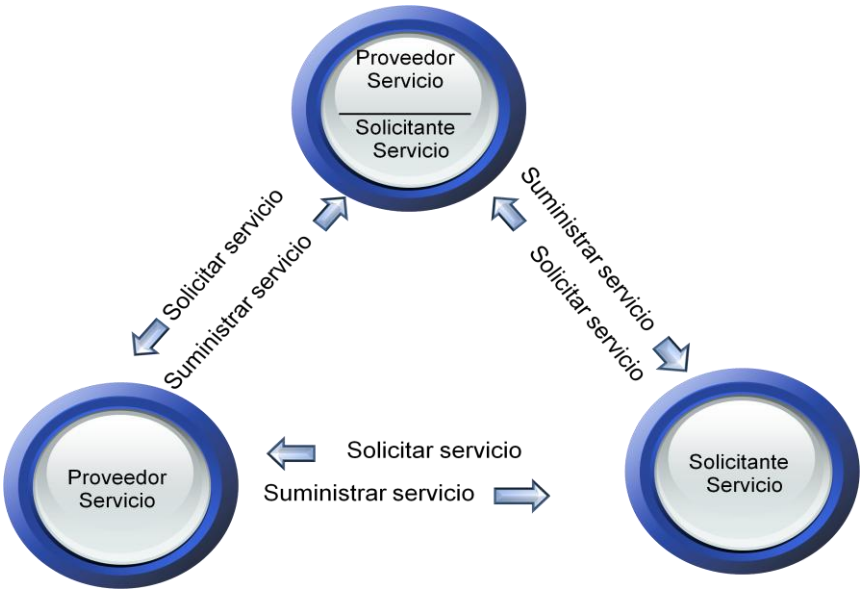


Figura 2. Representación del estilo Arquitectura Orientada a Servicios.

Esta arquitectura permite un alto grado de reutilización de los componentes desarrollados, debido a que “los servicios de aplicación son débilmente acoplados y altamente interoperables” (12), y las dependencias se establecen en tiempo de ejecución.

Modelo-Vista-Controlador (MVC):

Estilo arquitectónico que “separa el modelado del dominio, la presentación y las acciones basadas en datos ingresados por el usuario en tres componentes distintos: el modelo, la vista y el controlador” (10), los cuales se especifican a continuación:

Modelo*:* Gestiona el comportamiento y los datos del dominio de aplicación. Responde a las peticiones que formula la vista sobre su estado, y a las instrucciones de cambiar su estado enviadas por el controlador.

Vista*:* Muestra el estado del controlador o del modelo al usuario del sistema.

Controlador*:* Interpreta las acciones del usuario del sistema, e informa al modelo y/o a la vista para que estos cambien su estado según corresponda.

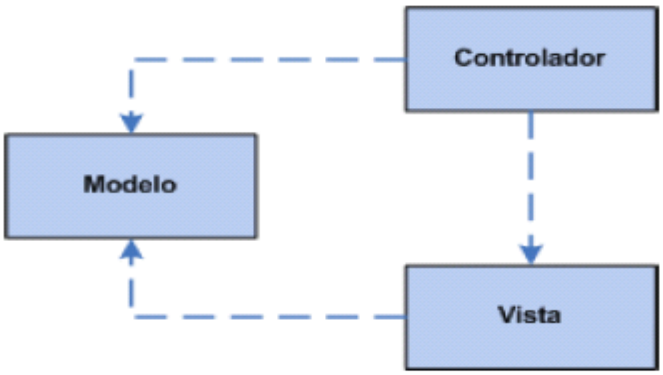


Figura 3. Representación del estilo MVC.

Los beneficios de usar MVC son: “su modularidad, un diseño claro y ampliamente aceptado, facilidad para crear distintas vistas del mismo modelo de negocio y su extensibilidad” (13).

1.1.3 PATRONES ARQUITECTÓNICOS

Un patrón se define como una solución probada con éxito que aparece una y otra vez ante determinado tipo de problema. Este engloba conocimientos específicos, y aplicarlo constituye un eslabón importante en el aprovechamiento de la experiencia acumulada en el campo en cuestión.

Los patrones se definen por un nombre, un problema, una solución y las consecuencias de su aplicación. En dependencia del problema que solucionan, estos pueden ser clasificados en: patrones de arquitectura, patrones de análisis, patrones de diseño, entre otros.

Los patrones arquitectónicos constituyen una descripción de un problema particular, que aparece en contextos de diseño específico, y presenta un esquema genérico demostrado con éxito para su solución.

Los patrones arquitectónicos “expresan el esquema de organización estructural fundamental para sistemas de software, proveen un conjunto de subsistemas predefinidos, especifican sus responsabilidades e incluyen reglas y pautas para la organización de las relaciones entre ellos” (14).

A continuación se listan los patrones arquitectónicos más conocidos y utilizados:

Layers (Capas):

Este modelo ayuda a estructurar aplicaciones que pueden estar “descompuestas en grupos de subtareas, que están en un determinado nivel de abstracción” (15). Define cómo organizar el diseño a través de capas, que pueden estar o no físicamente distribuidas en distintos computadores. Los componentes de una capa sólo pueden hacer referencia a componentes en capas inmediatamente inferiores. La comunicación sólo se establece entre componentes de la misma capa, o entre componentes de capas contiguas.

Este patrón simplifica la comprensión y la organización del desarrollo de sistemas complejos, reduciendo las dependencias de forma que las capas más bajas no son conscientes de ningún detalle de las superiores.

Pipes and Filtres (Tuberías y Filtros):

Esta arquitectura está constituida por "una tubería (pipeline) que conecta componentes computacionales (filtros) a través de conectores (pipes), de modo que las operaciones se ejecutan a la manera de un flujo" (16). Los datos se transportan a través de las tuberías entre los filtros, transformando gradualmente las entradas en salidas.

Es habitualmente utilizado en compiladores, sistemas de UNIX y tratamiento de documentos en XML.

Blackboard:

La arquitectura en pizarra consta de "una estructura de datos que representa el estado actual (pizarra) y una colección de componentes independientes que operan sobre él, denominados agentes." (17)

Los agentes son programas especializados en una tarea concreta. Todos ellos cooperan para alcanzar una meta común, sin embargo, sus objetivos individuales no están aparentemente coordinados.

La pizarra tiene un doble papel. Por una parte, coordina a los distintos agentes, y por otra, facilita su intercomunicación. El estado inicial de la pizarra es una descripción del problema a resolver. El estado final será la solución del problema. Este patrón es habitualmente utilizado en sistemas expertos de Inteligencia Artificial.

Broker:

Se utiliza para "estructurar la gestión de sistemas con componentes distribuidos, que interactúan por invocaciones de servicios remotos" (15). Permite que los objetos hagan llamadas a otros objetos remotos a través de un gestor o broker que redirige la llamada al nodo y al proceso que guarda al objeto deseado. El componente broker es el responsable de coordinar la comunicación, reenviando demandas múltiples, así como transmitiendo resultados y excepciones. Esta redirección se hace de manera transparente lo cual quiere decir que el llamante no necesita saber si el objeto llamado es remoto.

Model View Controller (MVC):

Divide la lógica de la aplicación en tres componentes: modelo, vista y controlador. El modelo contiene los datos y funcionalidades relacionadas con estos. La vista exhibe información para el usuario. Los controladores se comunican con el modelo, actualizando o consultando los datos, para dar respuesta a las solicitudes hechas por el usuario a través de la vista.

El patrón MVC se ve frecuentemente en aplicaciones web, donde la vista es la página HTML y el código que provee de datos dinámicos a la página, el modelo son los datos del dominio gestionados por un SGBD, y el controlador representa la lógica de negocio. Existen diferentes implementaciones del MVC (18). Entre ellas se encuentran las siguientes:

Variante I: La vista toma directamente del modelo los datos que va a mostrar, los busca dada una indicación del controlador, disminuyendo el conjunto de responsabilidades de este último.

Variante II: No existe ninguna comunicación entre el modelo y la vista. Esta recibe los datos a mostrar a través del controlador permitiendo mayor independencia entre los tres y el bajo acoplamiento.

Microkernel:

Se aplica a los sistemas del software que deben poder adaptarse a requisitos de sistema cambiantes. Separa "un núcleo mínimo funcional de la funcionalidad extendida y las especificaciones del usuario" (15). El microkernel sirve como conector entre estas extensiones y su objetivo es coordinar su colaboración.

¿Patrones o estilos arquitectónicos?

A lo largo del estudio realizado surge la disyuntiva acerca de cuáles son las diferencias entre estilos y patrones arquitectónicos. Es por esta razón que se analizan los siguientes criterios:

En Pattern-oriented software architecture (POSA), se definen los patrones como "un esquema de organización estructural para los sistemas de software que proporcionan un conjunto de subsistemas predefinidos, especifica sus responsabilidades e incluye reglas y lineamientos para organizar la relación entre ellos" (19)

Mary Shaw plantean que “los estilos se pueden conceptualizar como clases de patrones, o tal vez más adecuadamente como lenguajes de patrones a partir de los cuales los arquitectos pueden construir patrones de diseño para resolver problemas específicos” (20).

Robert Allen estima que “los patrones son similares a los estilos en la medida en que definen una familia de sistemas de software que comparte características comunes” (21), pero también señala diferencias sosteniendo que “un estilo representa específicamente una familia arquitectónica, construida a partir de bloques de construcción arquitectónicos, como componentes y conectores. Los patrones, en cambio, atraviesan diferentes niveles de abstracción y etapas del ciclo de vida del software partiendo del análisis del dominio y pasando por la arquitectura, llegando hasta el nivel de los lenguajes de programación” (21).

Microsoft adopta el criterio de que “los patrones se refieren más bien a prácticas de re-utilización y se encuentran más ligados al uso y al plano físico, mientras los estilos conciernen a teorías sobre la estructura de los sistemas a veces más formales que concretas, enfatizando descriptivamente las configuraciones de una arquitectura, desarrollando incluso lenguajes y notaciones capaces de expresarlas formalmente” (10).

Son muy disímiles los criterios respecto a la relación que existe entre los estilos y patrones arquitectónicos, aunque con respecto a la bibliografía consultada se arribó a las conclusiones que se comentan a continuación:

Al establecer diferencias entre estilo y patrón arquitectónico se puede comenzar mencionando que al contrario de los estilos, los patrones son muchos y muy variados, y es casi imposible revisarlos todos o su gran mayoría a la hora de desarrollar una arquitectura.

Los patrones son mucho más específicos que los estilos. Los estilos solo describen el esqueleto estructural general para cada aplicación, mientras que los patrones expresan un problema recurrente de diseño muy específico, y presentan una solución desde el punto de vista del contexto.

A pesar de estas diferencias es importante destacar que en diferentes bibliografías los patrones son aproximadamente lo mismo que lo que se acostumbra a definir como estilos, y ambos términos se usan indistintamente. Algunos patrones coinciden con los estilos incluso hasta en el nombre con

que se les designa. Sin embargo es posible concluir que en realidad cuando se habla de estilos o patrones se refiere a conceptos similares hasta cierto punto; cuando se habla de estilo, se expresa una solución en uso, mientras que un patrón constituye una solución y un método para evaluar su adecuación a un problema.

1.1.4 LENGUAJES DE DESCRIPCIÓN DE LA ARQUITECTURA

Un lenguaje de descripción de arquitectura (ADL, por sus siglas en inglés) se utiliza para describir una arquitectura de software especificando la estructura y el comportamiento de la misma. Se emplean generalmente al especificar y describir un estilo arquitectónico.

La definición más simple es la que define un ADL como una “entidad consistente en cuatro “Cs”: componentes, conectores, configuraciones y restricciones (constraints)” (22). Otros autores, como Shaw y Garlan han afirmado que los conectores son las entidades principales, y que un ADL genuino tiene que proporcionar propiedades de composición, abstracción, reusabilidad, configuración, heterogeneidad y análisis. Como especificación más completa se puede considerar la Mediovic, quien define a los ADLs a través de las características de sus componentes, conectores, configuraciones arquitectónicas y herramientas informáticas de soporte.

Son diversos los conceptos de ADLs, pero todos presentan elementos constitutivos comunes, los cuales se detallan a continuación:

Componentes: Representan los elementos u objetos primarios de un sistema. En la mayoría de los ADLs los componentes pueden exponer varias interfaces, las cuales definen puntos de interacción entre un componente y su entorno.

Conectores. Representan interacciones entre componentes. Los conectores también tienen una especie de interfaz que define los roles entre los componentes participantes en la interacción

Configuraciones o sistemas. Se representan como grafos de componentes y conectores.

Restricciones. Representan condiciones de diseño que deben acatarse incluso en el caso que el sistema evolucione en el tiempo.

Dentro de los principales ADLs se encuentran Acme, Aesop, CHAM, C2 y Jacal.

1.1.5 EL PROCESO UNIFICADO DE DESARROLLO Y LA ARQUITECTURA DE SOFTWARE

El Proceso Unificado de Desarrollo de Software (RUP, por sus siglas en inglés), es una metodología de desarrollo de software resultante de varios años de investigación y uso práctico, en el que se han unificado diferentes técnicas de producción de software. RUP se define como un conjunto de actividades necesarias para transformar los requisitos del usuario en un sistema de software.

RUP se distingue sobre otras metodologías en que propone un crecimiento incremental de la aplicación usando desarrollo por iteraciones, un alto grado de detalle de la documentación, buenas prácticas de prueba, manejo de riesgos, y contacto cercano y retroalimentación con los usuarios. Además requiere definir una arquitectura, desarrollar un modelaje visual y el uso de herramientas informáticas como puntos fundamentales.

RUP presenta tres características fundamentales: dirigido por casos de uso, centrado en la arquitectura, e iterativo e incremental (4).A continuación se explican cada uno de estos elementos:

- Dirigido por casos de uso: Un caso de uso según RUP es el conjunto de acciones a realizar por el sistema para producir un resultado de valor observable para un usuario; son utilizados para enmarcar los requisitos funcionales del sistema y guían todos los flujos de trabajo hasta elaborar un producto de software.

- Centrado en la arquitectura: Las vistas de la arquitectura muestran desde diferentes perspectivas el sistema que se va a construir. Esta permite ir agregando paulatinamente los casos de uso, comenzando por los más significativos, hasta obtener una versión ejecutable que satisfaga todos los casos de uso del sistema. Las vistas que la componen hacen que el trabajo en equipo sea más organizado teniendo una estructura clara y concisa del software que se quiere construir.

- Iterativo e incremental: Debido a la complejidad de los sistemas, se hace factible dividir el proceso de desarrollo en varias iteraciones. Cada una de estas iteraciones se planifica y se controla, por lo que son nombradas mini-proyectos y representan un incremento en el producto final. En cada iteración son seleccionados nuevos casos de uso que se diseñarán, implementarán y probarán; y pasarán a formar parte del prototipo de interfaz del usuario.

RUP divide en cuatro fases el ciclo de desarrollo del software. Cada fase tiene definido un conjunto de objetivos y un punto de control específico. Cada una de estas etapas es desarrollada mediante el ciclo de iteraciones, la cual consiste en reproducir el ciclo de vida en cascada a menor escala. Los objetivos de una iteración se establecen en función de la evaluación de las iteraciones precedentes. Vale mencionar que el ciclo de vida que se desarrolla a través de cada iteración se denomina flujo de trabajo y se dividen en dos disciplinas: disciplinas del proceso y disciplinas de soporte. La figura representa lo anteriormente explicado:

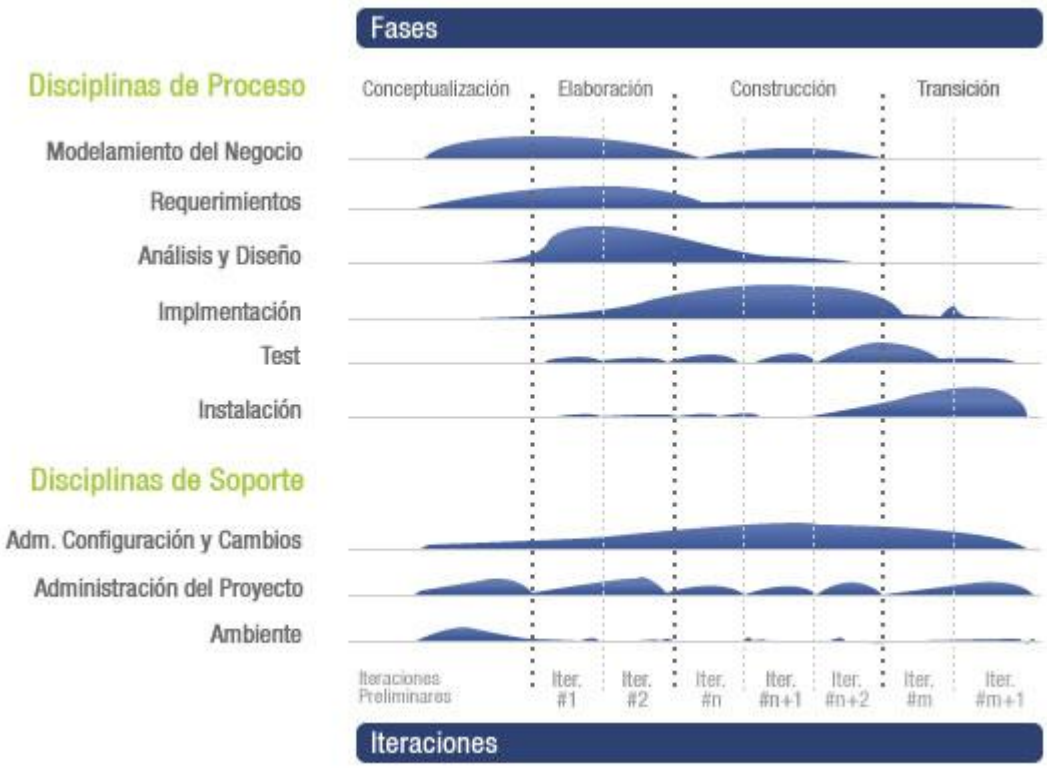


Figura 4. Metodología RUP.

Los elementos que utiliza RUP para describir la metodología son: actividades, trabajadores y artefactos. Estos se explican a continuación:

- Actividades: Son los procesos que se llegan a determinar en cada iteración.
- Trabajador: Define el comportamiento y las responsabilidades de un individuo dentro del proceso de desarrollo.

- Artefactos: Un artefacto puede ser un documento, un modelo, o un elemento de modelo.

RUP permite guiar cada rol desempeñado por los integrantes del proyecto definiendo claramente las actividades y los artefactos correspondientes a cada uno de ellos.

Complementario a RUP, el Lenguaje Unificado de Modelado (UML, por sus siglas en inglés), “es un lenguaje visual para especificar, construir y documentar un sistema de software” (23).

UML ofrece un estándar para describir un plano, o sea un modelo del sistema. Para ello incluye aspectos conceptuales tales como procesos de negocios y funciones del sistema, y aspectos concretos como expresiones de lenguajes de programación, esquemas de bases de datos y componentes de software reutilizables.

Para el Proceso Unificado de Desarrollo de Software la arquitectura es un elemento vital y abarca decisiones importantes sobre la organización del sistema, los elementos estructurales que compondrán el mismo, así como las interfaces, junto con sus comportamientos y colaboraciones; la composición de los elementos estructurales y el comportamiento de estos en subsistemas progresivamente más grandes, haciendo énfasis en el estilo arquitectónico como guía de esta organización. Además la arquitectura no solo involucra la estructura y el comportamiento, sino también el uso, la funcionalidad, el rendimiento, la flexibilidad, la reutilización, la facilidad de comprensión, las restricciones y compromisos económicos y tecnológicos, y la estética.

La arquitectura en RUP se representa mediante varias vistas que se centran en aspectos concretos y describen las principales partes del sistema. Estas vistas son la de casos de uso, lógica, procesos, implementación y despliegue del sistema, las cuales constituyen un extracto de los elementos arquitectónicamente más significativos de cada modelo correspondiente. Como vemos la arquitectura muestra la visión común del sistema completo en la que el equipo de proyecto y los usuarios deben estar de acuerdo, por lo que describe los elementos del modelo que son más importantes para su construcción, los cimientos del sistema que son necesarios como base para comprenderlo, desarrollarlo y producirlo económicamente.

La metodología de desarrollo RUP plantea que el rol de arquitecto es el responsable por el desarrollo y mantenimiento de la arquitectura de software, la que incluye como se ha visto las principales decisiones técnicas y las restricciones sobre el diseño e implementación del proyecto.

El arquitecto participa en los flujos de trabajo: Requerimientos, Análisis y Diseño e Implementación, y presenta asignadas las actividades y artefactos que se muestran en la siguiente figura:

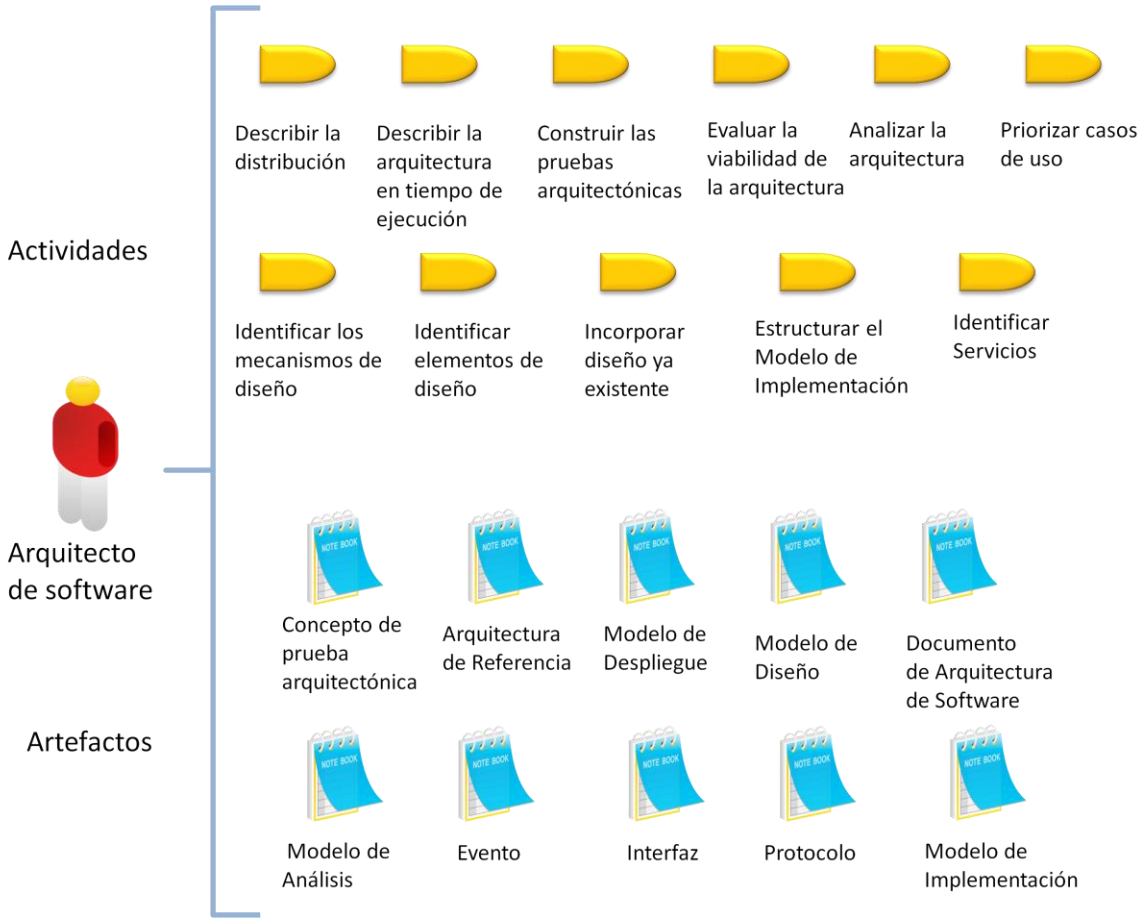


Figura 5. Artefactos y responsabilidades del arquitecto propuestos por RUP.

El arquitecto tiene dentro de RUP el soporte de UML y del Proceso Unificado para lograr sus objetivos, ya que UML posee construcciones potentes para la formulación de la arquitectura, y el Proceso Unificado ofrece un conjunto de directrices detalladas sobre lo que constituye una buena arquitectura.

Debido a la posibilidad que ofrece de llevar una construcción bien planificada con la generación de artefactos bien definidos, RUP permiten alcanzar rápidamente un grado de certificación en el desarrollo del software, posibilitando además crear un producto de software robusto y capaz de evolucionar con el tiempo. Por todo lo anteriormente explicado, RUP es una de las metodologías más utilizadas en la construcción de sistemas complejos, y se ha seleccionado como la metodología para el desarrollo del SIGM.

1.1.6 IMPORTANCIA DE UNA BUENA ARQUITECTURA DE SOFTWARE

Un sistema de software complejo necesita de una arquitectura para que los desarrolladores puedan progresar hasta tener una visión común del sistema. Se requiere una arquitectura para:

Comprender el sistema: Para que una organización desarrolle un sistema, dicho sistema debe ser comprendido por todos los que vayan a intervenir en el, para esto se crean las diferentes vistas arquitectónicas del sistema. Uno de los primeros requisitos que debe tener lugar en la descripción de la arquitectura es que se debe capacitar a todos los que intervienen en el sistema, tanto desarrolladores como usuarios finales para que comprendan que se está haciendo con suficiente detalle como para facilitar su propia participación.

Organizar el desarrollo: Una interfaz bien definida comunica eficientemente a los desarrolladores de ambas partes que necesitan saber sobre lo que los otros equipos están haciendo. El arquitecto es el responsable de definir claramente los subsistemas y sus interfaces mediante la aplicación de patrones de diseño adecuados, logrando así reducir la carga de comunicación entre los grupos de trabajo de los diferentes subsistemas.

Fomentar la reutilización: Los componentes de software reutilizables están diseñados y probados para encajar, y así el tiempo de construcción y el costo son menores. Escogiendo y diseñando con cuidado los subsistemas reutilizables, el arquitecto ayuda a los desarrolladores a encontrar elementos reutilizables de manera poco costosa, así como disminuir el tiempo de desarrollo del proyecto.

Hacer evolucionar el sistema: El sistema debe ser en si mismo flexible o tolerante a los cambios. Una arquitectura adecuada permite que los desarrolladores sean capaces de modificar partes del

diseño e implementación sin tener que preocuparse por los efectos inesperados que puedan tener repercusión en el sistema. Las arquitecturas de sistemas pobres, por el contrario, suelen degradarse con el paso del tiempo y necesitan ser parcheadas hasta que al final es imposible actualizarlas con un costo razonable.

