Universidad de las Ciencias Informáticas

Facultad 6

Título: Definición de la arquitectura del sistema alasARBOGEN 2.0

Trabajo de Diploma para optar por el título de

Ingeniero en Ciencias Informáticas

Autores: Reinier Asencio Correa
Jeanders Silvio Hinojosa Calzada

Tutores: Ing. Reynaldo Alvarez Luna
Ing. Yunier Santana Aldana

Consultante: Ing. Yusdenis Sánchez Perodín

Ciudad de La Habana, Cuba.

Junio 2010

“Año 52 de la Revolución”

Lo importante no es hacer cosas extraordinarias, sino hacer las cosas ordinarias extraordinariamente bien.

Ernesto