Una arquitectura de software adecuada resalta aquellos aspectos estructuralmente importantes, tanto funcionales como no funcionales que ayudan a comprender y manejar la estructura de las aplicaciones por los desarrolladores así como a reutilizar dicha estructura; contribuye a eliminar muchos riesgos y errores de diseño, desarrollo e implantación; y permite un desarrollo paralelo, aumentando la productividad.

1.2 ARQUITECTURA DEL SISTEMA NACIONAL DE SALUD

Como parte de la estrategia para la informatización del Sistema Nacional de Salud Pública en Cuba, el grupo de arquitectura MINSAP-MIC ha establecido un grupo de normas y tecnologías para el desarrollo de aplicaciones informáticas destinadas a este sector. Estas premisas se han establecido con el objetivo de que se garantice la continuidad y sostenibilidad de los productos que se obtengan, y contribuir con su integración como pauta fundamental.

Dentro de estas normas, se establece que cualquier aplicación informática desarrollada para la salud en Cuba, debe cumplir los siguientes requerimientos:

- Utilizar lenguaje de programación del lado del servidor PHP5.
- Funcionar sobre servidor Linux, distribución Debian (Sarge).
- Funcionar sobre Servidor Web Apache 2.0.
- Utilizar como sistema gestor de base datos MySQL5.0.
- Consumir o brindar servicios web para interactuar con otros sistemas.

Los sistemas desarrollados deben poder desplegarse en los servidores disponibles en Infomed.

Además las aplicaciones deben interactuar entre sí suministrando y/o consumiendo servicios web, según el esquema de seguridad desarrollado por Softel. Este mecanismo está implementado en el componente de seguridad (SAAA), que garantiza los tres elementos básicos de la seguridad (24):

• Confidencialidad: Que la información sea revelada sólo a los usuarios autorizados, en la forma y tiempo determinado.
• Integridad: Que la información sea modificada (incluyendo su creación y borrado) sólo por personal autorizado.
• Disponibilidad: Que la información sea utilizable cuándo y cómo lo requieran los usuarios autorizados.

El Sistema de Información para la Salud (SISalud) es un sistema desarrollado en el marco de la informatización del Sistema Nacional de Salud Pública en Cuba. Este presenta una arquitectura orientada a servicios (SOA) y un desarrollo basado en componentes (Ver Anexo 1: Representación estructural de SISalud).

1.2.1 ARQUITECTURA DEL SISTEMA DE INFORMACIÓN PARA LA SALUD (SISALUD)

Arquitectura Orientada a Servicios

SOA permite que los servicios puedan ser provistos a aplicaciones de usuarios finales, procesos de negocio ejecutables, o bien, otros servicios, a través de la publicación y descubrimiento de la interfaz de los servicios. Para lograr la comunicación entre los diferentes nodos se utilizan los servicios web, que recopilan una colección de protocolos y estándares para intercambiar datos entre aplicaciones.

SISalud utilizó SOAP tomando en cuenta los beneficios de su uso que se listan a continuación:

- Reusabilidad de servicios**:** Disminución de tiempos y costos de desarrollo de las aplicaciones, así como de los riesgos al utilizar servicios disponibles ya desarrollados y probados.
- Interoperabilidad de aplicaciones**:** Disminución de la complejidad de interrelación e integración entre aplicaciones al utilizar un protocolo común de entendimiento, que abstrae de la tecnología y ubicación de los servicios.

Desarrollo basado en componentes

Un componente de software es "una pieza de código preelaborado que encapsula alguna funcionalidad" (25). Son independientes entre ellos, y tienen su propia estructura e implementación. El Desarrollo de Software Basado en Componentes (DSBC) se centra en el desarrollo de grandes sistemas de software mediante la integración de componentes de software ya existentes.

SISalud llevó a cabo un desarrollo basado en componentes tomando en cuenta los beneficios de su uso que se listan a continuación:

- *Reutilización del software***.** Se pueden utilizar componentes desarrollados por otros a través de sus interfaces. Con esto se acorta el ciclo de desarrollo y disminuyen los riesgos al utilizar componentes ya probados.
- *Simplifica las pruebas***.** Las pruebas se ejecutan probando cada uno de los componentes independientemente antes de probar el sistema integrado.
- *Simplifica el mantenimiento del sistema***.** Al existir un débil acoplamiento entre componentes, el desarrollador puede actualizar y/o agregar componentes, sin afectar otras partes del sistema.

El sistema logró un desarrollo basado en componentes guiado por una arquitectura orientada a servicios definiendo a los componentes como proveedores o consumidores de servicios. Esta característica es aplicable debido a que un componente es una entidad ejecutable e independiente, y cada proveedor de servicios debe definirse igualmente como una unidad ejecutable e independiente, prevaleciendo la alta cohesión y el bajo acoplamiento entre ellos. Además los componentes utilizan interfaces para la comunicación, y en SOA existe una interfaz pública de servicios suministrados y las interacciones son a través de esta.

1.2.2 COMPONENTES DESARROLLADOS POR SISALUD.

En la actualidad SISalud presenta varios componentes desplegados que pertenecen al Registro Informatizado de Salud (RIS). Estos son:

Componente de Seguridad (SAAA).

Está basado en el modelo de Autenticación, Autorización y Auditoría (AAA). La autenticación debe ser la primera acción del usuario en el sistema y consiste en suministrar un nombre de usuario único y una contraseña que debe ser de conocimiento exclusivo de la persona que se autentica. Si el usuario autenticado no se encuentra registrado se reporta un error de acceso. En caso contrario, se autoriza su acceso y se crea un certificado digital y se retornan todos los datos y permisos del usuario. Cada petición del usuario, autorizada o no, es registrada, así como el día, mes, año, hora, minuto y segundo en que se efectuó la acción, y si fue o no autorizada.

Registro del Ciudadano (RC)

Contiene los datos personales de cualquier ciudadano que trabaje en la salud y/o reciba sus servicios. Los datos fundamentales están relacionados con la información que se recoge por el carné de identidad, y se podrán ir adicionando otros que sean necesarios para los diferentes componentes que necesiten conectarse a este registro para buscar una persona.

Registro de Unidades de Salud (RUS).

Mantiene la información relativa a todas las Unidades de Salud. Registra campos como Nombre, Código, Dirección, Tipo de Unidad, atributos de capacidad (por ejemplo, cantidad de camas) y cualitativos (por ejemplo, categoría, perfil), entre otros.

Registro de Equipos Médicos (REM).

Contiene el registro de los equipos médicos que se encuentran en las unidades de salud y en las otras dependencias del MINSAP.

Registro de Equipos No Médicos (RENM).

Mantiene el registro de los equipos no médicos que se encuentran en las unidades de salud y en las otras dependencias del MINSAP.

Registro de Facultades (RF).

Permite registrar los egresados de las diferentes Facultades, así como asignarle el número de registro a cada profesional.

Registro del Clasificador Internacional de Enfermedades y Problemas de Salud (RCIEPS).

Gestiona la estructura de la Clasificación Internacional de Enfermedades y Problemas Relacionados con la Salud y permite realizar búsquedas dinámicas de acuerdo a criterios seleccionados por el usuario.

Registro de Problemas de Salud de la Atención Primaria (RPSAP).

Gestiona una clasificación especial para los problemas de salud que se presentan en el nivel de Atención Primaria de Salud. Permite realizar búsquedas dinámicas de acuerdo a criterios seleccionados por el usuario.

Registro de Personal de la Salud (RPS).

Mantiene el registro actualizado de todo el personal que trabaja en la salud. Orientado actualmente a registrar los profesionales médicos y no médicos, con toda la información relativa a ellos: datos como profesionales, graduaciones, grados científicos, unidad de salud a la que pertenecen, entre otros.

Entre los componentes en fase de implementación, se encuentran**:**

Registro de Servicios Médicos.

Mantiene actualizado el registro de Servicios Médicos para configurar los mismos de manera uniforme en las unidades de salud.

Registro de Ubicación

Permite mantener un registro actualizado de todas las provincias, municipios, localidades, calles y manzanas. Las localidades, calles y manzanas son importantes para poder configurar las Áreas de Salud y garantizar que funcione adecuadamente el Registro de Población.

Registro de Localidad.

Permite mantener un registro actualizado de consejo popular, circunscripción, zonas y CDR, con vistas a poder configurar las Áreas de Salud y garantizar que funcione adecuadamente el Registro de Población.

Registro de Áreas de Salud.

Gestiona la información de las Áreas de Salud a nivel nacional, permitiendo un control de las mismas, su composición según la estructura organizativa propuesta por la Atención Primaria de Salud, así como sus integrantes según la plantilla del Grupo Básico de Trabajo y Equipo Básico de Salud. Brinda a la población información actualizada sobre las características de las Unidades de Salud que brindan servicios, dónde se ubican, áreas geográficas que atiende, servicios que brinda, especialidades que ofrecen sus servicios.

Registro de Población.

Gestiona la información de la Historia de Salud Familiar de los Consultorios del Médico de Familia, que forman parte de los diferentes Equipos Básico de Salud (EBS), permitiendo la captación, organización y elaboración de reportes.

Registro de Enfermedades de Declaración Obligatoria.

Gestiona en tiempo real y con alcance nacional la información sobre la incidencia de las Enfermedades de Declaración Obligatoria, facilitando así la vigilancia epidemiológica de dichas enfermedades.

1.3 CONCLUSIONES

Tras analizar los elementos fundamentales que intervienen en la definición de la arquitectura de software, como son los estilos, patrones y lenguajes de descripción de la arquitectura, se fomentaron las bases para la definición posterior de la arquitectura del SIGM. Además se expusieron las pautas de desarrollo para las aplicaciones de la salud propuestas por el grupo de arquitectura MINSAP-MIC, con las cuales se logró identificar algunos de los requisitos no funcionales que debe cumplir el sistema. Finalmente se describió la arquitectura de SISalud, explicando las funciones de sus registros, con la intención de poder reutilizar la información proporcionada por alguno de los mismos.

CAPÍTULO 2: CARACTERÍSTICAS DEL SISTEMA

Entre las características principales de un sistema de software se debe mencionar cuál es su objetivo, definiendo las capacidades o funciones que el sistema debe cumplir. Además estas funciones suelen venir acompañadas de ciertas propiedades o cualidades como requisitos de disponibilidad, seguridad o necesidades de hardware que se deben tomar en cuenta en la definición de la arquitectura. Un sistema también se caracteriza a partir de la forma en que se estructuran sus componentes, así como por las tecnologías y herramientas informáticas empleadas para su creación y funcionamiento.
El presente capítulo cuenta con una breve descripción de los requisitos del software, sirviendo como guía para la selección del patrón arquitectónico a utilizar. Además se expone una síntesis del estudio realizado con el objetivo de identificar las tecnologías y herramientas propuestas para llevar a cabo el desarrollo.

2.1 ANÁLISIS DE LOS REQUISITOS

El sistema está constituido por un conjunto de módulos que gestionan de forma independiente su negocio, pero comparten la misma base de datos. El objetivo es tener concentrada toda la información obtenida en los estudios realizados por el CNGM, de forma que no exista redundancia y se puedan ejecutar consultas cruzadas sobre datos de diferentes registros. Cada módulo gestiona los datos correspondientes a su estudio, lo cual incluye insertar, modificar y realizar determinados reportes sobre los datos. Los módulos del sistema son:

- Módulo Registro Cubano de Enfermedades Genéticas (RECUEGEN)
- Módulo Registro Cubano de Malformaciones Congénitas (RECUMAC)
- Módulo Registro Cubano de Discapacitados (RECUDIS)
- Módulo Registro Cubano de Retraso Mental (RECURM)
- Módulo Registro Cubano de Gemelos (RECUGEM)
- Módulo Registro Cubano de Historias Clínicas (RECUHCL)
- Módulo Teleconsulta

Además de los requisitos expuestos anteriormente, el sistema debe permitir el acceso rápido desde cualquier Centro de Genética del país, sin los inconvenientes propios que incluye la instalación y mantenimiento del mismo en cada centro independientemente, por lo que se decide el desarrollo de una aplicación web.

CAPÍTULO 2: CARACTERÍSTICAS DEL SISTEMA

El sistema a desarrollar constituye una aplicación informática para la salud en Cuba, y por tanto debe adoptar las normas establecidas por el Grupo de Arquitectura MINSAP-MIC para este tipo de aplicaciones, cumpliendo los siguientes requerimientos:

- Desarrollado en PHP5.
- Funcionar sobre servidor Linux, distribución Debian (Sarge).
- Funcionar sobre Servidor Web Apache 2.0.
- Utilizar como sistema gestor de base datos MySQL5.0.
- Consumir o brindar servicios web para interactuar con otros sistemas utilizando el componente SAAA
- Ser capaz de desplegarse sobre la estructura de servidores presente en Infomed.

Dentro de los requerimientos arquitectónicos anteriormente mencionados, también se insertan un grupo de requisitos no funcionales que el sistema debe cumplir:

Apariencia o interfaz externa:

Se deben utilizar imágenes y colores identificados con el negocio del sistema. La interfaz externa debe estar diseñada para verse en cualquier resolución igual o superior a 1024x768.

Usabilidad:

La aplicación informática debe garantizar un acceso fácil y rápido, contando con un menú que satisfaga las necesidades de los usuarios. Este podrá ser usado por cualquier persona que posea conocimientos básicos en el manejo de una computadora y del ambiente web.

Rendimiento:

Los tiempos de respuestas deben ser generalmente rápidos al igual que la velocidad de procesamiento de la información.

Soporte:

Se debe asegurar el soporte para los usuarios de manera que se puedan satisfacer sus necesidades a partir de mejoras, una vez puesta en marcha la aplicación. Para ello se crearán una serie de manuales de usuarios y videos tutoriales, y se mantendrá la asistencia a los usuarios.

Seguridad:

El sistema necesita información que contienen algunos de los registros del SISalud, específicamente el Registro de Ciudadanos y el Registro de Unidades de Salud. Para consumir los servicios web que brindan estos componentes, es necesario utilizar el componente de seguridad SAAA.

La información genética, por su naturaleza, corresponde al tipo de información más sensible, dado que no sólo desvela información del individuo al que se realiza el análisis (información que es permanente y que no se modifica con el tiempo), sino que también proporciona información de los familiares o su descendencia. Debido a la gran preocupación relacionada con posibles problemas de discriminación y estigmatización debidos al mal uso de la información genética, es necesaria la implantación de un mecanismo de seguridad estricto que regule el acceso a la misma. Este debe ser administrado de forma centralizada e independiente a cualquier sistema, y sus implicados deben estar bien identificados y comprometidos con el resguardo de la información.

Por lo anteriormente expuesto, el sistema debe tener un mecanismo propio para gestionar la seguridad a través de niveles de acceso a la información. Los permisos al ejecutar cualquier acción deben estar de acuerdo con el nivel jerárquico de acceso que presente el usuario en cada módulo, el cual es definido por los administradores del sistema.

Software:

Se requiere para el funcionamiento del sistema disponer de un servidor que cuente con Sistema Operativo Linux, Apache 2.0 y MySQL 5.0 o versiones superiores. Los usuarios del sistema deberán contar con un navegador Internet Explorer 5.5 o Mozilla Firefox 2.0 o superior, para poder acceder a las opciones que brinda el sistema.

Hardware:

Para el desarrollo y ejecución de la aplicación se necesitará:

Para el servidor de aplicación:

- Microprocesador Pentium IV a 3.0 GHz o superior.
- 1GB de RAM o superior.

Para el Cliente:

- Microprocesador Pentium a 233 MHz (Recomendado: Pentium a 500MHz o superior)
- 64 MB de RAM (Recomendado*:* 128 MB RAM o superior)
- 52 MB de espacio de disco duro. (Recomendado: 120MB para la instalación completa del Internet Explorer 5.5)
- Conexión al servidor a través de MODEM o tarjeta de red.

Además es necesario contar con una impresora para poder imprimir los diferentes tipos de reportes.

Disponibilidad:

El sistema debe ser capaz de funcionar por si solo en caso de que los servicios de los diferentes componentes de que utiliza SISalud no estén disponibles. Se debe garantizar el funcionamiento de la aplicación durante las 24 horas del día y los siete días de la semana, con el menor tiempo posible de recuperación ante fallos. Se deben crear copias de respaldo periódicas que puedan restaurar el sistema en caso de fallo crítico o pérdida total de la información.

Requisitos Legales:

Las tecnologías y herramientas en que estará basada la aplicación informática deberán cumplir con las licencias de software libre.

Persistencia:

La información debe almacenarse en bases de datos con carácter permanente con el objetivo de poder realizar análisis de la misma con el transcurso de los años.

2.2 TECNOLOGÍAS Y HERRAMIENTAS INFORMÁTICAS DE SOPORTE AL DESARROLLO

Brindar soporte al desarrollo del sistema con el uso de herramientas informáticas que faciliten la construcción del mismo, y definir las tecnologías a utilizar por el software, es considerado una de las buenas prácticas de la arquitectura de software. Para ello se debe realizar un estudio exhaustivo de las herramientas y lenguajes que se adecuan a los requerimientos y presentan un espectro de funciones correspondientes a las necesidades.

Dada la política hostil que el gobierno norteamericano mantiene sobre nuestro país, y el bloqueo económico que prohíbe el acceso a las nuevas tecnologías de la informática y las comunicaciones, la premisa en la selección de las herramientas y tecnologías de soporte al desarrollo fue su clasificación como software libre.

2.2.1 CONTROL DE VERSIONES

Un sistema de control de versiones es un software que administra el acceso a un conjunto de ficheros y mantiene un historial de cambios realizados. El control de versiones es útil para guardar cualquier documento que cambie con frecuencia, como el código fuente de un programa.

Normalmente consiste en una copia maestra en un repositorio central, y un programa cliente con el que cada usuario sincroniza su copia local. Esto permite compartir los cambios sobre un mismo conjunto de ficheros actualizando solamente aquellos modificados, creando copias de seguridad centralizadas, manteniendo un historial de cambios que permite volver a un estado anterior, y accediendo de forma remota con seguridad.

Entre los principales sistemas de control de versiones se encuentran el Concurrent Versions System (CVS) y Subversion.

El **CVS**, se ha hecho popular en el mundo del software libre y sus desarrolladores difunden el sistema bajo la licencia GPL. Este sistema de control de versiones utiliza una arquitectura cliente-servidor: un servidor guarda la(s) versión(es) actual(es) del proyecto y su historial. Los clientes se conectan al servidor para sacar una copia completa del proyecto. Esto se hace para que eventualmente puedan trabajar con esa copia y más tarde ingresar sus cambios, incrementándose el número de versión de

todos los archivos automáticamente y actualizando el registro de cambios en el servidor. Actualmente existen muchas versiones de CVS implantadas en diferentes sistemas operativos.

Entre sus principales limitaciones encontramos que los archivos en el repositorio sobre la plataforma CVS no pueden ser renombrados, estos deben ser agregados con otro nombre y luego eliminados. Además el protocolo CVS no provee un mecanismo para que los directorios puedan ser eliminados o renombrados.

Subversion (SVN) fue diseñado posteriormente con el objetivo principal de reemplazar al popular CVS. SVN mantiene las ideas fundamentales de CVS, pero suple sus carencias y evita sus errores. Sobre este punto, la principal característica que presenta es que los archivos versionados no tienen cada uno un número de revisión independiente, sino que todo el repositorio tiene un único número de versión que identifica un estado común de todos los archivos en cierto punto del tiempo.

Las principales ventajas de utilizar SVN como sistema de control de versiones se relacionan a continuación:

- Mantiene versiones no sólo de archivos, sino también de directorios a través de copias y renombrados.
- Mantienen versiones de los metadatos asociados a los directorios.
- Mantiene además de los cambios en el contenido de los documentos, la historia de todas las operaciones de cada elemento, incluyendo la copia, cambio de directorio o de nombre.
- Presenta atomicidad de las actualizaciones. Una lista de cambios constituye una única transacción o actualización del repositorio, minimizando así el riesgo de que aparezcan inconsistencias entre distintas partes del repositorio.
- Presenta la posibilidad de seleccionar el protocolo de red a utilizar en la transacción. Además de un protocolo propio (svn), puede trabajar sobre http (o https). La capacidad de funcionar con un protocolo tan universal como el http simplifica la implantación y universaliza las posibilidades de acceso.
- En las transacciones se transmiten sólo las diferencias y no los archivos completos, permitiendo aprovechar mejor el ancho de banda de la red.
- Permite selectivamente el bloqueo de archivos.

Luego de analizar las características de ambos sistemas de control de versiones se selecciona Subversion como una de las herramientas informáticas de soporte para el desarrollo del SIGM.

2.2.2 HERRAMIENTA CASE

Las Herramientas CASE (Computer Aided Software Engineering), son aplicaciones informáticas destinadas a contribuir con el desarrollo del software y a aumentar la productividad. Estas permiten modelar los sistemas de software abarcando todos los artefactos y actividades propuestos por una determinada metodología de software. Algunas de las herramientas CASE son ArgoUML, Rational Rose, Visual Paradigm, BOUML, y Enterprise Architect.

Entre las herramientas CASE de uso libre se encuentran ArgoUML, BOUML y Visual Paradigm:

ArgoUML es una sencilla herramienta desarrollada en Java bajo licencia BSD que soporta solo los diagramas definidos por UML1.4 Es una plataforma independiente por lo que puede ser instalado en cualquier sistema operativo. Presenta herramientas de ingeniería inversa, se encuentra disponible en 10 lenguajes y exporta los diagramas en formato GIF, PNG, EPS, entre otros.

BOUML es una herramienta CASE distribuida bajo licencia GPL. Esta herramienta permite trabajar con UML 2, es rápida y sencilla de utilizar y apenas consume memoria. Permite generar código para Java y C++, así como realizar ingeniería inversa. Es capaz de generar documentación en varios formatos entre los que se encuentra HTML. Es multiplataforma para sistemas operativos como Linux y Windows.

Visual Paradigm for UML Community Edition (VP-UML CE) es una potente herramienta CASE con licencia libre para uso de la comunidad de desarrolladores. Permite la modelar UML 2.1, con 13 tipos diferentes de diagramas que pueden ser exportados como imágenes en formato JPG, PNG, entre otros. Proporciona a los desarrolladores una plataforma con interfaz amigable que les permite diseñar un producto con calidad de forma muy rápida. Puede ser extendido, pues presenta soporte al diseño personalizado, permitiendo incorporar nuevas formas y notaciones, mediante el uso de imágenes o iconos importados. VP-UML CE permite ser instalado en múltiples plataformas.

Luego de analizadas las características de las herramientas informáticas anteriores, se selecciona como herramienta CASE para dar soporte al desarrollo del SIGM a Visual Paradigm for UML Community Edition.

2.2.3 ENTORNO DE DESARROLLO

Un entorno de desarrollo integrado (IDE) es un programa compuesto por un conjunto de herramientas integradas que facilitan inmensamente el trabajo del programador. Normalmente consta de un editor de código, un compilador, un depurador y un constructor de interfaz gráfica (GUI). Permite al desarrollador programar en uno o varios lenguajes de programación.

Entre los IDEs con licencia gratuita para el desarrollo se encuentran Aptana Studio y Eclipse.

Aptana Studio es un IDE de código libre para aplicaciones de la web 2.0. Está focalizada en el desarrollo web en Java, con soporte a HTML, CSS y Javascript, así como opcionalmente a otras tecnologías como PHP, y Ajax. Está disponible como una aplicación independiente o como plugin para Eclipse. Puede distribuirse para todos los sistemas operativos más comunes: Windows, Linux y Mac OS.

Eclipse es un IDE para todo tipo de aplicaciones, inicialmente desarrollado por IBM, y actualmente gestionado por la Fundación Eclipse, una organización independiente sin ánimo de lucro que fomenta una comunidad de código abierto. La característica clave de Eclipse es la extensibilidad. Eclipse es una gran estructura formada por un núcleo y muchos plugins que van conformando la funcionalidad final. La forma en que los plugins interactúan es mediante interfaces o puntos de extensión. Los lenguajes con proyectos más importantes son: Java, PHP y C/C++. Además existen paquetes que ayudan también con lenguajes para web como HTML y CSS.

Eclipse permite el desarrollo de aplicaciones utilizando Subversion como sistema de control de versiones, y brinda la facilidad de agregarle las librerías de los frameworks a utilizar en el desarrollo. Presenta un potente editor de código que posibilita realizar completamiento del mismo, incluso del código contenido en las librerías externas agregadas.

Eclipse PDT (PHP Development Tools) v 0.7 es un plugin con soporte para Eclipse Version: 3.2.1 o superior, que abarca un conjunto de herramientas necesarias para desarrollar PHP, permitiendo crear aplicaciones eficientes en menor tiempo.

Luego de analizadas las características propias de cada IDE identificado anteriormente, se selecciona para el desarrollo del SIGM a Eclipse integrado con el plugin PDT. Además se utilizará el plugin Subclipse v1.0.3 para conectarse al servidor de Subversion desde cada máquina cliente. Complementario a Eclipse, se empleará para el diseño de las páginas web a QuantaPlus, que es un editor HTML altamente estable bajo licencia GPL. Este presenta soporte para HTML, XML, PHP, CSS, JavaScript, plantillas, plugins, completamiento de código y resaltado de sintaxis. QuantaPlus está diseñado para ser eficiente y fácil de usar, presentando asistentes para creación de tablas, enlaces y páginas en blanco, previsualización y contiene un analizador que informa acerca de la correcta creación de las páginas. Su característica principal es que permite al usuario una máxima extensibilidad. Presenta gran cantidad de documentación en su sitio web, donde se publican de forma constante las nuevas versiones que van surgiendo.

2.2.4 FRAMEWORK DE DESARROLLO

Un framework o marco de trabajo, es una estructura conceptual básica usada para resolver un problema complejo. Un framework de desarrollo de aplicaciones puede estar basado en un conjunto de conceptos, modelos, metodologías, componentes, herramientas de administración y de diseño, entre otras cosas; de manera que facilite la construcción de las aplicaciones, manteniendo el orden y la homogeneidad en las partes y piezas que conforman la aplicación.

Un framework de software es un diseño reutilizable para un sistema de software. Esto se expresa como un conjunto de clases abstractas y el modo en que sus instancias colaboran para un tipo específico de software. Un framework de desarrollo puede incluir programas de soporte, librerías de código, códigos script, u otras aplicaciones que ayuden al desarrollo.

Utilizar un framework en el desarrollo de la aplicación permite aprovechar todo lo que una reutilización puede aportar: desarrollo más rápido al incorporar al sistema componentes ya construidos, y disminución de riesgos al utilizar un diseño y código ya probados. Sin embargo al utilizar un framework

de terceros se está a expensas de tener que reescribir el código en caso de que los proveedores decidan descontinuarlo y se necesite seguir manteniendo el sistema.

Para las aplicaciones desarrolladas en PHP existe una cantidad considerable de frameworks que pueden ser utilizados. Entre los que son libres se pueden encontrar a Zend Framework, Kumbia y Symfony.

Zend Framework (ZF) es un marco de trabajo de código abierto orientado a objetos, implementado en PHP 5 y bajo la licencia BSD. Tiene como objetivo simplificar el desarrollo web encapsulando al mismo tiempo las mejores prácticas de programación en PHP, y provee componentes para el patrón MVC. Entre los componentes que proporciona se encuentran aquellos que se utilizan con frecuencia en aplicaciones web, incluyendo la autenticación y autorización a través de listas de control de acceso (ACL), la configuración de aplicación, los datos de la memoria caché, el filtrado y validación de los datos proporcionados por el usuario, la internacionalización, las interfaces para Ajax, así como la composición y entrega de correo electrónico.

Kumbia es un framework para el desarrollo de aplicaciones web escrito en PHP5. Su principal objetivo es fomentar la velocidad y eficiencia en la creación y mantenimiento de aplicaciones web. Ha sido probado en software comercial y educativo con variedad de demanda y funcionalidad. Este framework está basado en los conceptos de compatibilidad y flexibilidad; además su instalación, configuración y aprendizaje son muy fáciles de realizar; brindando soporte a prácticas y patrones de programación productivos y eficientes.

Symfony es un framework bajo licencia MIT que sigue el patrón arquitectónico MVC para desarrollar aplicaciones web comerciales, gratuitas y/o de software libre escritas en PHP5. Tiene como objetivo acelerar la creación y mantenimiento de aplicaciones web y sustituir repetitivas tareas de codificación; proporcionando para esto varias herramientas que siguen la mayoría de mejores prácticas y patrones de diseño para la web. Este framework es fácil de instalar y extender, lo que permite su integración con librerías desarrolladas por terceros. Presenta también herramientas de configuración que lo hacen lo suficientemente flexible como para adaptarse a las necesidades de la aplicación que se desea desarrollar.

Otras de las características de Symfony son que brinda soporte a la internacionalización y que es independiente del SGBD. Se puede ejecutar tanto en plataformas *nix, Unix y Linux, como en plataformas Windows. Su código fuente incluye más de 8.000 pruebas unitarias y funcionales, y ha sido utilizado en numerosos proyectos reales como Yahoo Bookmarks y Yahoo Answers. Las aplicaciones que emplean Symfony, pueden controlar hasta el último acceso a la información, debido a que el framework brinda herramientas para ello, e incluye por defecto protección contra ataques XSS.

La empresa que lo ha creado está comprometida con el framework, ya que no se sustenta directamente del mismo, sino de las aplicaciones que crea con él; por tanto dicha empresa se interesa por aspectos como el rendimiento, la buena documentación, y un soporte a largo plazo. Las versiones estables del framework se mantienen durante tres años sin cambios, pero con una continua corrección de los errores conocidos. Además, Symfony cuenta con una documentación muy amplia, que incluye miles de páginas en el wiki oficial, tutoriales y una guía gratuita traducida al español conformada por más de 400 páginas.

Concluido el análisis anterior, se selecciona como framework para el desarrollo del SIGM a Symfony.

2.2.5 CONTROL DE ERRORES

Los sistemas para el control de errores (bug tracking en inglés), permiten la gestión y seguimiento de errores de forma organizada y entendible. Controlan los errores encontrados, registrando quién los descubre y arregla, manteniendo actualizado en todo momento al desarrollador sobre el estado de los mismos. Un sistema de control de errores ayuda a evitar la pérdida de errores en la implementación, contribuyendo con esto a que un software deficiente no llegue a manos del cliente.

Mantis es un popular software de seguimiento de errores lanzado bajo los términos de la licencia GPL. Está escrito en el lenguaje PHP y puede trabajar con los sistemas gestores de base de datos MySQL, MS SQL y PostgreSQL. Mantis es multiplataforma, por lo que puede ser instalado en Windows, Linux, Mac OS, OS/2 u otros sistemas operativos que interpreten PHP. Es un sistema montado sobre la web, fácil de instalar y utilizar, que permite realizar búsquedas y filtros, seguimiento de defectos específicos, notificaciones vía correo electrónico, y se integra con CVS o Subversión. Opcionalmente puede estar integrado con una wiki y un chat. Mantis presenta soporte para proyectos,

sub-proyectos y categorías, y los usuarios pueden tener diferentes niveles de acceso, presentando vistas personalizadas.

Trac es una aplicación distribuida bajo la licencia BSD modificada, basada en el seguimiento de fallos y la gestión de proyectos. Fue desarrollada en lenguaje de programación Python y puede trabajar sobre los gestores de base de datos MySQL, Postgre o SQLite. Está desarrollado como una aplicación web y presenta contenidos wiki. Integrado con el sistema de control de versiones Subversion, esta herramienta funciona como una interfaz web que permite mostrar y mantener un registro de los cambios realizados (con marcas de colores que lo hacen muy fácil de entender), el origen de los mismos, y otras características, facilitando así el seguimiento de los errores.

Trac brinda gestión de usuarios y mecanismos de gestión de tickets donde se pueden especificar los errores, su prioridad, el estado actual de los mismos, los responsables del cambio, y la asignación de tareas. Además presenta herramientas para la realización de reportes, que permiten buscar en un rango de versiones seleccionado los errores existentes, verificar quiénes han arreglado o introducido nuevos errores; así como crear reportes personalizados a través de consultas SQL. Este sistema mantiene a los desarrolladores actualizados en todo momento sobre los acontecimientos y los cambios dentro de un proyecto.

Trac presenta una interfaz interactiva y amigable, que permite la integración de todas sus páginas, presenta una amplia gama de formatos en la descripción de los errores y en el envió de mensajes, y crea vínculos y referencias entre los errores, las tareas, los ficheros y las páginas de documentación. Además ofrece un calendario que muestra cronológicamente todos los eventos del proyecto y una breve descripción de los mismos, facilitando la visión general del proyecto y el seguimiento de sus progresos. Trac contiene un plan de trabajo que provee una vista sobre las tareas del proyecto y ayuda a planificar y mantener el desarrollo del software.

Luego del análisis de las características de los diferentes sistemas de control de versiones descrito anteriormente, se decide utilizar a Trac como herramienta de soporte para el desarrollo del SIGM.

2.3 PATRONES DE ARQUITECTURA

El sistema enmarca varios procesos para gestionar la información de sus módulos, es por esto que debe contar con una base de datos que almacene los datos del dominio. Algunos de estos procesos son:

- Insertar la historia clínica de un paciente.
- Buscar la historia clínica de un paciente.
- Generar reportes.

El propósito general del sistema es tomar datos almacenados y mostrarlos al usuario, quien a su vez introduce modificaciones en los mismos, que son reflejadas en el almacenamiento. Dado que el flujo de información ocurre entre el almacenamiento y la interfaz, la lógica podría indicar unir ambas piezas para reducir la cantidad de código y optimizar el sistema.

Sin embargo, esta idea es antagónica al hecho de que la interfaz pueda cambiar en algún momento de acuerdo a la clase de dispositivos que se utilice para mostrarla. Actualmente solo se consultará el sistema desde máquinas clientes en los Centros de Genética Médica, pero en un futuro podría necesitarse recoger los datos de los pacientes mediante PDAs u otro tipo de dispositivos móviles desde cualquier punto del país. Además, la programación de interfaces requiere habilidades muy distintas de las necesarias en la programación de la lógica del negocio, y el desarrollo de la misma puede llevarse a cabo por especialistas de diseño con escasos conocimientos de programación. Otro problema es que la aplicación incorpora lógica de negocio que va más allá de la simple transmisión de datos, y puede cambiar sin tener repercusión sobre la visualización de los resultados, o sobre los datos del dominio. Además la aplicación debe estar preparada para realizar modificaciones sobre los datos de los pacientes o incluso incorporar nuevos módulos al sistema sin producir un impacto global en la misma.

El patrón conocido como Modelo-Vista-Controlador (MVC) separa el modelado del dominio, la presentación y las acciones realizadas sobre los datos en tres elementos diferentes. El modelo administra el comportamiento y los datos del dominio de la aplicación, y responde a instrucciones de cambiar su estado desde el controlador. La vista tramita la visualización de la información, y el controlador interpreta las acciones de los dispositivos de entrada de datos, realiza los procesos

propios de la aplicación e informa al modelo y/o a la vista para que cambien su estado según resulte apropiado.

Este patrón arquitectónico brinda soporte para múltiples vistas, pues la vista es independiente del modelo, posibilitando por ejemplo que múltiples páginas de una aplicación de web puedan utilizar el mismo modelo de objetos, mostrado de maneras diferentes. Además permite una adaptación total al cambio. Debido a que los requerimientos de interfaz de usuario tienden a cambiar con mayor rapidez que las reglas de negocios, y los usuarios pueden preferir distintas opciones de representación, o requerir soporte para nuevos dispositivos como teléfonos celulares o PDAs, este patrón permite agregar nuevas opciones de presentación, que generalmente no afectan al modelo, ya que este último no depende de las vistas.

El patrón MVC se ve frecuentemente en aplicaciones web, donde “la vista está conformada por la página HTML y el código que provee a la página de datos dinámicos, el modelo son los datos del dominio gestionados por un SGBD, y el controlador representa la lógica de negocio” (11).

Debido a los argumentos planteados anteriormente, se define como patrón arquitectónico a utilizar en el SIGM al MVC en su variante II, por lo que el sistema cumple con el siguiente flujo de control:
1. El usuario interactúa con la interfaz de usuario.
2. El controlador recibe la notificación de la acción realizada por el usuario y gestiona el evento que llega.
3. El controlador accede al modelo, modificándolo de forma correspondiente a la acción solicitada por el usuario.
4. El controlador delega a los objetos de la vista la tarea de desplegar la interfaz de usuario, pasándole los datos obtenidos desde el modelo.
5. La interfaz de usuario espera nuevas interacciones del usuario, comenzando el ciclo nuevamente.
El modelo no tiene conocimiento directo sobre el controlador ni la vista.

2.3 ORGANIZACIÓN ESTRUCTURAL DEL SISTEMA

Para el desarrollo de la aplicación se utilizará el framework Symfony que implementa el patrón arquitectónico MVC, y define las instancias que compondrán cada elemento del mismo. El sistema quedaría entonces conformado como se ilustra en la siguiente figura.

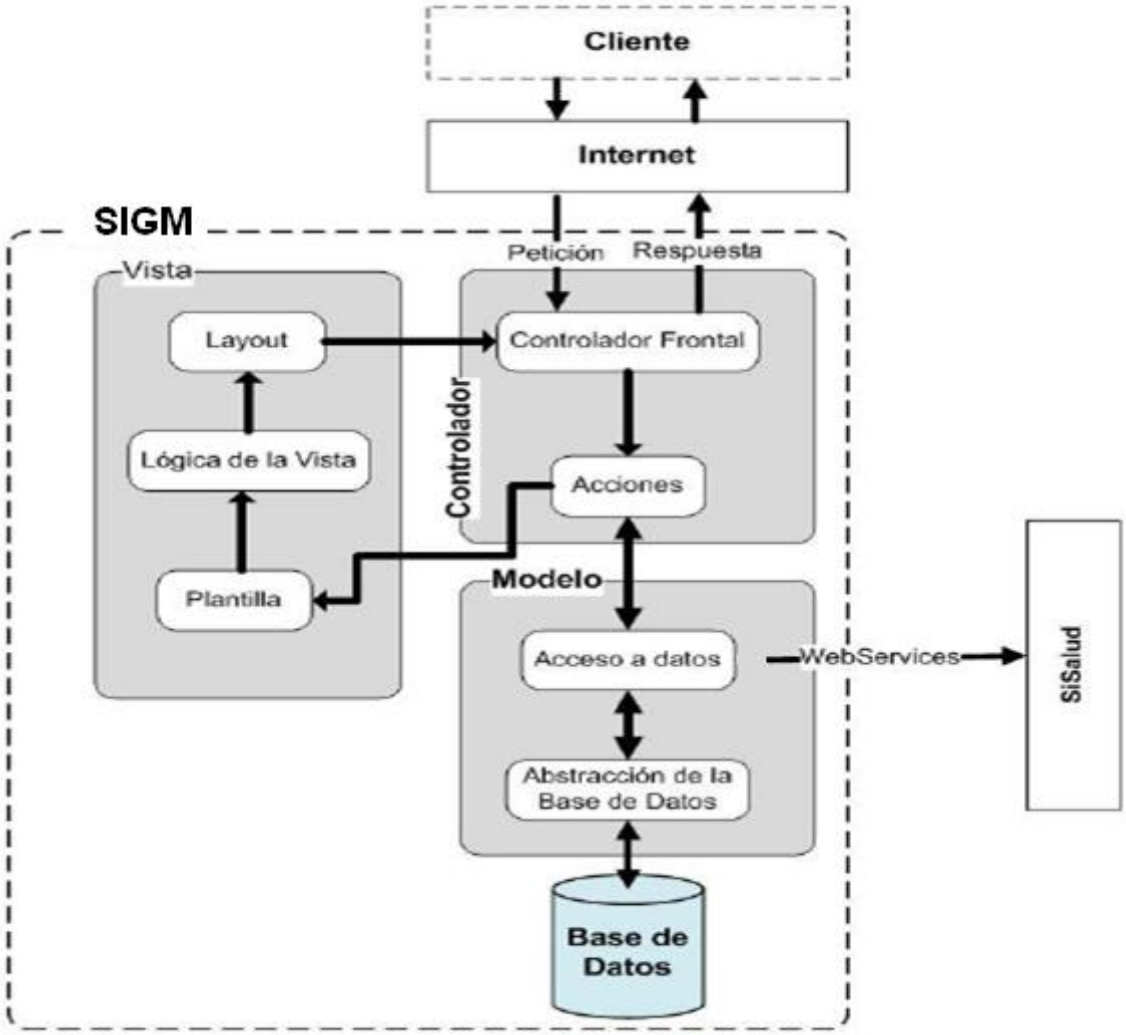


Figura 6. Organización estructural del sistema.

2.3.1 DESCRIPCIÓN DE LOS COMPONENTES

Para lograr un mayor entendimiento de la organización estructural del sistema se explicarán a continuación los elementos que la conforman.

El **modelo** solo se encarga del acceso a los datos almacenados en el gestor de base de datos. Para obtener un acceso a los datos de forma óptima e independiente del SGBD, el modelo ha sido desglosado en la capa de **acceso a los datos** y en la capa de **abstracción de la base de datos**. La capa de acceso a datos brinda una interfaz que permite acceder a los datos desde un contexto orientado a objetos, traduciendo la lógica relacional a lógica de objetos. La capa de abstracción a base de datos permite crear consultas utilizando una sintaxis propia y se encarga de realizar la conversión engorrosa entre este lenguaje y la base de datos específica que se esté utilizando. De esta forma, las funciones que acceden a los datos emplean sentencias independientes de la base de datos, permitiendo migrar de SGBD o base de datos fácilmente. A la capa de acceso a datos se le agregarán algunas funcionalidades para que el sistema consuma los servicios web publicados por los registros de SISalud. La **base de datos** contiene los datos del dominio de la aplicación.

La **vista** está compuesta por las páginas web de la aplicación. Debido a que estas contienen información estándar como la cabecera y el pie de la página, o la navegación global, la vista es dividida en el **layout** y la **plantilla**. El layout contiene los elementos idénticos para todas las páginas y es global para la aplicación. La plantilla encapsula el contenido que cambia en el interior de la página, y se encarga de visualizar las variables definidas en el controlador. Para que estos componentes interaccionen entre sí correctamente, es necesario añadir cierto código denominado **lógica de la vista**.

El controlador ha sido dividido en el controlador frontal y las acciones. Esta segmentación se produce para encapsular las acciones comunes en el controlador frontal, el cual atiende las peticiones de los usuarios, carga la configuración de la aplicación y maneja la seguridad a nivel de toda la aplicación. En el caso de las acciones, incluyen el código específico del controlador para cada módulo. Todas las peticiones web son manejadas por el controlador frontal, el cual es el único punto de entrada que tiene el sistema, posibilitando que se gestione la seguridad y la configuración a nivel central de aplicación.

2.4 CONCLUSIONES

En este capítulo se presentaron los requisitos del sistema, los cuales influyeron en gran medida en la selección del patrón arquitectónico MVC. Este último contribuye a que el sistema presente una alta independencia entre sus elementos, haciendo posible el soporte a múltiples vistas.

Además se analizaron y definieron las principales tecnologías y herramientas informáticas para dar soporte al desarrollo del SIGM. Entre estas se incluyen a Apache 2.0 como servidor web, MySQL 5 como Sistema Gestor de Base de Datos y PHP 5 como lenguaje de programación, con las que se da cumplimento a las normas establecidas por el Grupo de Arquitectura MISAP- MIC. Otras herramientas propuestas a utilizar en el desarrollo del sistema fueron:

- Sistema de control de versiones Subversion 1.4
- Herramienta CASE Visual Paradigm for UML Community Edition 3.1
- Entorno de desarrollo Eclipse PDT sobre Eclipse Plataform 3.3.1.
- Framework Symfony 1.0
- Gestor de errores Trac 0.10.4

Con las mismas se contribuye a mantener un entorno de desarrollo confiable, rápido y seguro.

CAPÍTULO 3: DESCRIPCIÓN DE LA ARQUITECTURA

La arquitectura de software muestra la visión común del sistema completo en la que el equipo de proyecto y los usuarios deben estar de acuerdo, por lo que describe los elementos del modelo que son más importantes para su construcción. Esto es posible mediante la presentación de varias vistas que se centran en aspectos concretos y describen las principales partes del sistema.
En el presente capítulo se toman en cuenta los elementos que define RUP para el desarrollo de las vistas de la arquitectura, las que son descritas detalladamente con el objetivo de elevar el nivel de comprensión de los desarrolladores.

3.1 REPRESENTACIÓN ARQUITECTÓNICA

RUP propone las siguientes vistas arquitectónicas para la descripción del sistema:

- Vista de casos de uso: “Muestra los casos de uso arquitectónicamente significativos del modelo de caso de uso del sistema” (26). RUP cataloga esta vista como obligatoria en el proceso de desarrollo.
- Vista lógica: “Muestra el subconjunto arquitectónicamente significativo del modelo de diseño” (26). Agrupa las clases más importantes del sistema y su organización en paquetes y subsistemas, proporcionando una base para entender la estructura y organización del diseño del sistema. RUP define a esta vista como obligatoria en el proceso de desarrollo.
- Vista de procesos: Describe los procesos concurrentes del sistema. “Ilustra la descomposición del sistema en procesos, incluyendo las clases y subsistemas activos en cada proceso e hilo de ejecución” (26). RUP considera opcional la representación de esta vista.
- Vista de despliegue: “Modela la configuración física de los nodos” (26) sobre la cual se ejecutará el software. RUP define esta vista como opcional.
- Vista de implementación: Representa los componentes y subsistemas principales del sistema y sus dependencias. “Su objetivo es capturar las decisiones arquitectónicas adoptadas para la implementación” (26). RUP considera esta vista opcional.

Estas vistas se desarrollan utilizando el Lenguaje Unificado de Modelado (UML), que aunque no se considera propiamente un ADL, es una notación que mediante un conjunto definido de elementos y formas de representación, permite de igual manera establecer la descripción de una arquitectura de software .

3.2 VISTA DE CASOS DE USO

La vista de casos de uso es la representación de los casos de uso (CU) arquitectónicamente significativos, aquellos que engloban las funcionalidades principales y críticas del sistema, o que implementan algún requisito importante que debe desarrollarse pronto dentro del ciclo de vida del producto.

Los actores del sistema involucrados en los casos de uso arquitectónicamente significativos se muestran la siguiente tabla:

Actor	Descripción
Genetista	Representa a los usuarios del sistema, o sea todos los genetistas que se encuentran dentro de los niveles de unidad de salud, municipal, provincial y nacional.
Genetista de Unidad de Salud	Representa a los genetistas del centro de genética médica donde el paciente recibe atención médica.
Genetista Municipal	Representa al genetista con permisos a nivel municipal. Tiene acceso a insertar los datos complementarios, modificar los datos complementarios y obtener los reportes de los datos de los pacientes pertenecientes a un municipio.
Genetista Municipal / Unidad de Salud	Es una generalización del Genetista de Unidad de Salud y del Genetista Municipal.
Defectólogo	Genetista especialista en Defectología encargado de introducir y modificar los datos del paciente con discapacidad.

Administrador (Teleconsulta)	Genetista encargado de la gestión de la teleconsulta.
Moderador (Teleconsulta)	Genetista encargado de iniciar la discusión de un caso, de definir el estado del caso discutido, de guardar el historial y de aprobar el informe de la teleconsulta
Solicitante (Teleconsulta)	Genetista que solicita la discusión de un caso en teleconsulta.
Registro del Ciudadano	Componente de SISalud que le proporciona al sistema información primaria de los pacientes.
Registro de Unidades de Salud	Componente de SISalud que le proporciona al sistema información acerca de las Áreas de Salud del país.
SAAA	Componente de seguridad con el cual debe interactuar el sistema para acceder a cualquier otro componente de SISalud.
Servidor de Correo	Servidor de correo utilizado para la gestión de las teleconsultas.

Tabla 1. Actores del sistema.

La vista de casos de uso del sistema queda constituida como se muestra en la siguiente figura:

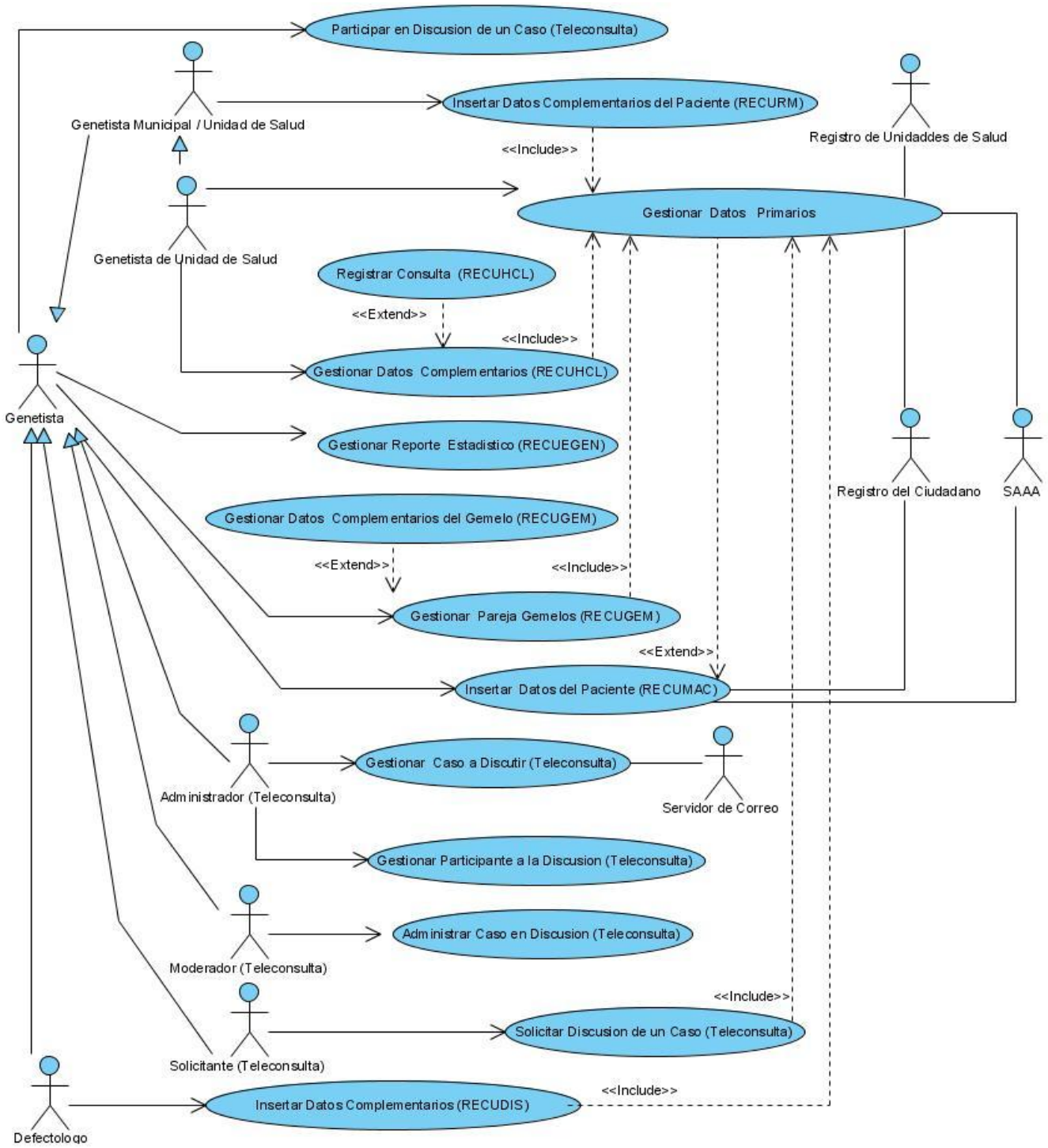


Figura 7. Vista de casos de uso del sistema.

Para detallar la vista de casos de uso del sistema, a continuación se relacionan los casos de uso

arquitectónicamente significativos por cada módulo:

RECUHCL:

- Gestionar Datos Primarios
- Gestionar Datos Complementarios
- Registrar Consulta

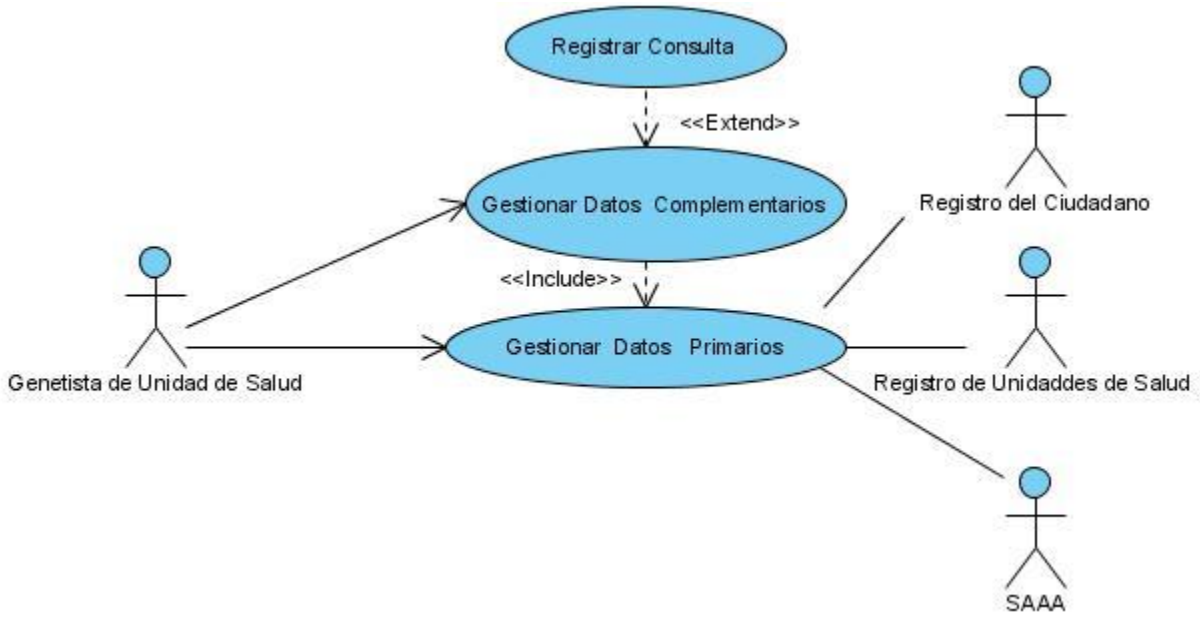


Figura 8. Diagrama de CU Críticos. RECUHCL

RECUDIS:

- Insertar Datos Complementarios

Figura 9. Diagrama de CU Críticos. RECUDIS

RECUEGEN:

- Gestionar Reporte Estadístico

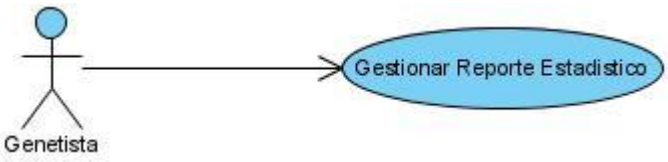


Figura 10. Diagrama de CU Críticos. RECUEGEN

RECUGEM:

- Gestionar Pareja de Gemelos
- Gestionar Datos Complementarios del Gemelo

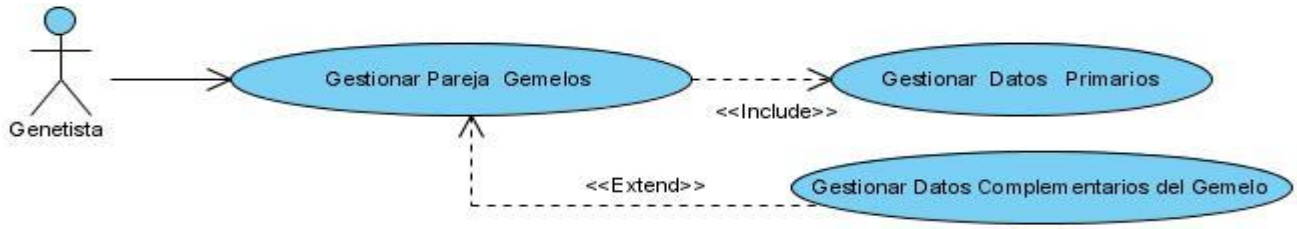


Figura 11. Diagrama de CU Críticos. RECUGEM

RECUMAC:

- Insertar Datos del Paciente

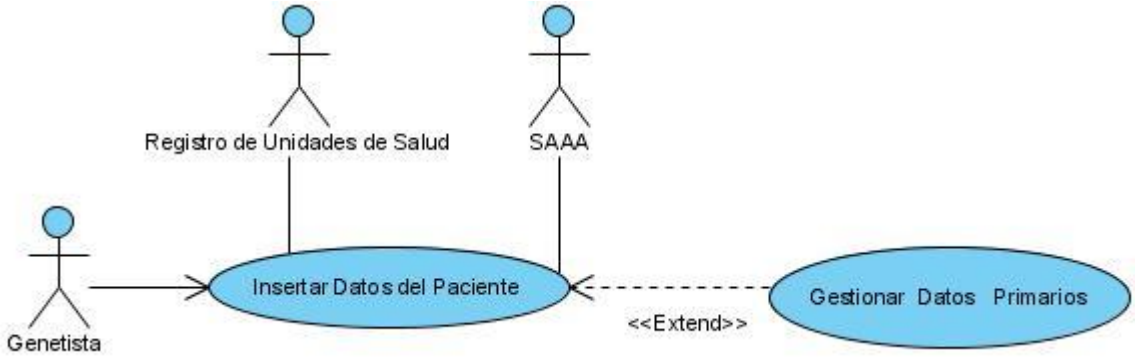


Figura 12. Diagrama de CU Críticos. RECUMAC

RECURM:

- Insertar Datos Complementarios del Paciente

Figura 13. Diagrama de CU Críticos. RECURM

Teleconsulta:

- Solicitar Discusión de un Caso
- Gestionar Caso a Discutir
- Participar en Discusión de un Caso
- Administrar Caso en Discusión
- Gestionar Participante a la Discusión

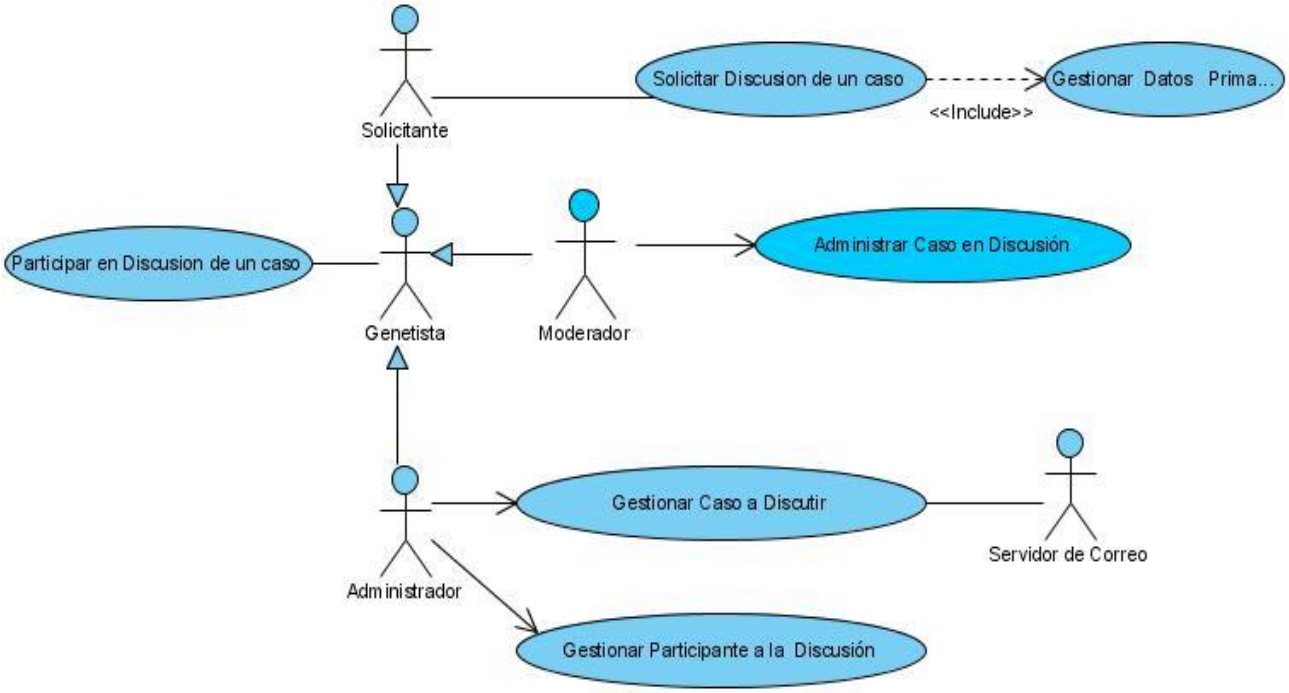


Figura 14. Diagrama de CU Críticos. Módulo Teleconsulta

3.2.1 BREVE DESCRIPCIÓN DE LOS CU ARQUITECTÓNICAMENTE SIGNIFICATIVOS

Un caso de uso encapsula una funcionalidad del sistema, y se basa en una secuencia de interacciones que se desarrollarán entre el sistema y sus actores. Describirlo brevemente consiste en proporcionar de forma textual y mediante un lenguaje natural, dicha secuencia de eventos y acciones, especificando las precondiciones y poscondiciones de ese flujo de eventos. A continuación se describen los casos de uso arquitectónicamente significativos.

3.2.1.1 GESTIONAR DATOS PRIMARIOS. RECUCHL

El caso de uso inicia cuando el actor Genetista de Unidad de Salud accede a la opción Datos Primarios, y el sistema muestra una interfaz de búsqueda para insertar los datos necesarios para encontrar a un paciente. El sistema interactúa con el actor Registro de Ciudadanos, Registro de Unidades de Salud y SAAA para obtener algunos datos necesarios. Si la búsqueda es satisfactoria el sistema muestra una interfaz con los resultados de la búsqueda, el genetista selecciona el paciente para luego modificar sus datos primarios a través de una interfaz. En caso de que el sistema no encuentre al paciente, se mostrará una interfaz para introducir los datos personales del mismo. El sistema valida e introduce los datos en cualquiera de los dos escenarios finalizando así el caso de uso. Si el sistema encontrara algún error en los datos insertados mostrará un mensaje de error.

Precondiciones: El Genetista de unidad de salud debe haberse autenticado y presentar los permisos correspondientes para realizar la acción.

Poscondiciones: Quedan registrados los datos primarios del paciente.

3.2.1.2 GESTIONAR DATOS COMPLEMENTARIOS. RECUCHL

El caso de uso inicia cuando el actor Genetista de unidad de salud selecciona la opción insertar historia clínica y el sistema llama al caso de uso incluido “Gestionar Datos Primarios” para buscar al paciente siguiendo determinados criterios. Después que el genetista selecciona el paciente del listado que se muestra, el sistema abre una interfaz para insertar o modificar los datos genéticos del mismo. Una vez llenos los campos de recogida de datos, el genetista selecciona la opción insertar. Después de aceptar el mensaje de confirmación de envío de datos mostrado por el sistema, se validan e insertan los datos finalizando así el caso de uso. En caso de que existan datos erróneos el sistema

mostrará un mensaje de alerta.

Precondiciones: El genetista de unidad de salud debe haberse autenticado y presentar los permisos correspondientes para realizar la acción.

Poscondiciones: Se registran los datos genéticos de la Historia Clínica.

3.2.1.3 REGISTRAR CONSULTA. RECUCHL

El caso de uso inicia cuando el genetista de unidad de salud ha iniciado el caso de uso Gestionar Datos Complementarios y ha aceptado la opción de insertar los datos de la consulta. El sistema muestra entonces una interfaz que permite añadir una nueva consulta. El genetista llena los datos pertinentes y acepta el envío de los mismos. El sistema inserta los datos después de haberlos validado, finalizando así el caso de uso. En el caso de que el genetista de unidad de salud introduzca algún valor no admitido, el sistema mostrará un mensaje de alerta informando el tipo de error.

Precondiciones: El genetista debe haberse autenticado y presentar los permisos correspondientes para realizar la acción.
Se debe haber ejecutado el caso de uso Gestionar Datos Complementarios.

Poscondiciones: Queda registrada la consulta realizada al paciente.

3.2.1.4 INSERTAR DATOS COMPLEMENTARIOS. RECUDIS

El caso de uso inicia cuando el defectólogo selecciona la opción de insertar datos complementarios de un paciente con discapacidad y el sistema llama al caso de uso incluido “Gestionar Datos Primarios” para buscar al paciente siguiendo determinados criterios. Luego el defectólogo selecciona al paciente con discapacidad al que se le van a introducir los datos complementarios y el sistema muestra una interfaz para la inserción de los mismos. El defectólogo introduce los datos complementarios que son verificados por el sistema para evitar campos obligatorios vacíos, el sistema inserta los datos y muestra un mensaje comunicándole al defectólogo que los datos han sido insertados correctamente, finalizando así el caso de uso. En caso de que existan campos obligatorios vacíos o datos incorrectos, el sistema muestra un mensaje de error.

Precondiciones: El defectólogo debe haberse autenticado y presentar los permisos correspondientes

para realizar la acción.

Poscondiciones: Quedan registrados los datos del paciente con discapacidad.

3.2.1.5 GESTIONAR REPORTE ESTADÍSTICO. RECUEGEN

El caso de uso se inicia cuando el genetista entra al sistema y selecciona la opción gestionar estadística. El sistema muestra una interfaz donde el genetista debe escoger el estadígrafo que va a calcular, entre los que se encuentran: Tasa de Incidencia, Tasa de Incidencia Acumulada, Tasa de Prevalencia al Nacimiento, Tasa de Prevalencia, Tasa de Letalidad, Tasa de Morbilidad, y Tasa de Mortalidad, además de los parámetro por los que se va a regir que puede ser el tiempo, edad, género, enfermedad, municipio o provincia. El genetista establece todas sus opciones y el sistema realiza los cálculos necesarios y muestra la respuesta en formato gráfico finalizando así el caso de uso.

Precondiciones: El genetista debe haberse autenticado y presentar los permisos correspondientes para realizar la acción.

Poscondiciones: El sistema está listo para realizar otra estadística.

3.2.1.6 GESTIONAR PAREJA DE GEMELOS. RECUGEM

El caso de uso inicia cuando el genetista selecciona la opción registrar o actualizar pareja de gemelos y el sistema llama al caso de uso incluido “Gestionar Datos Primarios” para buscar al paciente siguiendo determinados criterios. Luego el genetista selecciona el paciente gemelo perteneciente a la pareja sobre la cual se va a realizar la acción. Si el genetista seleccionó la opción actualizar pareja y juega el rol de visualizador, el sistema muestra el instrumento sin permitir modificar los datos y finaliza el caso de uso. En caso de que el genetista juegue otro rol el sistema muestra el formulario correspondiente para registrar o actualizar la pareja de gemelos según sea la opción que haya seleccionado al comienzo del caso de uso. Los datos nuevos o modificados son validados por el sistema, e insertados. El genetista entonces puede seleccionar la opción que muestra el sistema para gestionar los datos complementarios de los gemelos que componen la pareja, pasando así al caso de uso extendido Gestionar Datos Complementarios del Gemelo. En otro caso el genetista culmina la acción, finalizando así el caso de uso. En todo momento de entrada de datos, el sistema notifica un problema si existieran errores en los mismos.

Precondiciones: El genetista debe haberse autenticado y presentar los permisos correspondientes para realizar la acción.

Poscondiciones: El instrumento de la pareja debe quedar actualizado, insertado o visualizado.

3.2.1.7 GESTIONAR DATOS COMPLEMENTARIOS DEL GEMELO. RECUGEM

El caso de uso inicia cuando el genetista ha iniciado el caso de uso Gestionar Pareja de Gemelos y escoge la opción de gestionar los datos complementarios de los gemelos de la pareja, y el sistema muestra el formulario con los datos del gemelo seleccionado. Si el actor desempeña el rol de editor el sistema permite modificar la información presente o llenar los datos por primera vez. Después de verificar los datos, el sistema inserta los mismos, dando fin al caso de uso. En otro caso, solo se visualiza la información y no se permite modificarla, finalizando así el caso de uso. De presentarse datos incorrectos, el sistema los notifica alertando sobre los mismos.

Precondiciones: El genetista debe haberse autenticado y presentar los permisos correspondientes para realizar la acción.

Se debe haber ejecutado el caso de uso Gestionar Datos Complementarios.

Poscondiciones: Los datos complementarios del gemelo deben quedar actualizados, insertados o visualizados.

3.2.1.8 INSERTAR DATOS DEL PACIENTE. RECUMAC

El caso de uso inicia cuando el genetista desea insertar los datos del paciente con malformaciones genéticas.

Si el genetista necesita registrar los datos de un feto con malformaciones genéticas que fue interrumpido en el embarazo, el sistema muestra una interfaz donde el genetista inserta los datos correspondientes, y estos son almacenados después de ser validados, finalizando así el caso de uso.

En otro caso el sistema llama al caso de uso extendido “Gestionar Datos Primarios” para buscar al paciente siguiendo determinados criterios. Luego de que el genetista elige al paciente con malformaciones congénitas para insertar sus datos, selecciona que tipo de planilla va a insertar, llena el formulario del paciente con malformación genética y pasa a insertar el control u otro paciente con

malformación genética según el tipo de estudio seleccionado.

Si el genetista selecciónó la opción insertar los datos del paciente con malformación genética y control simple, o de un gemelo con malformación genética y control su hermano, el sistema muestra el formulario específico para recoger los datos del control, el cual llena el genetista y valida el sistema antes de ser insertarlos, finalizando así el caso de uso.

Si el genetista selecciónó la opción insertar los datos del paciente con malformación genética simple y control un gemelo, el sistema llama al caso de uso extendido “Gestionar Datos Primarios” para buscar el paciente gemelo. El genetista selecciona al gemelo y pasa a insertar los datos en el formulario específico que le muestra el sistema. Después de ser validada la integridad de los datos, el sistema los inserta, finalizando el caso de uso.

Si el genetista selecciónó la opción gemelos con malformación genéticas, el sistema llama al caso de uso extendido “Gestionar Datos Primarios” para buscar el paciente gemelo. El genetista selecciona al hermano gemelo y pasa a insertar los datos del segundo gemelo con malformación genética en la interfaz que le muestra el sistema. Después de ser validada la integridad de los datos, estos son insertados. Luego el sistema brinda la posibilidad de insertar los controles correspondientes a los gemelos con malformación genética. El usuario selecciona que tipo de control va a utilizar y llena los datos, que luego el sistema inserta, finalizando así el caso de uso.

El sistema interactúa con el actor Registro de Unidades de Salud y SAAA para obtener algunos datos necesarios. Si los datos a insertar en cualquier caso presentan error o hay campos obligatorios sin llenar, el sistema muestra un mensaje informando al usuario del problema.

Precondiciones: El genetista debe haberse autenticado y presentar los permisos correspondientes para realizar la acción.

Poscondiciones: Quedan registrados los datos de los pacientes con retraso mental y su control.

3.2.1.9 INSERTAR DATOS COMPLEMENTARIOS DEL PACIENTE. RECURM

El caso de uso se inicia cuando el genetista selecciona la opción de insertar los datos complementarios de un paciente con retraso mental y el sistema llama al caso de uso incluido

"Gestionar Datos Primarios" para buscar al paciente siguiendo determinados criterios. Luego el genetista selecciona al paciente con retraso mental para introducir los datos complementarios y el sistema muestra una interfaz para la inserción de estos. El genetista llena y envía los datos complementarios, los cuales son verificados por el sistema para evitar campos obligatorios vacíos. El sistema inserta los datos y muestra un mensaje comunicándole al genetista que los datos han sido insertados correctamente finalizando así el caso de uso. En caso de que exista error en las validaciones el sistema muestra un mensaje de error informando el problema.

Precondiciones: El genetista debe haberse autenticado y presentar los permisos correspondientes para realizar la acción.

Poscondiciones: Quedan registrados los datos del paciente con retraso mental.

3.2.1.10 ADMINISTRAR CASO EN DISCUSIÓN. TELECONSULTA

El caso de uso se inicia cuando el moderador selecciona el primer caso a discutir planificado para el día, y el sistema muestra el listado de los mismos ordenados en esa fecha por su hora de inicio. El moderador selecciona el primer caso y da inicio al debate, el sistema muestra la sala de discusión. Cuando se desee dar por concluida la discusión, el sistema permite al moderador definir el estado del caso en: pospuesto, remitido o resuelto. Si el moderador selecciona la opción posponer caso, el sistema muestra una interfaz para reprogramar el caso, pide aceptación del moderador y muestra nuevamente la interfaz de discusión. En el caso de que el moderador escoja la opción remitir caso, el sistema cambia su estado a remitido para que sea analizado por el administrador nacional, y es eliminada la sala de discusión que contenía el caso. Si el estado del caso a definir es resuelto, el sistema sencillamente cambia el estado del caso y es eliminada la sala de discusión que lo contenía. Luego de discutido el caso seleccionado, el moderador guarda el historial del caso y genera un informe que contendrá los datos importantes del mismo finalizando así el caso de uso.

Precondiciones: El moderador debe haberse autenticado y presentar los permisos correspondientes para realizar la acción.

Debe existir al menos un caso a discutir planificado para la fecha del sistema.

Poscondiciones: Queda pospuesto, remitido o resuelto un caso.

Se guarda el historial del caso discutido.

3.2.1.11 SOLICITAR DISCUSIÓN DE UN CASO. TELECONSULTA

El caso de uso inicia cuando el solicitante selecciona la opción realizar una nueva solicitud y debe aceptar el código de ética que le muestra el sistema. En este momento el sistema llama al caso de uso incluido “Gestionar Datos Primarios” para buscar al paciente siguiendo determinados criterios. Luego el genetista selecciona el paciente cuyo caso propondrá discutir, y el sistema muestra una interfaz para introducir los datos de la solicitud. Después de verificados los datos y ser almacenados en el sistema finaliza el caso de uso. En caso de que existan errores en los datos el sistema notifica el problema.

Precondiciones: El administrador debe haberse autenticado y presentar los permisos correspondientes para realizar la acción.

Deben existir casos a discutir.

Poscondiciones: Queda almacenada una nueva solicitud.

3.2.1.12 GESTIONAR CASO A DISCUTIR. TELECONSULTA

El caso de uso se inicia cuando el administrador desea ver los casos a discutir y el sistema muestra un listado de casos a discutir planificados y sin planificar, permitiendo planificar un nuevo caso a discutir, modificar o eliminar uno existente. Si el administrador escoge la opción de planificar un nuevo caso, el sistema muestra una interfaz para especificar la fecha y la hora de inicio y fin del caso y pasa a seleccionar los participantes en el caso. El sistema almacena la información y envía un correo, citando a todos los usuarios propuestos finalizando el caso de uso. En caso de que el administrador seleccione la opción de modificar un caso existente, el sistema presentará las interfaces para actualizar los datos del caso y sus participantes. El sistema actualiza la información y envía un correo, citando a todos los usuarios propuestos, finalizando así el caso de uso. Si el administrador selecciona eliminar un caso existente, el sistema presenta un mensaje de confirmación. Después de aceptado este último, el sistema pone el caso en estado cancelado y notifica a los participantes, finalizando así el caso de uso.

Precondiciones: El administrador debe haberse autenticado y presentar los permisos correspondientes para realizar la acción.

Deben existir casos a discutir.

Poscondiciones: Queda planificado, modificado o eliminado un caso a discutir.

3.2.1.13 PARTICIPAR EN DISCUSIÓN DE UN CASO. TELECONSULTA

El caso de uso se inicia cuando el genetista desea entrar a la discusión de un caso y selecciona la opción de ver las salas de conferencia listas para esa fecha. El sistema muestra el listado de las mismas permitiendo al genetista seleccionar en cual quiere participar. Si la sala seleccionada ya está habilitada, el sistema muestra una página que contiene el caso que se está discutiendo en ese momento, y la sala de conversación. Una vez terminado el debate finaliza el caso de uso para los genetistas participantes que no coincidan con el solicitante de la discusión. En otro caso el sistema mostrará una interfaz para insertar un resumen de la discusión, las recomendaciones y el diagnóstico del caso, finalizando el caso de uso con el almacenamiento de los datos. Si la sala de discusión seleccionada no está habilitada el sistema muestra un mensaje informando que aún no se ha iniciado la discusión, finalizando el caso de uso.

Precondiciones: El genetista debe haberse autenticado y presentar los permisos correspondientes para realizar la acción.

Debe existir al menos un caso a discutir para la fecha.

El usuario debe estar entre los propuestos a participantes del caso a discutir.

Poscondiciones: Queda inicializada la discusión del caso a discutir.

3.2.1.14 GESTIONAR PARTICIPANTE A LA DISCUSIÓN. TELECONSULTA

El caso de uso se inicia cuando el administrador necesita adicionar o eliminar un participante a la discusión de un caso en determinada sala. Si escoge la opción de adicionarlo, el sistema muestra una pantalla de criterios de búsqueda para seleccionar los posibles participantes a la discusión. Una vez seleccionados, el sistema guarda los datos y actualiza el listado de los participantes, finalizando así el caso de uso. Si el administrador desea eliminar a un participante de la discusión, el sistema permite seleccionar el o los posibles participantes a eliminar. Dados los participantes seleccionados para

eliminar, el sistema muestra un aviso de confirmación, que el usuario aprobará y se dará por terminado el caso de uso.

Precondiciones: El genetista debe haberse autenticado y presentar los permisos correspondientes para realizar la acción.

Deben existir al menos una sala planificada.

Poscondiciones: Queda adicionado o eliminado un participante a la discusión de un caso.

3.3 VISTA LÓGICA

La vista lógica describe el sistema mostrando las clases más importantes y su organización en paquetes y subsistemas. Esta vista permite observar cómo está diseñada la funcionalidad en el interior del sistema.

Symfony define un diseño, conceptos, componentes y herramientas de administración que se reutilizarán en el desarrollo de sistema. Establece una estructura del proyecto en aplicaciones, y cada aplicación está compuesta por módulos que representa a un grupo de páginas con un propósito relacionado. Como se explicó en el Capítulo 2, Symfony implementa el patrón MVC seleccionado para el sistema, por cuanto la estructura organizacional del SIGM se representa a través de tres elementos fundamentales: el controlador, el modelo y la vista. En la siguiente figura se muestra la vista lógica del sistema.

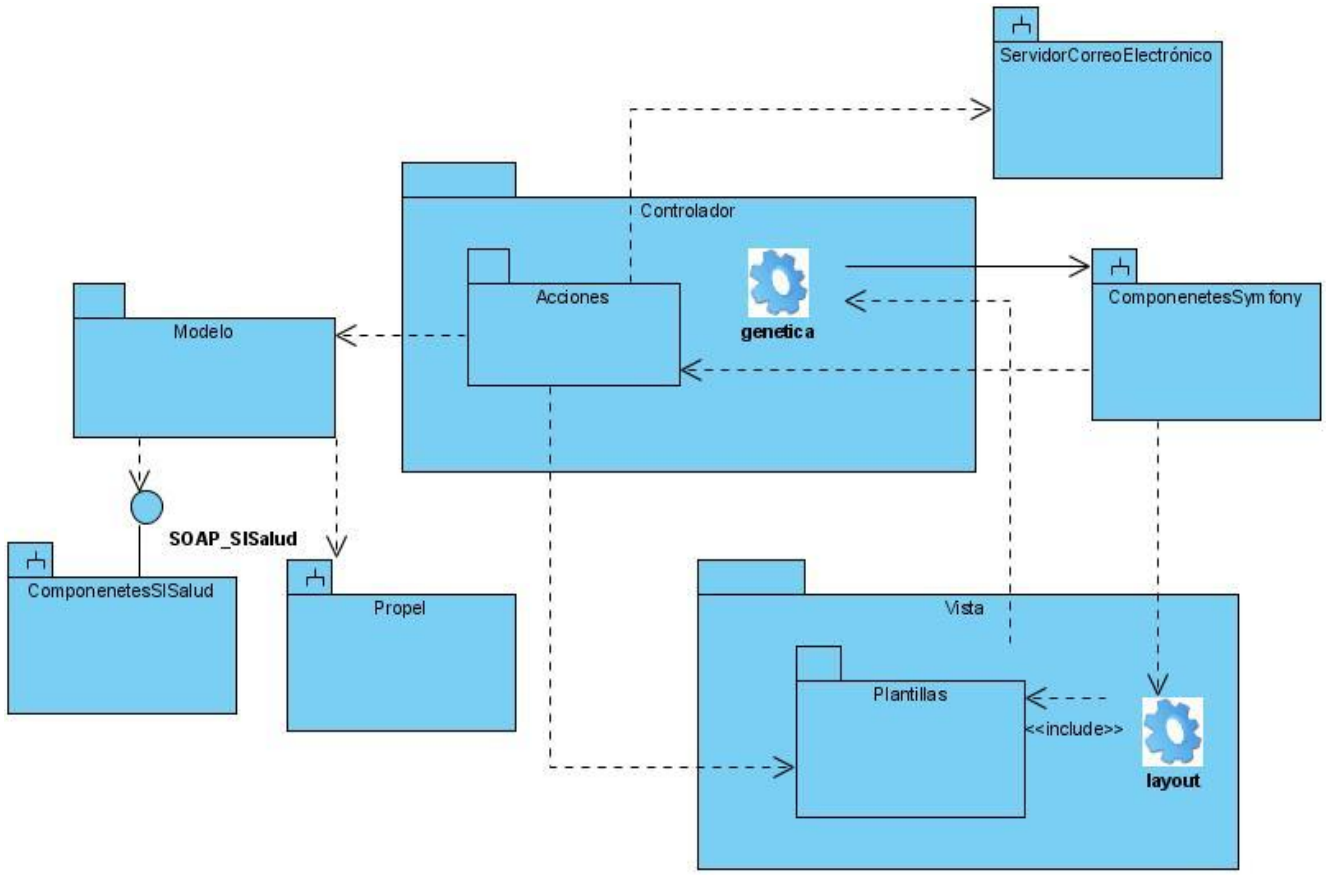


Figura 15. Vista lógica de primer nivel

A continuación se expone la descripción de los paquetes y subsistemas del diseño que conforman la vista lógica.

Paquete Controlador: Agrupa las clases que implementan la lógica de la aplicación.

La clase controlador frontal, ***genetica*** encapsula las tareas comunes de la lógica de cada módulo. La primera configuración de la aplicación se encuentra en este controlador frontal que contiene la definición de las constantes principales, carga el archivo general de configuración y despacha la petición.

Paquete Acciones: Encierra la clase ***Actions*** de cada módulo. Estas clases definen las acciones que incluyen el código específico del controlador de cada página.

Paquete vista: Incluye las clases necesarias para presentar al usuario la lógica de la aplicación.

La clase ***layout*** contiene los elementos que se muestran de forma idéntica para las páginas web a lo largo de toda la aplicación.

Paquete Plantillas: Agrupa todas las plantillas o páginas de cada módulo que se encargan de visualizar las variables definidas en el paquete del controlador.

Paquete del modelo: Contiene las clases que encapsulan la lógica del dominio, y se encargan del acceso a los datos almacenados en el gestor de base de datos.

Subsistema Propel: Es un ORM cuya función es gestionar el modelo del sistema. Implica que el acceso y la modificación de los datos almacenados en la base de datos se realicen mediante objetos, nunca de forma explícita, permitiendo un alto nivel de abstracción y fácil portabilidad.

Subsistema ComponentesSymfony: Represan todas las clases del framework Symfony que serán utilizadas durante el funcionamiento del sistema, tales como validadores de formularios, sistema de enrutamiento, componentes de seguridad y configuración, entre otros.

Subsistema ComponentesSISalud: Representa los componentes de SISalud que exportan los servicios web (representados a través de ***SOAP_SISalud)***, que consumirá el sistema.

Subsistema ServidorCorreoElectrónico: Representa al servidor de correo electrónico con el cual se comunica el sistema al enviar un correo.

Todas las peticiones web son manejadas por un solo controlador frontal, que es el punto de entrada único de toda la aplicación. Cuando el controlador frontal recibe una petición, inicializa las clases y constantes del núcleo del framework, carga la configuración, utiliza el sistema de enrutamiento para asociar el nombre de una acción y el nombre de un módulo con la URL escrita, ejecuta la acción y produce la vista, mostrando finalmente la respuesta.

Las acciones contienen la lógica de la aplicación, verifican la integridad de las peticiones, utilizan el modelo y definen variables para la vista. El valor retornado por la acción determina como será producida la vista para especificar la plantilla que se utiliza al mostrar el resultado de la acción. El

contenido de la plantilla se incluye en el layout, visualizando finalmente el resultado de la petición al usuario.

El modelo es el encargo de obtener los datos del dominio a través del acceso directo a la base de datos del sistema o consumiendo los servicios web que brindan los componentes de SISalud. Las clases del modelo obtienen los datos de la base de datos utilizando el ORM Propel que viene integrado con el framework.

Para mayor detalle, se presentan a continuación un grupo de vistas específicas que describen el contenido de los paquetes, y reflejan algunas relaciones que no se evidencian en la vista lógica general.

Paquete Controlador

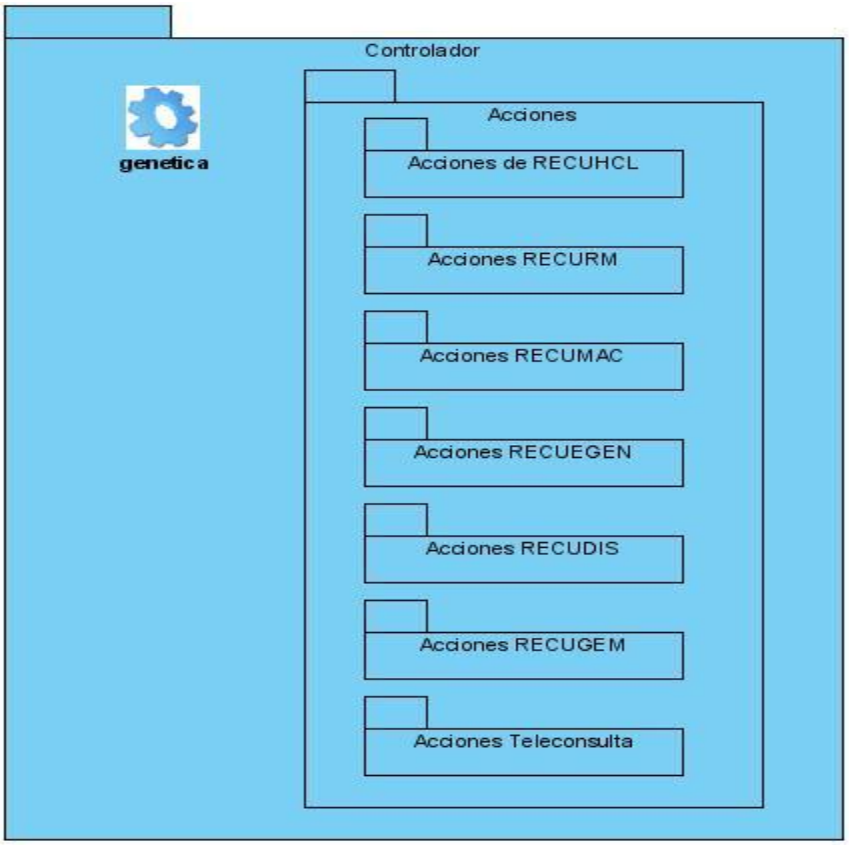


Figura 16. Vista lógica de segundo nivel. Paquete Controlador

El **Paquete Acciones** agrupa las acciones de los módulos en cada paquete *Acciones_NombreMódulo* correspondiente, los cuales contienen la clase *NombreMóduloActions* que define las acciones propias del módulo.

Paquete Vista

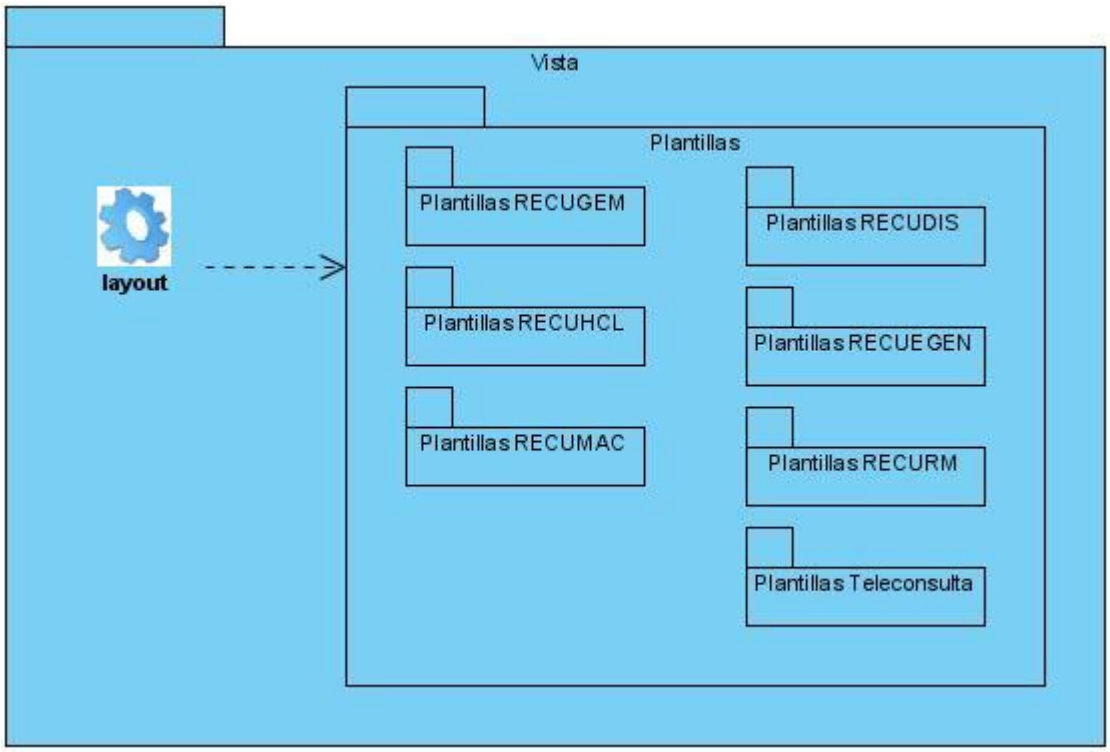


Figura 17. Vista lógica de segundo nivel. Paquete Vista

El **Paquete Plantillas** agrupa las plantillas o páginas de cada módulo en el paquete *Plantilla_NombreMódulo* correspondiente.

Paquete Modelo

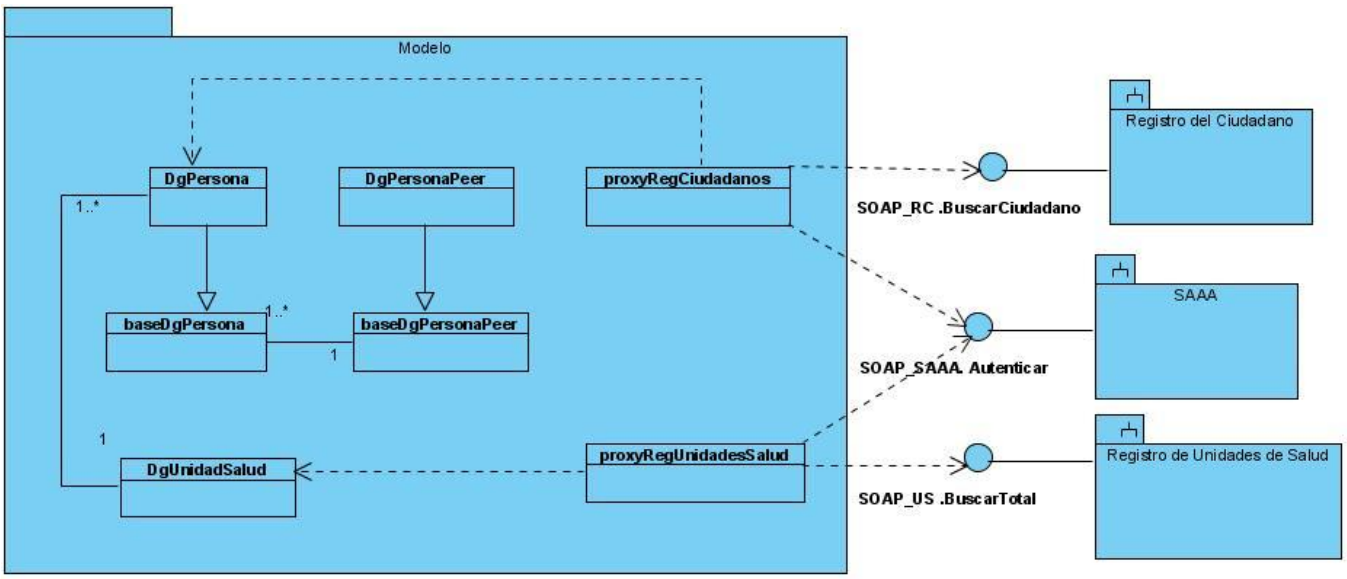


Figura 18. Vista lógica de segundo nivel. Paquete Modelo

Symfony automatiza la creación del modelo utilizando la descripción del esquema relacional de la base de datos. Por cada tabla del esquema relacional crea cuatro clases fundamentales: Base*NombreTabla*, Base*NombreTabla*Peer, *NombreTabla*, *NombreTabla*Peer. Dos de ellas con nombre Base y abstractas son generadas directamente a partir del esquema y contienen implementado los métodos básicos de acceso a los datos. Nunca se deberían modificar porque cada vez que se genera el modelo se borrarán y crearán nuevamente. Las otras dos clases de objetos propias heredan de las clases con nombre Base correspondiente. Estas clases no se modifican cuando se actualiza el modelo, por lo que son las clases en las que se añaden los métodos propios. Entre estas últimas se distinguen la de nombre terminado en Peer, lo que significa que contiene métodos estáticos para trabajar sobre las tablas de la base de datos, proporcionando los medios necesarios para obtener los registros de las tablas. Sus métodos devuelven normalmente un objeto o una colección de objetos de la clase *NombreTabla* correspondiente. La clase *NombreTabla* es por su parte la encargada de representar un registro de la base de datos.

El **Paquete Modelo** contiene todas las clases necesarias para el acceso a los datos del sistema. Se muestra por ejemplo a la clase ***DgPersona*** y sus tres clases correspondientes, todas generadas por Symfony, que permiten el acceso a la información contenida en la tabla Persona. Además se muestra

la clase ***proxyRegUnidadesSalud*** que permite recuperar los datos brindados por el servicio web ***SOAP_US.BuscarTotal*** expuesto por el Registro de Unidades de Salud, utilizando para ello los servicios del componente de seguridad SAAA. Consumiendo el servicio ***Autenticar***, el sistema se autentifica con la SAAA suministrando un nombre de usuario único y una contraseña. El sistema recibe un certificado digital, que contiene un identificador único de 32 caracteres generado de manera aleatoria (token), el identificador del usuario, y el nivel de acceso del mismo. Para cada petición que se haga a un componente de SISalud, debe enviársele el token, el cual contiene la autorización para recibir el servicio.

El sistema para dar cumplimiento a los requisitos de disponibilidad, realiza en su base de datos una copia de la información que necesita obtener a través de otros componentes de SISalud, de forma tal que si se encuentra desconectado, la clase ***proxyRegUnidadesSalud*** pide los datos de la tabla correspondiente en la base de datos del sistema, en este caso ***DgUnidadSalud***. A pesar de que este mecanismo no es considerado una buena estrategia, ya que incluye gasto de recursos por guardar información redundante, así como responsabilidad distribuida sobre el control de la misma, en la práctica resulta más eficiente pues garantiza la autonomía de la aplicación, abstrayéndose de los problemas de conectividad y respuesta de sistemas dependientes.
En esta vista solo se representó un extracto mínimo de las clases del modelo por cuestiones de espacio, organización y entendimiento.

Para lograr un detalle más completo, se muestra a continuación un tercer nivel de profundización de la vista lógica, en el cual se representan las clases contenidas en cada paquete especificado anteriormente. Esta vista solo incluye las clases involucradas en la realización del caso de uso Gestionar Datos Primarios.

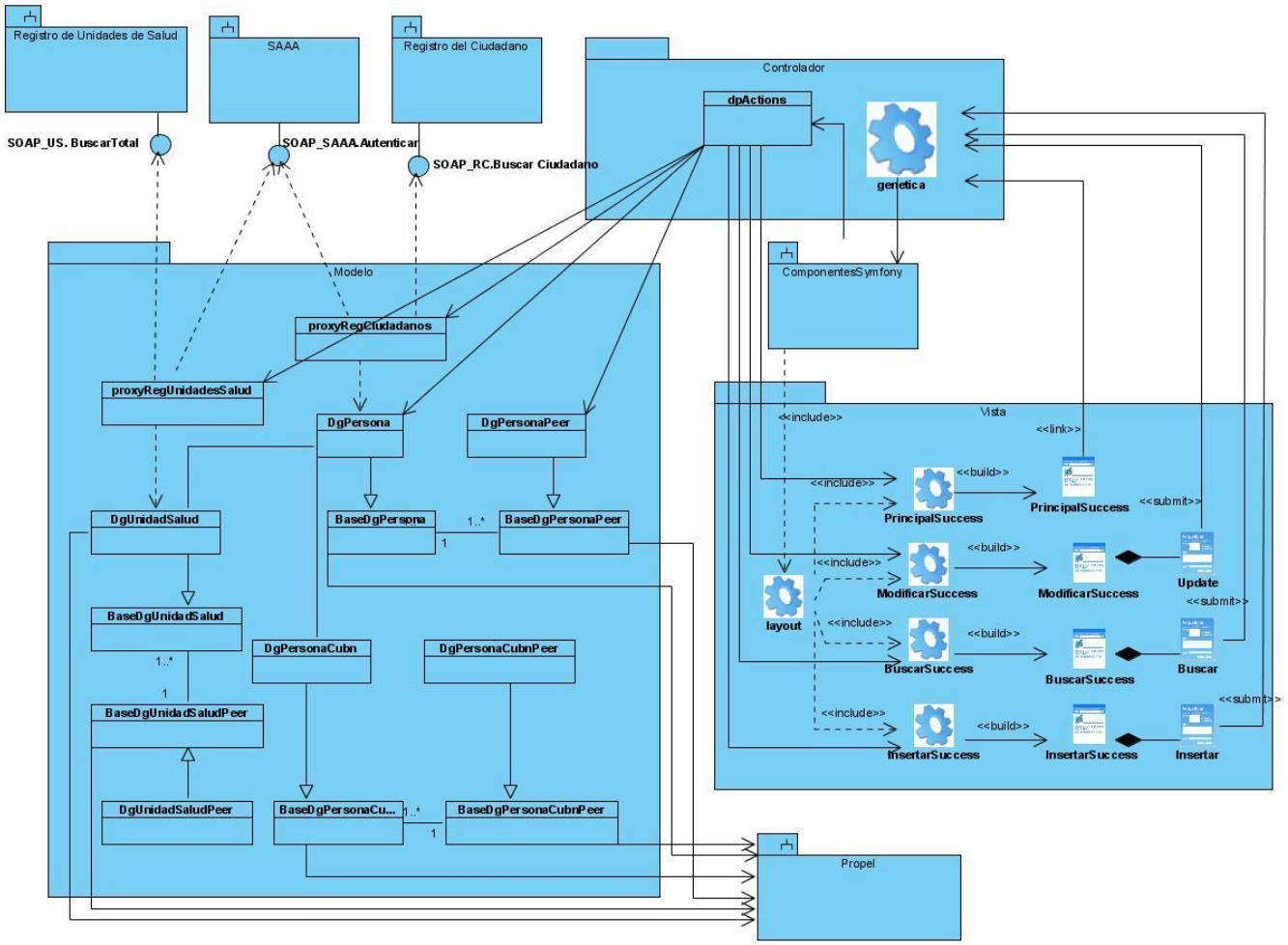


Figura 19. Vista Lógica de tercer nivel. Caso de uso Gestionar Datos Primarios.

3.4 VISTA DE DESPLIEGUE

La vista de despliegue describe la configuración física sobre la que será desplegado el software. Esta presenta los nodos computacionales que intervienen en el funcionamiento del sistema, las conexiones entre estos y los protocolos de comunicación, estableciendo posibles configuraciones que se ilustran mediante los diagramas de despliegue.

Para poner en funcionamiento el sistema, es necesario que se despliegue en tres servidores dedicados en Infomed, corriendo sobre el sistema operativo Linux, distribución Debian (Sarge). Estos servidores presentan la siguiente estructura:

• Un servidor denominado Presentación que contiene la lógica de la presentación y se conecta a la red virtual privada de Infomed. Constituye el punto de entrada del usuario al sistema, pues es el nodo con que el cliente se conecta directamente.

• Un servidor llamado Aplicación que contiene la lógica de la aplicación, y posee conectividad punto a punto únicamente con el servidor que hospeda la capa de presentación y el servidor de datos.

• Un servidor llamado Datos que contiene los datos, y posee únicamente conectividad punto a punto con el servidor de aplicación.

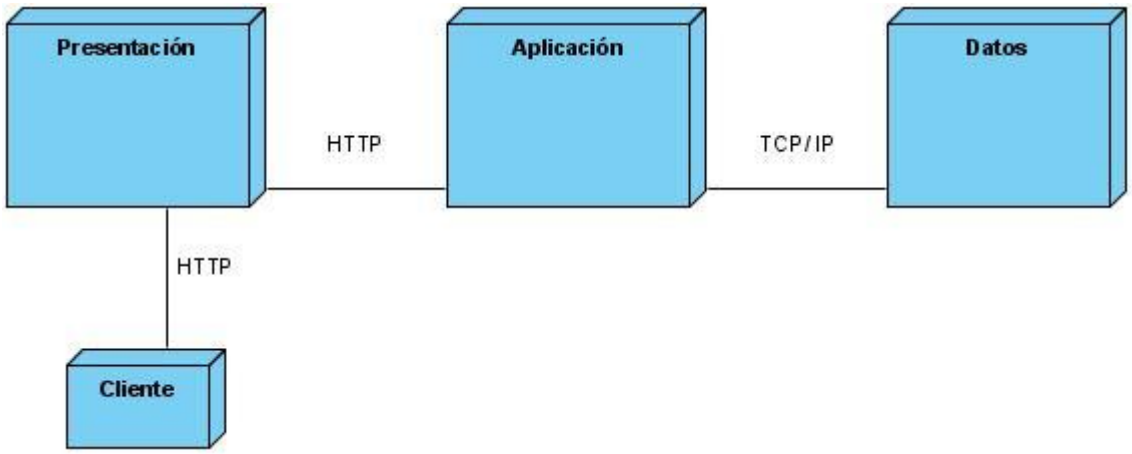


Figura 20. Infomed, estructura básica de servidores.

Esta estructura básica es justificada por Infomed, y aceptada por la arquitectura que se propone para el SIGM, debido a las ventajas de seguridad y rendimiento que posibilita utilizar un despliegue basado en tres servidores. El único punto de acceso del cliente a la aplicación es a través del nodo de Presentación, el servidor de Aplicación se encuentra en un segundo nivel y el de Datos en un tercer nivel de acceso, implementando una barrera física de conexión difícil de violar por usuarios maliciosos. Además todos los procesos que lleva a cabo el sistema se dividen en tres servidores, aumentando el rendimiento.

Asumiendo la estructura anteriormente planteada, el diagrama de despliegue del sistema quedaría conformado como se muestra en la siguiente figura:

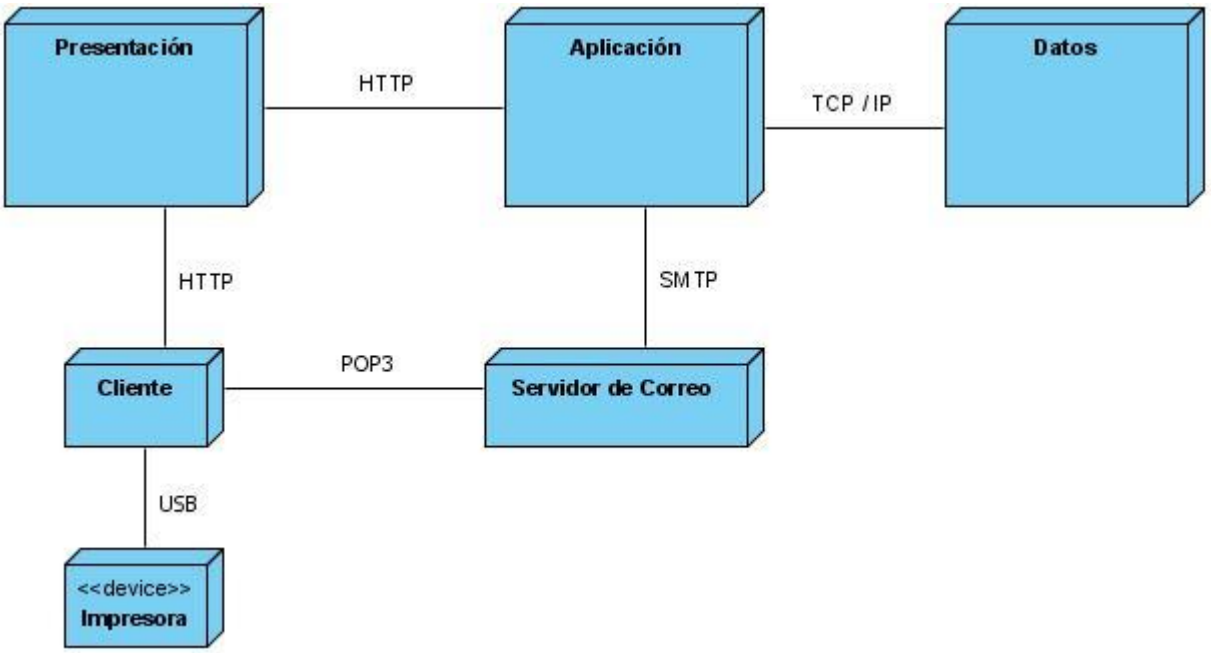


Figura 21. Vista del despliegue. Propuesta básica.

El sistema necesita un servidor de correo electrónico, el que se propone desplegar en un nodo independiente, identificado en el diagrama con igual nombre. Además se representa la impresora como dispositivo externo para dar cumplimiento a los requisitos del cliente.

A raíz del análisis sobre estudios que arrojaron una relación promedio de una consulta de actualización por cada nueve de selección, y con el objetivo de optimizar el sistema y garantizar mayor disponibilidad, Infomed decidió implantar una estrategia de replicación de las bases de datos con que operan los sistemas desplegados en sus servidores. La replicación es la copia sincronizada entre dos o más servidores de bases de datos, de forma tal que cualquiera de ellos pueda entregar los mismos resultados a sus clientes. Se basa en un esquema "maestro-esclavos", en el que el maestro mantiene la base de datos original y los esclavos las copias. Para implantar dicho mecanismo, Infomed dispone de un nodo servidor de base de datos maestro, quien atiende las consultas de modificación invocadas por el nodo de Aplicación, y otro nodo servidor de base datos esclavo, que responde ante las solicitudes de selección enviadas por el mismo nodo de Aplicación. La propuesta del Grupo de Arquitectura MINSAP-MIC, consiste en ir incorporando servidores de datos esclavos a medida que

aumenten los niveles de información de las bases de datos y el acceso a los mismos, lo cual posibilitará un mayor balance de carga y por consiguiente un mejor rendimiento.

Para asumir la propuesta anteriormente planteada, los desarrolladores deben ser conscientes de redireccionar las consultas correspondientes a cada servidor, utilizando las variables de conexión especificadas para cada conexión en el archivo de configuración de la base datos de la aplicación, ***$sf_symfony_data_dir/config/databases.yml.*** No se considera en la propuesta la utilización de un proxy para redireccionar las consultas y abstraer al desarrollador sobre este tema, debido a que un proxy de este tipo debe parsear la cadena de la consulta para definir si es de selección o de actualización, haciendo lento al sistema.

Asumiendo los beneficios de un despliegue con réplica de base de datos, la estructura general del sistema sería la siguiente:

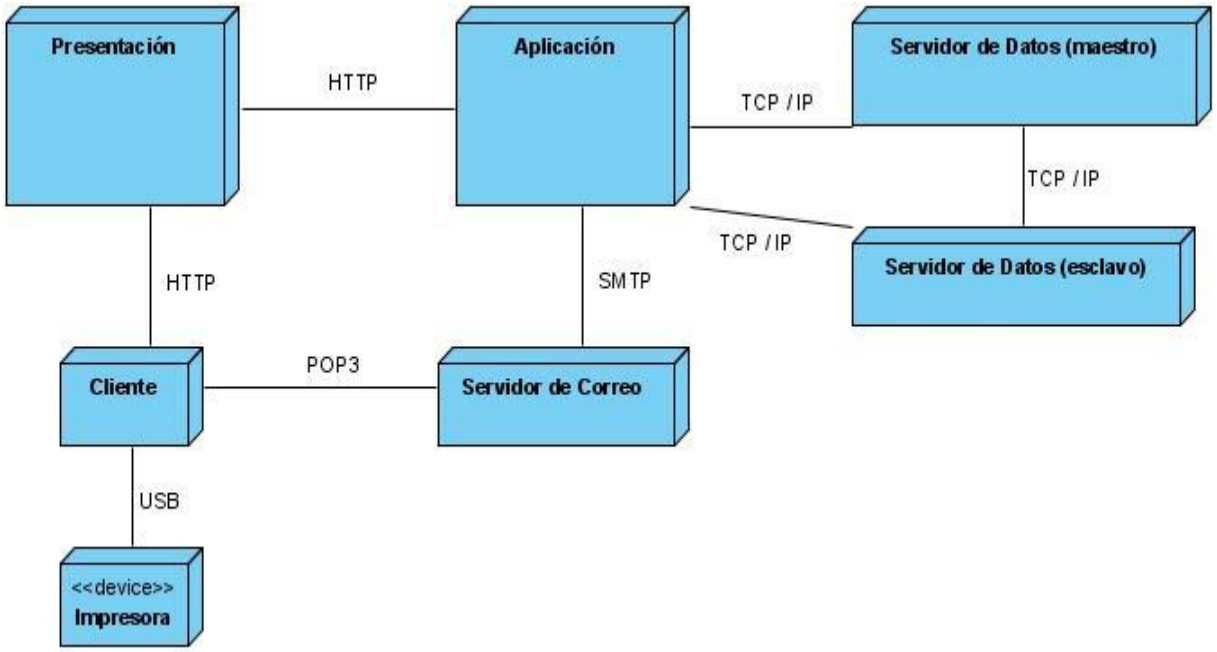


Figura 22. Vista de despliegue del sistema.

CAPÍTULO 3: DESCRIPCIÓN DE LA ARQUITECTURA

3.5 VISTA DE IMPLEMENTACIÓN

La vista de implementación constituye una selección de los aspectos fundamentales del diagrama de componentes del sistema, el cual modela el empaquetado físico del sistema en unidades reutilizables llamadas “componentes” y sus relaciones. Un componente es una unidad física de implementación que encapsula una o más clases del diseño. Estos se pueden agrupar en subsistemas de implementación, para mayor organización y claridad del diagrama. En resumen, la vista de implementación de la arquitectura muestra los elementos físicos reales más significativos del sistema.

Symfony establece una estructura de carpetas para sus proyectos. Dentro de un proyecto, las operaciones se agrupan de forma lógica en aplicaciones. Cada aplicación está formada por uno o más módulos. Entre las carpetas más importantes se encuentran las siguientes:

apps: Contiene un directorio por cada aplicación del proyecto, en este caso genética/ que a su vez contiene la carpeta modules/ que separa los componentes específicos de cada módulo. Además dentro de genética/ se encuentra el subdirectorio templates/ que agrupa las plantillas globales de la aplicación, o sea, las que utilizan todos los módulos.

lib: Guardar el código común a todas las aplicaciones del proyecto. El subdirectorio model/ guarda el modelo de objetos del proyecto. Por su parte symfony/ agrupa las clases que implementan el framework

web: Constituye la raíz web del sistema. Contiene los únicos archivos accesibles desde Internet.

Para ilustrar mejor lo antes descrito se muestra la siguiente figura:

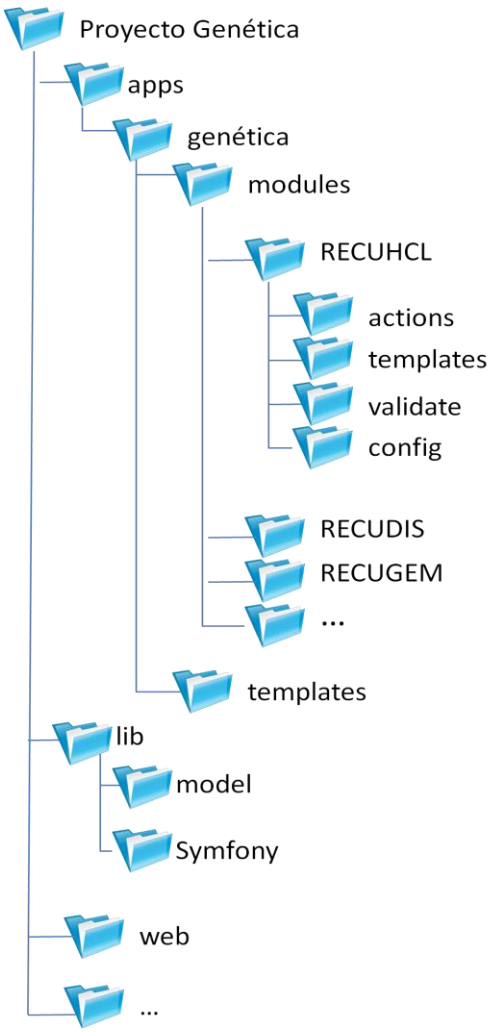


Figura 23. Estructura física del sistema.

Otros subdirectorios agrupan componentes propios de Symfony, que son clases que implementan el núcleo del framework, y por tanto no se modifican en la construccion del sistema.

La vista de implementación del sistema queda constituida por un grupo de componentes, y subsistemas de implementación que modelan el empaquetado fisico real del sistema. Estos subsistemas de implementación respresentan a las carpetas del directorio del proyecto descritas anteriormente, y contienen a los ficheros modelados en término de componentes. A continuación se muestra la vista de los principales elementos del digrama de componentes.

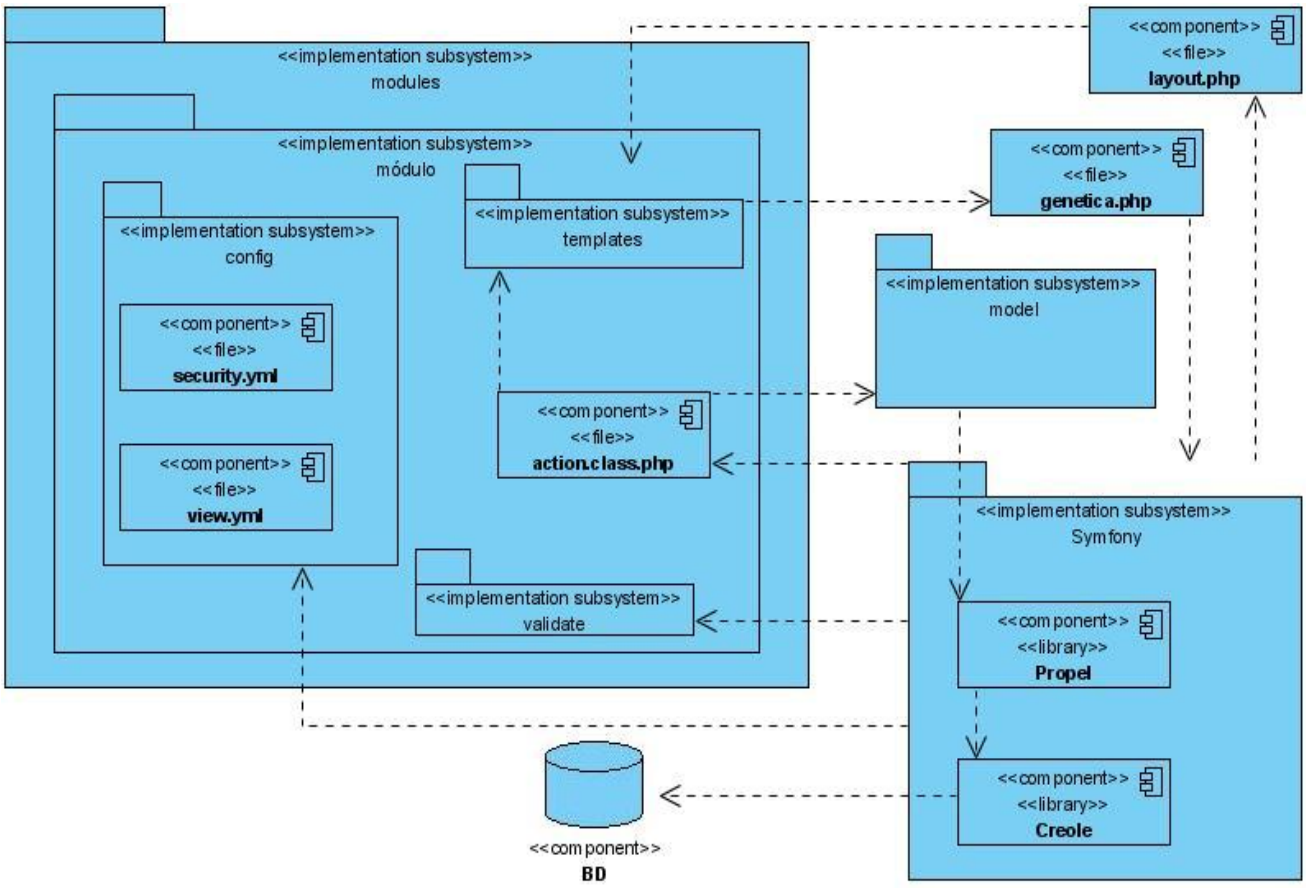


Figura 24. Vista de implementación.

A continuación se expone la descripción de los componentes y subsistemas de implementación que conforman la vista de implementación.

Componente genetica.php: Componente que implementa la clase *genética* del diseño. Se corresponde con el controlador frontal del sistema.

Componente layout.php: Componente que implementa la clase *layout* del diseño. Contiene los elementos que se muestran de forma idéntica a lo largo de toda la aplicación.

Componente BD: Encapsula todos los datos del sistema.

Subsistema de implementación model*:* Agrupa los componentes que definen el modelo. Cada componente de model se corresponde con un archivo php que implementa una clase de diseño del Paquete Modelo del diseño.

Subsistema de implementación modules: Se representa por cuestiones organizativas. Encapsula los diferentes *Subsistemas de implementación módulo* correspondientes a cada módulo del sistema

Subsistemas de implementación módulo: Encapsula los componentes de un módulo.

El componente ***actions.class.php*** implementa la clase *Actions* del diseño. Define las acciones que incluyen el código específico del controlador para cada página del módulo.

Subsistema de implementación templates: Agrupa los componentes que implementan el código correspondiente a cada plantilla que utiliza el módulo. Se corresponde con la Carpeta Plantillas del diseño.

Subsistema de implementación validate: Contiene los componentes de tipo archivos de texto en formato YML que especifican las reglas de validación de datos.

Subsistema de implementación config: Encapsula los componentes de tipo archivos de texto en formato YML que definen la configuración específica de cada módulo.

Componente security.yml: archivo de configuración que permite restringir el acceso a determinadas acciones del módulo.

Componente view.yml: archivo de configuración de las vistas de cada módulo.

Subsistemas de implementación Symfony: Encapsula todos los componentes propios del framework. Se especifica el **componente Propel,** que es el ORM que utiliza Symfony, Propel utiliza el **componente Creole** como capa de abstracción a base de datos

3.5.1 INSTALANDO LA APLICACIÓN.

El proceso de despliegue consiste en transferir la aplicación funcional desde el entorno de desarrollo hasta entorno de producción.

Para poder desplegar el directorio raíz del proyecto en servidores de producción dedicados como se explicó en el epígrafe 3.6, es necesario distribuir la estructura básica de directorios. Cada vez que el framework requiere buscar una clase, plantilla o archivo de configuración utiliza una variable que almacena la ruta. Modificando el valor de estas variables almacenadas en el archivo de configuración ***$sf_symfony_data_dir/config/constants.php***, se puede modificar por completo la estructura de directorios del proyecto para adaptarla a las necesidades específicas del despliegue, permitiendo establecer la distribución de componentes mostrada en la siguiente figura.

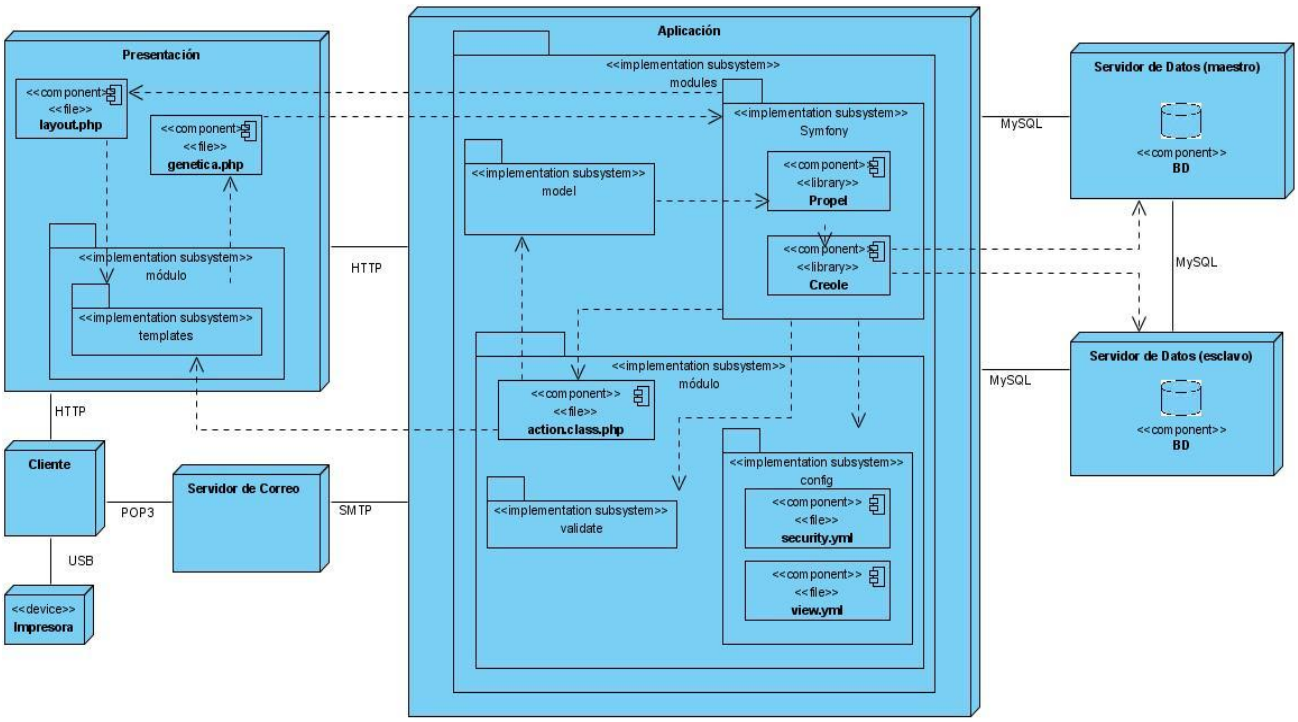


Figura 25. Despliegue de los componentes.

Esta distribución de componentes sigue el esquema presentado en el diagrama de despliegue (Ver Acápite 3.4), donde el servidor denominado Presentación contiene la lógica de la presentación y se

conecta con el cliente a través de la red virtual privada de Infomed, constituyendo el punto de entrada del usuario al sistema. Debido a sus características, en el mismo se deben desplegar las vistas generales de la aplicación, así como las de los módulos, al igual que el controlador frontal que es el punto de acceso a toda la aplicación. El servidor llamado Aplicación contiene la lógica de la aplicación, por lo que en el se encontrarán desplegados los componentes de Symfony que representan el núcleo del framework; el modelo, que contiene la lógica de acceso a los datos; así como la lógica propia de cada módulo contenida en las acciones y sus ficheros de configuración. Los servidores llamados Datos mantiene la base de datos del sistema.

3.6 CONCLUSIONES

Las vistas arquitectónicas muestran la propuesta de solución del sistema desde diferentes perspectivas, proporcionando a los desarrolladores una visión común del mismo.

En el presente capítulo se seleccionaron los casos de uso arquitectónicamente significativos que conforman la vista de casos de uso del sistema, agrupando las principales funcionalidades que este debe cumplir. El sistema presenta catorce funcionalidades significativas, las cuales son inicializadas por un actor genérico llamado genetista. *Gestionar Datos Primarios* es la funcionalidad que más se ejecuta, puesto que tiene el objetivo de insertar, modificar y buscar los datos básicos de un paciente.

Al desarrollar la vista lógica se tuvieron en cuenta las clases, paquetes y subsistemas de diseño más significativos para el sistema, permitiendo observar la forma en están diseñadas las funcionalidades en el interior del mismo. Además se identificaron los componentes y subsistemas de implementación en la vista de implementación, modelando el empaquetado fisico real del SIGM. En la vista de despliegue se representaron los nodos y conexiones que intervienen en el funcionamiento del sistema, mostrando la configuración física sobre la que será desplegado el software.

La vista lógica y la de implementación se diseñaron incorporando la estructura básica definida por el framework de desarrollo Symfony, que a pesar de contar con una amplia documentación, se tornó complejo modelar las relaciones entre los elementos del sistema debido al grado de abstracción que propone el mismo.

CAPÍTULO 4: EVALUACIÓN DE LA ARQUITECTURA PROPUESTA

CAPÍTULO 4: EVALUACIÓN DE LA ARQUITECTURA PROPUESTA

Una arquitectura de software no solo define la estructura que tendrá el sistema, sino que determina en gran medida el éxito del proceso de desarrollo llevado a cabo, y verifica que se cumplan ciertos atributos de calidad de gran importancia como son: la portabilidad, la usabilidad, la confiabilidad, entre otros.

Seleccionar una arquitectura indebida y comprobarlo al terminar el producto traería consecuencias desastrosas, tales como tener fallas en el sistema o que la aplicación no satisfaga los requerimientos establecidos por el cliente. La solución a este problema es la evaluación temprana de la misma, que permite comprobar si la arquitectura propuesta es la más factible según las necesidades del sistema.

En el presente capítulo se explican los aspectos más relevantes sobre los atributos de calidad, para definir los que se emplearán en la evaluación de la arquitectura propuesta. También son descritas brevemente las técnicas y métodos de evaluación, con el propósito de seleccionar las adecuadas para llevar a cabo el proceso, que finalmente se ejecuta a través de la identificación y comprobación de ciertos escenarios.

4.1 NECESIDADES DE EVALUAR UNA ARQUITECTURA

Evaluar la arquitectura software de un sistema permite identificar los riesgos significativos en su estructura, comprobar que incluyen en su diseño los requerimientos no funcionales y conocer el grado en que se satisfacen los atributos de calidad; constituyendo la forma más económica de evitar todos los cambios que traería consigo el diseño de un sistema que no cumple los requerimientos necesarios.

La evaluación de una arquitectura no da una respuesta específica de sí o no en cuanto a la arquitectura empleada, sino que identifica las deficiencias y fortalezas de la misma. Según los resultados de la evaluación se determina que decisión seguir, que puede ser: mantener la arquitectura propuesta, seguir trabajando con los riesgos identificados tras la evaluación, o si es necesario cambiar la arquitectura de software. Mientras más temprano sean encontrados los errores de un proyecto en desarrollo, más fácil será corregirlos y obtener un producto final sin fallas.

CAPÍTULO 4: EVALUACIÓN DE LA ARQUITECTURA PROPUESTA

Para fortalecer un sistema resulta conveniente evaluar la arquitectura del mismo a través de más de un método de evaluación, debido a que aplicando estos se obtienen diferentes zonas de falla que serán corregidas posteriormente, elevando así la calidad del sistema.

4.2 ASPECTOS RELEVANTES A TENER EN CUENTA PARA EVALUAR LA ARQUITECTURA

A la hora de evaluar una arquitectura candidata se pueden tener interrogantes como: ¿Qué se debe evaluar? ¿Cuándo se debe evaluar? o ¿Quién o quiénes intervienen en la evaluación?

Una arquitectura puede ser evaluada en varios momentos en dependencia de la fase de construcción en la que se encuentre el sistema. Se puede hacer una evaluación temprana o tardía del sistema. La evaluación temprana se realiza sin tener la arquitectura completa, es decir, aún está siendo construida, mientras que la tardía se efectúa una vez terminada e implantada la arquitectura.

En la evaluación de la arquitectura intervienen los mismos integrantes del equipo de desarrollo: el arquitecto, el diseñador y el administrador del proyecto, pero se puede contratar a un grupo de especialistas que realicen la evaluación deseada. El cliente también interviene de cierta forma en la evaluación, debido a que los resultados obtenidos son de su interés, puesto que decidirán si se continúa con la arquitectura prevista o si se hace necesaria la suspensión del proyecto.

4.2.1 ATRIBUTOS DE CALIDAD

Los requerimientos no funcionales son conocidos también como atributos de calidad, pues son aspectos que afectan en gran medida la calidad de los componentes que integran un sistema. Para obtener un software que satisfaga todas las necesidades de los que interactúan con él, se deben tener en cuenta estos atributos para definir la estructura y el comportamiento del mismo, influyendo así en la arquitectura del sistema.

El trabajo con los atributos de calidad es un tanto engorroso, pues los cambios realizados en la arquitectura con el objetivo de mejorar un atributo específico puede afectar de forma negativa a otro, siendo necesario establecer un balance entre los mismos.

CAPÍTULO 4: EVALUACIÓN DE LA ARQUITECTURA PROPUESTA

Debido a que las decisiones en la arquitectura determinan los atributos de calidad del sistema, será posible saber que la arquitectura seleccionada es correcta evaluando el resultado que han producido dichas decisiones sobre los atributos de calidad.

Existen diferentes atributos de calidad que se pueden tomar en cuenta a la hora de evaluar una arquitectura. En la presente investigación se decidió usar los atributos definidos por el Modelo ISO 9126 (27). Este identifica seis atributos básicos de calidad que pueden estar presentes en cualquier producto de software, los cuales se relacionan a continuación:

- Funcionalidad
- Confiabilidad
- Eficiencia
- Usabilidad
- Mantenibilidad
- Portabilidad

El estándar provee una descomposición de los atributos en subatributos (28), que se muestran en el Anexo 2.

4.2.2 TÉCNICAS DE EVALUACIÓN

Las técnicas de evaluación de la arquitectura se clasifican en cualitativas (escenarios, cuestionarios y listas de chequeo) y cuantitativas (métricas, normas, y máximos y mínimos teóricos).

Las técnicas cualitativas se emplean cuando la arquitectura está siendo construida y arrojan como resultados respuestas del tipo sí o no. Estos tipos de técnicas posibilitan evaluar una arquitectura o establecer comparaciones entre arquitecturas candidatas y determinar cual satisface más un atributo de calidad específico.

Las técnicas cuantitativas se aplican ya implementada la arquitectura. Estas están encaminadas a obtener valores para la toma de decisiones en cuanto a los atributos de calidad. Los resultados son obtenidos luego de comparar los valores de los atributos con los márgenes máximos y mínimos establecidos para evaluar la arquitectura candidata. Esta técnica tiene la desventaja de depender del

conocimiento de los valores máximos y mínimos de la medición que se está empleando en la comparación, los cuales son difíciles de predecir.

En vista de que el interés es tomar decisiones de tipo arquitectónico en las fases tempranas del desarrollo, son necesarias técnicas que requieran poca información detallada y puedan conducir a resultados relativamente precisos. Las técnicas de evaluación más empleadas por los arquitectos son las cualitativas debido al costo que implica realizar una evaluación cuantitativa.

4.2.3 MÉTODOS DE EVALUACIÓN DE LA ARQUITECTURA

Dentro del conjunto de métodos existentes para evaluar las arquitecturas se encuentran los siguientes:

Método de Análisis de Arquitectura de Software (SAAM)

El Método de Análisis de Arquitecturas de Software o SAAM (Software Architecture Analysis Method) fue el primero que se difundió y documentó. “Este se centra en la enumeración de un grupo de escenarios que representan principalmente los posibles cambios a los que podrá someterse el sistema en el futuro. En sus inicios fue creado con el propósito de analizar la modificabilidad de una arquitectura, pero hoy en día es muy útil para evaluar rápidamente atributos de calidad tales como modificabilidad, portabilidad e integrabilidad” (29).

Dichos atributos interactúan de manera tal, que introducir mejoras en uno en específico provocaría el empeoramiento de otros; basado en esto, el método SAAM contribuye a analizar las decisiones arquitectónicas que afectan a las interacciones de los atributos de calidad. “Con la aplicación de este método, si el objetivo de la evaluación es una sola arquitectura, se obtienen los lugares en los que la misma puede fallar, en términos de los requerimientos de modificabilidad. Para el caso en el que se cuenta con varias arquitecturas candidatas, el método produce una escala relativa que permite observar qué opción satisface mejor los requerimientos de calidad con la menor cantidad de modificaciones.” (30)

Este método de evaluación no fue seleccionado para la evaluación de la arquitectura propuesta debido a que se centra fundamentalmente en la modificabilidad del sistema.

Método de Análisis de Acuerdos de Arquitectura de Software (ATAM)

"El Método de Análisis de Acuerdos de Arquitectura de Software o ATAM (Architecture Tradeoff Analysis Method) se basa en tres áreas diferentes, los estilos arquitectónicos empleados, el análisis de los atributos de calidad y el método SAAM" (30), descrito anteriormente. Este método muestra la manera en que una arquitectura específica satisface determinados atributos de calidad y proporciona una visión de como interactúan los atributos de calidad entre sí, estas interacciones son los tipos de acuerdos que se establecen entre ellos. "ATAM se concentra en la identificación de los estilos arquitectónicos utilizados." (29)

Este método de evaluación no fue seleccionado para la evaluación de la arquitectura propuesta debido a que se centra fundamentalmente en los estilos arquitectónicos empleados.

Revisión de Diseño Activo (ADR)

"El método de Revisión de Diseño Activo o ADR (Active Design Review) se emplea en la evaluación de diseños bien detallados de unidades de software, tales como componentes. Se centra en la calidad y el nivel de completamiento de la documentación, y en el nivel de conveniencia y suficiencia de los servicios que contiene el diseño propuesto." (30)

Este método de evaluación no fue seleccionado para la evaluación de la arquitectura propuesta debido a que se centra fundamentalmente en la calidad de la documentación del diseño propuesto para la arquitectura.

Revisiones Activas para los Diseños Intermedios (ARID)

"El método ARID (Active Reviews for Intermediate Designs) es conveniente para realizar la evaluación de diseños parciales en las etapas tempranas del desarrollo." (30) ARID es un híbrido entre los métodos ADR y ATAM. Estos últimos presentan características útiles para la evaluación de diseños parciales, pero no son muy satisfactorios de forma separada. En ADR, los involucrados en la evaluación reciben documentación detallada y llenan sus cuestionarios de forma independiente. En ATAM se evalúa toda la arquitectura. Dadas las anteriores características y la necesidad de realizar

evaluaciones tempranas de los diseños de la arquitectura, el método ARID se concibió como resultado de la combinación de ADR y ATAM, para efectuar la evaluación temprana de los diseños de una arquitectura de software. Este "recoge de ADR, la fidelidad de las respuestas que se obtiene de los involucrados en el desarrollo y del ATAM la idea de que sean los involucrados con el sistema los encargados de generar los escenarios." (30)

Este método de evaluación fue el seleccionado debido a que permite realizar la evaluación temprana de la arquitectura.

4.3 EVALUACIÓN DE LA ARQUITECTURA PROPUESTA

Para evaluar la arquitectura propuesta se tomaron en cuenta los atributos de calidad definidos por la ISO 9126: Funcionalidad, Confiabilidad, Eficiencia, Usabilidad, Mantenibilidad y Portabilidad; y como método de evaluación el ARID, ya que evalúa tempranamente la arquitectura de software, obteniendo las zonas de falla del sistema.

ARID establece nueve pasos agrupados en dos fases principales (30), los cuales se relacionan a continuación:

Fase1: Actividades Previas

1. Identificación de los encargados de la revisión: Definir quiénes llevarán a cabo la evaluación de la arquitectura.
2. Preparar el informe de diseño: El diseñador es el encargado de elaborar un informe donde se explique detalladamente el diseño, para capacitar a los encargados de la revisión sobre las cuestiones fundamentales del sistema.
3. Preparar los escenarios base: El líder del grupo de revisión define junto con el diseñador un conjunto de escenarios para realizar la evaluación.
4. Preparar los materiales: Se confeccionan los materiales que serán presentados en la fase siguiente. Se convoca a una reunión con los involucrados.

Fase2: Revisión

5. Presentación del ARID: Se explican los pasos del ARID a los involucrados en la evaluación.

6. Presentación del diseño: Se explica el diseño a los involucrados en la evaluación.
7. Lluvia de ideas y establecimiento de prioridad de escenarios: El grupo de revisión define los escenarios finales que se emplearán para realizar pruebas sobre el diseño; y determinan la prioridad de los mismos.
8. Aplicación de los escenarios: El líder del grupo de revisión es el encargado de probar que el diseño provea los escenarios, comenzando por los de mayor prioridad, hasta que concluya el tiempo estimado de la evaluación, se hayan analizado los escenarios de mayor prioridad, o la conclusión de la revisión satisfaga a los implicados.
9. Resumen: Finalmente el líder del grupo de revisión se reúne con los implicados y se debaten las opiniones de los participantes con respecto a la eficiencia de la revisión.

En este método los atributos de calidad se emplean para evaluar el grado en que estos se satisfacen en cada uno de los escenarios definidos, tomando como guía los pasos establecidos. “Un escenario es una descripción resumida de funcionalidades deseadas por el sistema, o cambios que podría sufrir este en el futuro” (30).

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

4.3.1 PRESENCIA DEL ATRIBUTO FUNCIONALIDAD

La arquitectura de software debe lograr la funcionalidad del sistema, mediante el cumplimiento de los requerimientos del usuario, que son tenidos en cuenta desde la versión inicial de la arquitectura hasta su completamiento con la implementación de todos los casos de uso del sistema. Para una correcta funcionalidad es necesario que el software manifieste características tales como la adecuación, la idoneidad, la seguridad de acceso y la interoperabilidad.

A continuación se verifican los escenarios propuestos para evaluar este atributo de calidad:

Escenario #1: El sistema cumple con los requisitos funcionales solicitados por el cliente y solo con estos.

La arquitectura propuesta satisface los requisitos funcionales planteados por los usuarios. Brinda soporte a las acciones de los registros que conforman el SIGM, posibilitando la gestión de los datos primarios de un paciente, la gestión de la historia clínica, la gestión de los datos complementarios de los pacientes con discapacidad, retraso mental, malformaciones congénitas, enfermedades genéticas, entre otros, así como posibilita la gestión de reportes sobre los datos de los mismos. El sistema se concreta a las necesidades del usuario, no realiza predicciones estadísticas sobre los datos, ni gestiona información fuera del negocio, pues no han sido requisitos funcionales expuestos por el cliente.

Escenario #2: Se debe garantizar el acceso a las acciones de forma segura en todo momento.

El sistema gestiona la seguridad a través de la autenticación de usuario y credenciales. La autenticación será la primera acción del usuario en el sistema y consistirá en suministrar un nombre de usuario único y una contraseña que debe ser de conocimiento exclusivo de la persona que se autentica.

Cada acción antes de ejecutarse es pasada por un filtro especial que verifica si el usuario actual está autenticado y tiene privilegios de acceder a la acción requerida. Dichos privilegios se gestionan a través de credenciales que se definen bajo un nombre, y permiten organizar la seguridad en grupos. Para cada petición, el sistema se encarga de buscar la acción solicitada por el usuario y verifica que se satisfagan los permisos necesarios para acceder a ella. El comportamiento del sistema ante el intento de acceso de un usuario a una acción depende de las credenciales de este.

- Si el usuario está autenticado y tiene las credenciales apropiadas, entonces la acción se ejecuta.
- Si el usuario no está autenticado, es redireccionado a la página de login.

- Si el usuario está autenticado, pero no posee las credenciales apropiadas, será redirigido a la página por defecto que informa que no tiene acceso a la petición.

Escenario #3: Se emplea un mecanismo de cierre de sección de usuario en caso de pasado determinado tiempo de inactividad.

Las sesiones en el sistema expiran automáticamente a los 30 minutos.

Escenario #4: Existe una infraestructura física que limite el acceso a los datos del sistema por parte de usuarios maliciosos.

Los únicos archivos que se pueden acceder directamente por el cliente son los que se encuentran en el directorio web del proyecto en el nodo de Presentación, y contiene solamente la vista del sistema. Los datos del sistema se encuentran en los servidores de datos que poseen únicamente conectividad punto a punto con el servidor de aplicación, el que a su vez se conecta únicamente con este y con el de Presentación, siendo el puente entre ambos. El usuario para violar el sistema y llegar a los datos directamente debe burlar el acceso a tres servidores.

Escenario #5: Existe un mecanismo de registro de eventos.

En caso de fallas en la ejecución de una petición, el sistema puede comprobar mediante los registros de logs, la traza generada por el proceso que intentó ejecutarse. Cada evento es introducido en el archivo de log de la aplicación como una línea de código, que incluye la fecha y hora en que ha sido creada, el tipo de evento, el objeto que se ha procesado y otros detalles relevantes que dependen del tipo de evento registrado. Los archivos de log contienen información como las consultas que se han enviado a la base de datos, las plantillas que han sido procesadas, las llamadas que se han realizado entre objetos, entre otros.

Escenario #6: El sistema no es vulnerable a ataques de tipo XSS

El sistema presenta un mecanismo de escape ante el riesgo de que los contenidos introducidos puedan incluir scripts y otros elementos maliciosos que se encargan de realizar ataques de tipo XSS (cross-site scripting).

4.3.2 PRESENCIA DEL ATRIBUTO CONFIABILIDAD

La confiabilidad del sistema está relacionada con la capacidad de este para reaccionar ante fallos internos o externos que puedan afectarlo. Además incluye la validez de la información que brinda.

A continuación se verifican los escenarios propuestos para evaluar este atributo de calidad:

Escenario #1: Los datos introducidos por el usuario deben ser los más correctos posible.

El sistema lleva a cabo la validación de los formularios para lograr una mayor confianza sobre los datos con que opera. El funcionamiento básico de este mecanismo consiste en que si el usuario introduce datos no válidos y envía el formulario, la próxima página que se muestra contiene los datos con los mensajes de error, posibilitando así al usuario arreglarlos antes de volver a enviar el formulario.

Escenario #2: Los registros de SISalud no responden a las solicitudes del sistema.

El sistema al obtener los datos de los registros de SISalud, guarda esta información en su base de datos. El objetivo es que si existiera alguna falla en estos servicios, el sistema pueda seguir operando normalmente, consumiendo los datos directamente de su base de datos.

4.3.3 PRESENCIA DEL ATRIBUTO EFICIENCIA

La eficiencia está relacionada con los tiempos de respuesta, de procesos y de potencia que el sistema puede brindar. Además un sistema se puede calificar como eficiente, según la capacidad que presente a la hora de emplear la cantidad de recursos y los tipos de recursos que necesita para funcionar de forma adecuada.

A continuación se verifican los escenarios propuestos para evaluar este atributo de calidad:

Escenario #1: Los tiempos de respuesta a las peticiones de los usuarios deben ser mínimos.

Debido a que el sistema debe ser utilizado por un gran número de usuarios, cuenta con un alto rendimiento y optimización.

El despliegue propuesto para el sistema presenta dos servidores de datos donde se mantendrá la base de datos del sistema. Estos implementan una estrategia de replicación que permite balancear la carga, encargándose el servidor maestro de las consultas de modificación, y el esclavo de las correspondientes a la selección, aumentando los tiempos de respuesta y facilitando la recuperación inmediata del sistema ante el fallo de uno de ellos.

El sistema optimiza la capa de la modelo (clasificada como la más lenta) inicializándola solamente cuando se produce una acción que lo requiere utilizar, garantizando que aquellas páginas que no lo emplean no se penalicen por la carga del mismo. Además como la velocidad de ejecución de una consulta depende del número de resultados que devuelve, se optimizan los métodos del modelo para restringir el número de resultados. Por otra parte el modelo minimiza el número de consultas a través de Joins, siempre y cuando el objetivo sea visualizar atributos de las dos tablas vinculadas, mejorando así el rendimiento global de la página. El diseño de la base de datos del sistema, por su parte comprende la definición de índices para aumentar la velocidad de ejecución de las consultas realizadas a columnas que no son claves primarias.

El sistema presenta un tiempo de respuesta a las peticiones relativamente rápido, gracias al mecanismo de cache de configuración que le facilita Symfony. Además garantiza que algunas páginas web respondan más rápido, debido a que no se hace necesario recargarla completamente cada vez que el usuario realiza cambios con el uso de Ajax, incrementando la interactividad y la rapidez.

4.3.4 PRESENCIA DEL ATRIBUTO USABILIDAD

La usabilidad expresa la capacidad que tiene el software para ser entendido, aprendido y operado por el usuario, manifestándose además en el grado de atracción que logre sobre el este.

A continuación se verifican los escenarios propuestos para evaluar este atributo de calidad:

Escenario #1: El sistema debe ser entendible y fácil de usar.

El sistema presenta al usuario una interfaz atractiva e interactiva, con un menú general que lo guía en la búsqueda de la información que necesita. Además implementa patrones de diseño gráfico que

garantizan una estructura afín a todas las páginas de los módulos, estableciendo iguales íconos para acciones similares y regularidades en el orden de aparición de algunos campos de formularios.

Otra característica que contribuye a la facilidad de uso del sistema es la forma en que trata el enrutamiento. Se trata de un mecanismo encargado de reescribir las URL para hacer más entendible su aspecto. La URL que se le muestra al usuario se relaciona con el recurso solicitado, y no está obligada a guardar relación con la instrucción del servidor necesaria para completar su petición. Configurar la estructura interna de la aplicación no afecta el aspecto hacia el exterior que presentan las URL.

4.3.5 PRESENCIA DEL ATRIBUTO MANTENIBILIDAD / PORTABILIDAD

La portabilidad y la mantenibilidad de un sistema lo hacen sumamente potente. Dentro de este atributo de calidad se enmarcan características como poder cambiar fácilmente un componente para darle mantenimiento o para agregarle nuevas funcionalidades. En dependencia del nivel de portabilidad que presente el sistema se migrará de sistema operativo o de gestor de base de datos de una forma rápida y sencilla, sin que estos cambios afecten las funcionalidades. Las decisiones arquitectónicas tomadas en cuanto a tecnologías y herramientas informáticas para dar soporte al desarrollo del software afectan considerablemente a este atributo de calidad.

A continuación se verifican los escenarios propuestos para evaluar este atributo de calidad:

Escenario #1: El sistema puede cambiar alguna de sus partes fácilmente.

Gracias a la implementación del patrón arquitectónico MVC se consigue un mantenimiento más sencillo del sistema, puesto que acciones como sustituir las vistas, agregar nuevas funcionalidades al controlador o cambiar de base de datos, no afecta a los elementos externos al cambio.

Escenario #2: El sistema podrá cambiar de gestor de base de datos con facilidad.

El sistema logra un alto nivel de portabilidad ya que puede migrar de gestor de base de datos con tan solo cambiar una línea de código. La capa de acceso a datos que presenta le permite utilizar objetos en vez de registros y clases en vez de tablas, posibilitando agregar fácilmente métodos asesores e

invocar a los métodos de un objeto desde diferentes partes de la aplicación, logrando una alta reutilización. Por su parte la capa de abstracción a base de datos posibilita generar consultas con una sintaxis independiente de la base de datos, permitiéndole al sistema migrar de base de datos o de gestor de base de datos con facilidad.

Escenario #3: El sistema podrá ser instalado en varios sistemas operativos.

Para implementar el sistema se definió como lenguaje de programación PHP5 que es multiplataforma, posibilitando que el producto se pueda instalar tanto en servidores Linux como Windows.

4.3.6 PRESENCIA DEL ATRIBUTO PORTABILIDAD

Escenario #1: Es fácil instalar las actualizaciones del sistema.

El sistema, una vez desplegado y en ejecución, permite realizar las transferencias de archivos de actualización de forma incremental mediante el uso de la herramienta sincronización rsyn que brinda Symfony mediante SSH. Rsync es una utilidad de línea de comandos que permite realizar una transferencia incremental de archivos de manera rápida y segura, donde solamente se transfieren los datos que han sido modificados, y en caso de que un archivo solo haya sufrido cambios parciales, solo se envían los cambios efectuados. Gracias a este mecanismo las sincronizaciones requieren del envío de muy pocos datos, sin el consiguiente gasto de tiempo y ancho de banda que provocaría hacerlo si se utilizara FTP, que además es muy propenso a cometer errores y puede ocasionar que el sitio web no esté disponible durante el traspaso de la información.

4.4 CONCLUSIÓN

En el presente capítulo se han analizado los elementos fundamentales relacionados con la evaluación de la arquitectura de software, propiciando la selección adecuada de un método de evaluación y de un grupo atributos de calidad para realizar el proceso de evaluación de la arquitectura propuesta.

Para realizar la evaluación, fueron definidos un conjunto de escenarios, dentro de los cuales prevalecieron aquellos que exponen situaciones relacionadas con la seguridad (subatributo de Funcionalidad), debido a la necesidad de restringir el acceso por el alto nivel de sensibilidad que presenta la información genética. Después de este, le siguieron en número los escenarios relacionados con los atributos Portabilidad y Mantenibilidad, y seguidamente se contemplaron los que definen situaciones para evaluar la idoneidad, madurez, tolerancia a fallos, tiempo de uso, y facilidad de comprensión, subatributos propuestos por la ISO 9126.

Luego de seleccionados los escenarios por cada atributo de calidad, se realizó la evaluación, obteniendo como resultado que las funcionalidades deseadas por el sistema fueron satisfactoriamente cumplidas, demostrando la adecuación de la arquitectura propuesta. Se llega a la conclusión de que el sistema es principalmente funcional con respecto a la seguridad, ya que fue evaluado a través de seis escenarios diferentes. Además se comprobó que tiene gran capacidad de adaptabilidad y facilidad de cambio después de la evaluación de los tres escenarios propuestos para los atributos Portabilidad y Mantenibilidad. El sistema respondió satisfactoriamente ante otros escenarios que evaluaron características respecto a la idoneidad, madurez, tolerancia a fallos, tiempo de uso y facilidad de comprensión.

No obstante a los resultados obtenidos en el presente capítulo, se pueden aplicar otros métodos de evaluación para fortalecer el sistema.

CONCLUSIONES

La arquitectura de software es una disciplina que se incorpora en las metodologías de desarrollo de software, e influye en gran medida en el éxito del producto final. Esta identifica los principales componentes del sistema, sus relaciones y restricciones, proporcionando una visión global del mismo. Es capaz de organizar el desarrollo, fomentar la reutilización y hacer evolucionar el sistema.

Con el objetivo de definir la arquitectura de software del SIGM se especificaron los requerimientos arquitectónicos del sistema, propiciando un estudio abarcador de todas las funcionalidades que este debe cumplir. Estos requerimientos constituyeron el punto de partida para la descripción de la arquitectura propuesta, así como para la selección del patrón arquitectónico a utilizar, el cual contribuye a un alto grado de independencia entre sus elementos, haciendo posible el soporte a múltiples vistas del sistema. Mediante la selección de las tecnologías y herramientas informáticas para dar soporte al desarrollo, el sistema cumple con la estandarización de desarrollo acorde al resto de las aplicaciones destinadas al sector de la salud en Cuba, así como contribuye a un entorno de desarrollo confiable, rápido y seguro.

A través del desarrollo de las vistas arquitectónicas definidas por RUP para describir la arquitectura, se logró mostrar la propuesta de solución del sistema desde diferentes perspectivas: la vista de casos de uso mostró las principales funcionalidades que el sistema debe cumplir, la vista lógica permitió observar la forma en están diseñadas las funcionalidades en el interior del mismo, la vista de implementación modeló el empaquetado fisico real de este, y la vista de despliegue especificó la configuración física de nodos y conexiones sobre la que será desplegado.

A partir de la evaluación de la arquitectura propuesta se obtuvo un sistema funcional, específicamente con respecto a la seguridad del mismo, ya que se satisfizo seis escenarios para este. Además se comprobó que tiene gran capacidad de adaptabilidad y facilidad de cambio después de la evaluación de los tres escenarios propuestos para los atributos portabilidad y mantenibilidad. El sistema respondió satisfactoriamente ante otros escenarios que evaluaron características respecto a la idoneidad, madurez, tolerancia a fallos, tiempo de uso y facilidad de comprensión.

RECOMENDACIONES

- Se recomienda para posteriores versiones del SIGM, diseñar servicios web con el objetivo de suministrar datos a otras aplicaciones informáticas para la salud, una vez estas últimas definan que funcionalidades necesitan consumir.
- Se recomienda aplicar otros métodos de evaluación de la arquitectura para aumentar la fortaleza del sistema.

REFERENCIAS BIBLIOGRÁFICAS

1. *Arquitectura del Software: arte y oficio.* **Mollineda, Ramón.** 2005, Revista del Instituto Tecnológico de Informática, pág. 4.
2. **Clements, Paul.** *A Survey of Architecture Description Languages.* Alemania : Proceedings of the International Workshop on Software Specification and Design, 1996.
3. **Kazman, R, Clements, P y Bass, L.** *Software Architecture in Practice.* s.l. : SEI Series in Software Engineering: Addison Wesley, 2003.
4. **Jacobson, Ivar, Booch, Grady y Rumbaugh, James.** *El Proceso Unificado de desarrollo de software.* s.l. : Addison Wesley, 1999.
5. **IEEE.** *IEEE Std. 1471-2000, IEEE Recommended Practice for Architectural Description of Software Systems.*
6. **Perry, Dewayne E. y Wolf, Alexander L.** *Foundatiosn for the study of software architecture.* s.l. : ACM SIGSOFT Software Engneering Notes, 1992.
7. **Shaw, Mary y Clements, Paul.** A field guide to Boxology: Preliminary classification of architectural styles for software systems. s.l. : Computer Science Department and Software Engineering Institute, Carrnegie Mellon University, 1996.
8. **Thomas Fielding, Roy.** *Architectural styles and the desing of network-based software architectures* . University of California, Irvine, 2000 : Tesis doctoral .
9. **Shaw, Mary y Garlan, David.** *Software Architecture: Perspectives on an Emerging Discipline* . s.l. : Prentice Hall Publishing, 1996.
10. **Reynoso, Carlos y Kiccillof, Nicolás.** MSDN. Microsoft Developer Network. *Estilos y Patrones en la Estrategia de Arquitectura de Microsoft.* [En línea] 2004,Marzo. http://www.microsoft.com/spanish/msdn/arquitectura/roadmap_arq/style.asp.
11. **UCI.** Conferencia Ingenieria de Software I. Arquitectura y Patrones de diseño. Ciudad de la Habana : UCI, 2007-2008.
12. laWebSemantica. [En línea] Ingeniería Técnica en Informática de Gestión en la Universidad Politécnica de Madrid. Licencia de Creative Commons. , 2008. http://www.lawebsemantica.com/contents/serviciosWeb/serviciosWeb1.html.
13. MSDN. Microsoft Developer Network. *Desarrollo basado en el patrón MVC.* [En línea] http://msdn.microsoft.com/practices/type/Patterns/Enterprise/DesMVC/.
14. **BUSCHMANN, F., MEUNIER, R., ROHNERT, H., SOMMERLAD, P., STAL, M.** *Pattern – Oriented Software Architecture. A System of Patterns.* Inglaterra : John Wiley & Sons, 1996.
15. *Architect Patterns.* [En línea] 13 de March de 2000. [Citado el: 06 de 02 de 2008.] http://www.rationalrose.com/models/architectural_patterns.htm.
16. **Doberkat, Ernst-Erich.** *Pipes and filters: Modelling a software architecture through relations.* 2002, Junio.
17. **Stiger, P. R y Gamble, R. F.** *Blackboard systems formalized within a software architectural style.*
18. **Burbeck, Steve.** Application programming in Smalltalk-80: How to use Model-View-Controller (MVC). [En línea] http://st-www.cs.uiuc.edu/users/smarch/st-docs/mvc.html.
19. **Buschmann, Frank, y otros.** *Pattern-oriented software architecture (POSA).* s.l. : John Wiley & Sons. LTD, 1996.

REFERENCIAS BIBLIOGRÁFICAS

20. **Shaw, Mary.** *"Some Patterns for Software Architecture" en Pattern Languages of Program Design, Vol. 2.* s.l. : Addison-Wesley, 1996.
21. **Allen, Robert.** Technical Report, CMU-CS-97-144. *A formal approach to Software Architecture.* 1997.
22. **Wolf, Alexander.** (ISAW-2). ACM SIGSOFT Software Engineering Notes. *"Succeedings of the Second International Software Architecture Workshop" .* enero de 1997.
23. **Rumbaugh, James, Ivar, Jacobson y Grady., Booch.** *El Lenguaje Unificado de Modelado. Manual de Referencia.* . s.l. : Addison Wesley, 1998, Noviembre.
24. **SOFTEL.** *Documento sobre la Arquitectura de Software para los componentes a emplear por el Sistema de Información para la Salud.* Ciudad de la Habana : s.n., 2006.
25. ComponentSource. The Definitive Source of Software Components. [En línea] Copyright 1996-2008 ComponentSource®. http://www.componentsource.com/services/about-us/components.html.
26. **Corporation, Copyright (C) IBM.** Ayuda Rational Unified Process. Version 7.0. 2005.
27. ISO. International Organization for Standarization. *ISO/IEC 9126-1:2003.* [En línea] © 2008 ISO. http://www.iso.org/.
28. **R., Pressman.** *Ingeniería de Software. Un Enfoque Práctico. Quinta Edición.* 2002.
29. **Kazman, Rick, Klein, Mark y Barbacci, Mario.** The Architecture Tradeoff Analysis Method. *Software Engineering Institute.* Carnegie Mellon University : s.n.
30. **Clements, Paul, Kazman, Rick y Klein, Mark.** *Evaluating Software Architectures: Methods and Case Studies.* USA, Boston : Addison-Wesley, 2002. ISBN 0-201-70482-X.

BIBLIOGRAFÍA.

BIBLIOGRAFÍA

1. SOFTEL-Soluciones Informaticas. *Portal Arquitectura del MINSAP-MIC.Repositorio de Componenetes.* [En línea] 12 de 2007. [Citado el: 22 de 12 de 2007.] http://10.128.52.28/SitioArquitectura.
2. Web Services and Service-Oriented Architectures . [En línea] Barry & Associates, Inc. http://www.service-architecture.com/.
3. **Reynoso, Carlos y Kicillof, Nicolás.** MSDN, Microsoft Developer Network. *Lenguajes de Descripción de Arquitectura (ADL).* [En línea] [Citado el: 04 de 02 de 2008.] http://www.microsoft.com/spanish/msdn/arquitectura/roadmap_arq/lenguaje.mspx.
4. MSDN, Microsoft Developer Network. *Elección de la arquitectura de capa de presentación correcta.* [En línea] http://www.microsoft.com/spanish/msdn/articulos/archivo/040405/voices/choosing_presentation_layer.mspx.
5. **Reynoso, Carlos.** MSDN. Microsoft Developer Network. *Introducción a la Arquitectura de Software.* [En línea] http://www.microsoft.com/spanish/msdn/arquitectura/roadmap_arq/intro.asp.
6. SISalud. Sistema de Información para la Salud . [En línea] SOFTEL. http://10.128.52.50/Home/home.php .
7. Infomed. *Red Telemática de Salud en Cuba.* [En línea] MISAP-MIC. http://www.sld.cu/red/.
8. **MINSAP-MIC, Grupo de Arquitectura.** *Arquitectura, normas y tecnologías para el desarrollo de aplicaciones.* Ciudad de la Habana : s.n., Junio,2007.
9. WS-I. *Web Services Interoperability Organization.* [En línea] Created by PixelMEDIA, Inc. http://www.ws-i.org/.
10. **IBM.** IBM Service Oriented Architecture (SOA). [En línea] http://www-306.ibm.com/software/solutions/soa/.
11. **Shaw, Mary y Garlan, David.** Technical Report CMU-CS-94-210. *"Characteristics of Higher-Level Languages for Software Architecture".* Carnegie Mellon University : s.n., 1994, Diciembre .
12. **Medvidovic, Neno.** Technical Report UCI-ICS-97-02. *"A classification and comparison framework for software Architecture Description Languages". .* 1996. .
13. **Schmuller, Joseph.** *Aprendiendo UML en 24 horas.* 1999 : Prentice Hall.
14. **Larman, Craig.** *UML y Patrones. Introducción al análisis y diseño orientado a objetos.* s.l. : Prentice Hall., 1999.
15. **Sun.** MySQL. [En línea] 1995-2008 MySQL AB. All rights reserved. http://dev.mysql.com/.
16. **Fuentes, Lidia, Troya, José M. y Vallecillo, Antonio.** Desarrollo de Software Basado en Componentes. Campus Teatinos, s/n. 29071 Málaga, Spain. : Dept. Lenguajes y Ciencias de la Computación. Universidad de Málaga. ETSI Informática.
17. **Terreros, Julio Casal.** MSDN. Microsoft Developer Network. [En línea] http://www.microsoft.com/spanish/msdn/comunidad/mtj.net/voices/MTJ_3985/default.aspx.
18. WebCab Components - produced for developers by developers. [En línea] 1999-2006 WebCab Components. http://webcabcomponents.com/.

BIBLIOGRAFÍA.

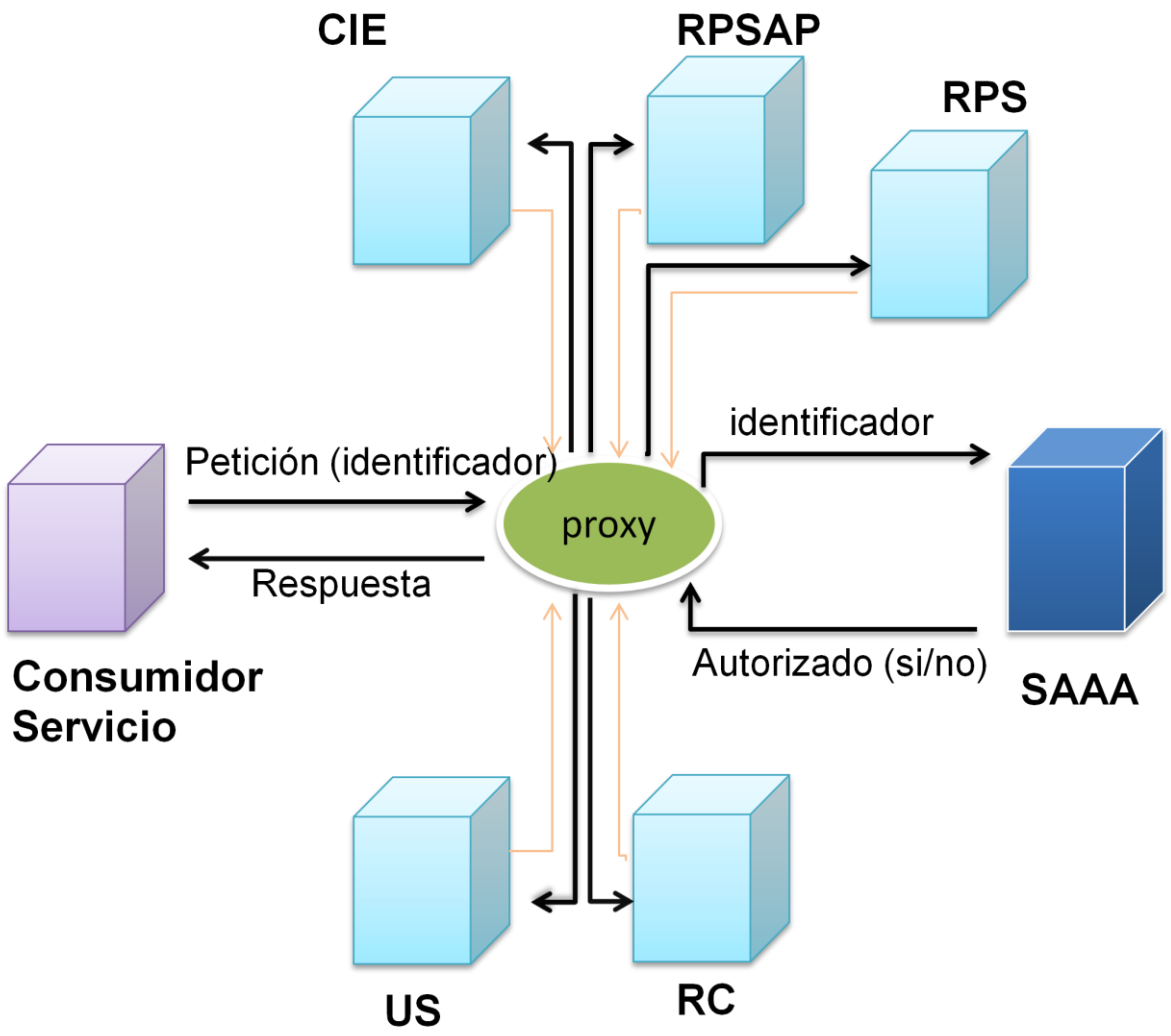
19. **The Apache Software Foundation .** Apache. HTTP SERVER PROJECT. [En línea] Copyright © 2008 . http://httpd.apache.org/.
20. **The Apache Software Foundation.** The Apache Software Foundation. . [En línea] Copyright © 2008 . http://www.apache.org/.
21. **Debian.** Debian-The Universal Operating System. [En línea] 17 de Mayo de 2008 . http://www.debian.org/releases/sarge/.
22. **Sun.** MySQL. [En línea] 2008 . http://www.mysql.com/.
23. **Mozilla.** Mozilla. Firefox 2. Systema Requirements. [En línea] Copyright © 2005–2008 Mozilla. All rights reserved. http://www.mozilla.com/en-US/firefox/system-requirements.
24. **Microsoft Corporation.** Microsoft Download Center. *Internet Explorer.* [En línea] © 2008 Microsoft Corporation. All rights reserved. http://www.microsoft.com/downloads/details.aspx.
25. **Mozilla Europe and Mozilla Foundation.** Mozilla Firefox - System Requirements | Mozilla Europe. [En línea] Last update: 2008-04-09. http://www.mozilla-europe.org/en/products/firefox/system-requirements/.
26. **The PHP Group.** PHP. [En línea] Copyright 2001-2008. http://www.php.net/downloads.php.
27. Ximbiot. *Your source for CVS & Subversion support.* [En línea] http://ximbiot.com/.
28. **Free Software Foundation.** CVS - Concurrent Versions System. [En línea] Derek Robert Price & Ximbiot, Copyright (C) 2005-2006. http://www.nongnu.org/cvs/.
29. **Tigris.org. Open Source Software Engineering Tools.** Subversion. [En línea] © 2006 CollabNet. . http://subversion.tigris.org/.
30. **Collins-Sussman, Ben, . Fitzpatrick, Brian W y Michael, Pilato C.** Version Control with Subversion For Subversion 1.1 (book compiled from Revision 1337). [En línea] Copyright © 2002, 2003, 2004, 2005. http://svnbook.red-bean.com/.
31. **Visual Paradigm International .** Visual Paradigm. [En línea] 2008. http://www.visual-paradigm.com/.
32. **Sparx Systems .** [En línea] 2000 - 2008 Sparx Systems Pty Ltd. All rights Reserved. http://www.sparxsystems.com.au.
33. **Tigris.org. Open Source Software.** ArgoUML. [En línea] © 2006 CollabNet. CollabNet is a registered trademark of CollabNet, Inc. http://argouml.tigris.org/index.html.
34. **Bruno Pagès.** BOUML. [En línea] Copyright (C) 2004-2008 by Bruno Pagès. http://bouml.free.fr/.
35. **OMG.** Unified Modeling Language. *UML Resource Page.* [En línea] Copyright © 1997-2008 Object Management Group. http://www.uml.org/.
36. **The Eclipse Foundation.** eclipse. [En línea] Copyright © 2008 . http://www.eclipse.org/downloads/index.php.
37. **The Eclipse Fundation.** eclipse plugin central. [En línea] http://www.eclipseplugincentral.com/modules.php.
38. **KDE Web Dev Team.** Quanta Plus. [En línea] Copyright 2005. http://quanta.kdewebdev.org/.
39. **Potencier, Fabien y Zaninotto, François.** *Symfony, la guía definitiva.* . 20 de febrero de 2008.
40. librosweb.es. *Symfony, la guia definitiva.* [En línea] http://www.librosweb.es/symfony.
41. symfony.es. [En línea] 2008. http://www.symfony.es/.
42. **The PHP Foundation.** DesarrollosPHP . *Kumbia: Framework PHP5.* [En línea] 2008. http://www.desarrollosphp.com/2008/03/02/kumbia-framework-php5/.

BIBLIOGRAFÍA.

43. **Kumbia PHP Framework.** [En línea] Copyright 2007. . http://www.kumbiaphp.com/.
44. **Zend Technologies.** Zend Framework. [En línea] 2006 - 2008 . http://framework.zend.com/.
45. **Zend Technologies .** Zend. The PHP Company . [En línea] http://www.zend.com/en/community/.
46. Comunidad Zend Framework. [En línea] http://zendframework.programania.net/.
47. **Mantis.** Mantis. Bugtracking System. [En línea] http://www.mantisbt.org/.
48. **JumpBox, Inc.** JumpBox. [En línea] http://www.jumpbox.com/app/mantis.
49. GNUnettProyect. [En línea] https://gnunet.org/mantis/view_all_bug_page.php.
50. **Edgewall Software.** trac. *Integrate SCM & Project Management.* [En línea] Copyright © 2003-2007. All rights reserved. http://trac.edgewall.org/.
51. symfony develpment. [En línea] Powered by Trac by Edgewall Software. http://trac.symfony-project.com/.
52. **TGMT-systems Inc. .** BugTrack.com. [En línea] Copyright © 2001-2008 . http://www.bug-track.com/.
53. **Casanovas,Josep.** Alzado.org . *Usabilidad y arquitectura del software.* [En línea] Julio de 2004.
54. **PERALTA, ARTURO.** UCALIDAD EN SISTEMAS BASADOS EN COMPONENTES. Calidad y Medición de Sistemas de Información. NIVERSIDAD DE CASTILLA-LA MANCHA. ESCUELA SUPERIOR DE INFORMÁTICA. : s.n., 2008.
55. **Carnegie Mellon University.** Software Engineering Institute. CarnegieMellon. *Attribute Tradeoff Análisis Method.* [En línea] © 2008 . http://www.sei.cmu.edu/architecture/ata_method.html. .
56. **Wesley, Addison.** *Design & Use of Software Architectures.* .
57. **Softel.** *Documento sobre la Arquitectura de Software para los componentes a emplear por el Sistema de Información para la Salud.* 2006.
58. **Jacobson, Ivar.** *The Unified Software Development Process.* s.l. : Addison Wesley Professional, 1999.
59. Moors, Copyright © 1999-2000 Michael. *Architectural Patterns.* [En línea] 13 de March de 2000. [Citado el: 06 de 02 de 2008.] http://www.rationalrose.com/models/architectural_patterns.htm.
60. **Jacobson, Ivar, Booch, Grady y Rumbaugh, James.** *El proceso Unificado de desarrollo de software.* s.l. : Addison Wesley, 1999.
61. Wikimedia Foundation, Inc. *Wikipedia. La enciclopedia libre.* [En línea] 23 de 12 de 2007. [Citado el: 04 de 02 de 2008.] http://es.wikipedia.org/wiki/Componentes_de_software.
62. **Astudillo, Hernán.** Arquitectura de Software III. Elaboracion. s.l. : Departamento de Informatica. Universidad Técnica Federico Santa María, 2004.
63. **Perry, D. E. y Wolf, A. L.** *Foundation for the Study of Software Architecture.* s.l. : ACM SIGSOFT Software Engineering Notes, 1992.
64. **Corporation, Microsoft.** msdn. [En línea] 2008. [Citado el: 04 de 02 de 2008.] http://www.microsoft.com/spanish/msdn/arquitectura/roadmap_arq/lenguaje.mspx.
65. **Reynoso, Carlos y Kiccillof, Nicolás.** Estilos y Patrones en la Estrategia de Arquitectura de Microsoft. s.l. : UNIVERSIDAD DE BUENOS AIRES, 2004,Marzo.

ANEXOS

Anexo 1. Representación estructural de SISalud



Anexo 2. Atributos y subatributos de calidad – Modelo ISO 9126 (28)

Atributos de Calidad	Subatributos
Funcionalidad	Idoneidad Corrección Interoperatividad Conformidad Seguridad
Confiabilidad	Madurez Tolerancia a fallos Facilidad de recuperación
Usabilidad	Facilidad de comprensión Facilidad de aprendizaje Operatividad
Eficiencia	Tiempo de uso Recursos utilizados
Mantenibilidad	Facilidad de análisis Facilidad de cambio Estabilidad Facilidad de pruebas
Portabilidad	Facilidad de instalación Facilidad de ajuste Facilidad de adaptación al cambio

GLOSARIO

ACL: Lista de control de acceso es un concepto de seguridad informática utilizado para fomentar la separación de privilegios cuyas siglas vienen dadas por su nombre en inglés Access Control List.

ADL: Lenguaje de descripción de arquitectura cuyas siglas vienen dadas por su nombre en inglés Architecture Description Language y se utiliza para describir una arquitectura de software.

Ajax: Técnica de desarrollo web para crear aplicaciones interactivas cuyas siglas vienen dadas por su nombre en inglés Asynchronous JavaScript And XML que significa JavaScript asincrónico y XML.

Alzheimer: Enfermedad neurodegenerativa, que se caracteriza por una pérdida progresiva de la memoria y de otras capacidades mentales, a medida que las células nerviosas (neuronas) mueren y diferentes zonas del cerebro se atrofian.

Aplicación web: Sistema informático utilizado por los usuarios que acceden a un servidor web a través de Internet o de una Intranet.

BSD: Licencia que se le otorga fundamentalmente a los sistemas BSD que significa Berkeley Software Distribution. Esta licencia pertenece a un grupo de licencias de software libre y permite el uso del código fuente en software no libre.

C++: Lenguaje de programación para el desarrollo de software

CNGM: Centro Nacional de Genética Médica, este es el centro sede de todos los estudios de genética desarrollados en el país y de la red nacional de genética conformada por los Centros de Genética del país.

Constructor: Método especial para inicializar una nueva instancia de una clase y presenta el mismo nombre de Esta.

Compilador: Programa informático capaz de traducir un programa escrito en un lenguaje de programación de alto nivel (más entendible desde el punto de vista humano) a otro de nivel inferior (lenguaje de máquina) obteniendo un programa más manejable por la computadora.

CSS: Una hoja de estilo en cascada es un lenguaje formal empleado para definir la presentación de un documento estructurado escrito en HTML o XML y sus siglas viene dadas por su nombre en inglés Cascading Style Sheets.

CU: Un caso de uso es una técnica de ingeniería de software para la captura de requisitos deseados por cliente en el desarrollo de un nuevo sistema o una actualización de software.

CVS: Sistema de control de versiones cuyas siglas vienen dadas por su nombre en inglés Concurrent Version System.

Depurador: Programa que permite depurar o limpiar los errores de otro programa informático.

Editor de código: Programa que permite escribir código fuente.

Entorno de desarrollo: Conjunto de nodos de hardware y herramientas informáticas de software para el desarrollo de una aplicación.

Entorno de producción: Conjunto de nodos de hardware y herramientas informáticas de software para el despliegue de la aplicación.

EPS: Formato estándar para insertar imágenes en otros documentos. Normalmente incluye una versión a baja resolución para el visionado a pantalla. Sus siglas vienen dadas por su nombre en inglés Encapsulated Postscript.

Framework: Se conoce como marco de trabajo y es un conjunto de conceptos, metodologías y herramientas de administración y diseño para el desarrollo de forma estandarizada de una aplicación.

GIF: Formato binario de archivos que contienen imágenes. Es utilizado por su alta capacidad de compresión. Sus siglas vienen dadas por su nombre en inglés Graphics Interchange Format

GPL: Licencia diseñada para garantizar la libertad de compartir y modificar el software cuyas siglas vienen dadas por su nombre en inglés General Public License.

Grupo de Arquitectura MINSAP-MIC: Grupo de Arquitectura de Software constituido por acuerdo entre el MINSAP y el MIC que tiene como objetivo coordinar centralmente la integración de las aplicaciones desarrolladas como parte del proceso de informatización del sistema de salud cubano.

GUI: Interfaz gráfica de usuario permite a través del uso y la representación visual la interacción de forma amigable entre el usuario y el sistema informático, sus siglas vienen dadas por su nombre en inglés Graphical User interface.

Herramientas CASE: Conjunto de aplicaciones informáticas orientadas al incremento de la productividad en el desarrollo de software, las siglas CASE vienen dadas por su nombre en inglés Computer Aided Software Engineering que se conoce como Ingeniería de Software Asistida por Computadoras.

HTML: Lenguaje de marcas de hipertexto cuyas siglas vienen dadas por su nombre en inglés Hyper Text Markup Language y es empleado para la construcción de páginas web.

IEEE: Asociación técnico-profesional mundial dedicada entre otras cosas a la estandarización, su siglas vienen dadas por su nombre en inglés de The Institute of Electrical and Electronics Engineers, Instituto de Ingenieros Eléctricos y Electrónicos.

IBM: Empresa internacional mundialmente conocida por la venta de tecnología informática cuyas siglas vienen dadas por su nombre en inglés International Business Machines Corporation.

IDE: Un entorno de desarrollo integrado es un software compuesto por un grupo de herramientas integradas para facilitar el trabajo del programador cuyas siglas vienen dadas por su nombre en inglés Integrated Development Environment.

Infomed: Red Telemática de Salud en Cuba.

Ingeniería inversa: Método que obtiene información de un producto ya liberado con el propósito de determinar qué elementos lo componen, qué utiliza para su funcionamiento y cómo fue fabricado.

JavaScript: Lenguaje de programación para el desarrollo de software.

JPG: Técnica de compresión de imágenes que casi no afecta la calidad, reduce el peso en bytes de las imágenes hasta 5%. Sus siglas vienen dadas por el diminutivo de su nombre en inglés JPEG (Joint Photographic Experts Group)

Linux: Sistema operativo de código abierto.

Login: Momento de autenticación al ingresar a un servicio o sistema.

MIC: Ministerio de la Informática y las Comunicaciones en Cuba.

MINSAP: Ministerio de Salud Pública en Cuba.

MIT: Licencia de software libre propuesta por el Instituto Tecnológico de Massachusetts cuyas siglas vienen dadas por su nombre en inglés Massachusetts Institute of Technology.

MS SQL: Sistema Gestor de Base de Datos

MVC: Patrón arquitectónico Modelo-Vista-Controlador. Identifica tres componentes fundamentales que se corresponden con su nombre, en los que se separan la interfaz del usuario, la lógica del negocio y los datos de la aplicación.

MySQL: Sistema Gestor de Base de Datos.

ORM: Mapeo de objetos a bases de datos que permite el acceso a los datos desde un contexto orientado a objetos a través de una interfaz capaz de traducir la lógica relacional de la base de datos. Sus siglas vienen dadas por su nombre en inglés Object Relational Mapping.

PDA: Computador de mano creado en sus inicios como agenda electrónica que presenta un sistema de reconocimiento de escritura, cuyas siglas vienen dadas por su nombre en inglés Personal Digital Assistant conocido como asistente digital personal.

Plugin: Aplicación informática que interactúa con otra aplicación para aportarle una función o utilidad específica.

PNG: Formato gráfico muy completo especialmente pensado para redes. Sus siglas vienen dadas por su nombre en inglés Portable Network Graphics

Postgre: Sistema Gestor de Base de Datos.

RUP: Metodología de desarrollo cuyas siglas vienen dadas por su nombre en inglés Rational Unified Process. Establece para el desarrollo de un software una serie de flujos de trabajo agrupados en cuatro fases fundamentales.

SGBD: Un sistema gestor de base de datos es un software específico que interviene como interfaz entre la base de datos, el usuario y las aplicaciones que lo emplean.

SIGM: Sistema Informático de Genética Médica, es el sistema que concibe la integración de los registros pertenecientes a los estudios de genética desarrollados en Cuba.

SOA: Arquitectura Orientada a Servicios que define la utilización de servicios para la comunicación entre aplicaciones, cuyas siglas vienen dadas por su nombre en inglés Service Oriented Architecture.

SOAP: Protocolo de comunicación entre aplicaciones a través de mensajes vía Internet cuyas siglas vienen dadas por su nombre en inglés Simple Object Access Protocol.

SISalud: Sistema de Información para la Salud.

SQLite: Sistema Gestor de Base de Datos

SSH: Es un protocolo para acceder a computadoras de forma remota a través de la red, cuyas siglas vienen dadas por su nombre en inglés Secure SHell. Usa técnicas de cifrado para que ningún atacante pueda descubrir el usuario y contraseña de la conexión, ni lo que se escribe durante toda la sesión.

SVN: Sistema de control de versiones que se originó con el fin de sustituir al CVS.

UDDI: Modelo de directorios para publicar los servicios web en Internet cuyas siglas vienen dadas por su nombre en inglés Universal Description, Discovery and Integration.

UNIX: Sistema operativo.

UML: Lenguaje visual para especificar, construir y documentar un sistema de software. Sus siglas vienen dadas por su nombre en inglés Unified Modeling Language.

VP-UML CE: Plataforma de modelado diseñada para el aprendizaje del modelado visual cuyas siglas vienen dadas por su nombre en inglés Visual Paradigm for UML Community Edition.

Web services: Conocidos en español como servicios web, están compuestos por un conjunto de protocolos y estándares que sirven para intercambiar datos entre aplicaciones que han sido desarrolladas en distintos lenguajes de programación y funcionan sobre múltiples plataformas.

Windows: Sistema operativo propietario.

WSDL: Lenguaje de descripción de servicios web para definir el acceso a estos cuyas siglas vienen dadas por su nombre en inglés Web Services Description Language.

XML: Estándar de información cuyas siglas vienen dadas por su nombre en inglés eXtensible Markup Language.

XSS: Vulnerabilidad que se basa en explotar la confianza que tiene el usuario en un determinado sitio web o aplicación, introduciendo scripts y otros tipos de elementos maliciosos en los formularios. Sus siglas vienen dadas por su nombre en inglés Cross Site Scripting.

Yahoo Answers: Mecanismo que presenta Yahoo de publicación de preguntas y respuestas de los usuarios con el objetivo de compartir conocimiento y experiencias.

Yahoo Bookmarks: Servicio de marcadores para almacenar las páginas favoritas de Internet de cada usuario.

YML: Lenguaje de metadatos para representar información cuyas siglas vienen dadas por su nombre en inglés Ain't Another Markup Language.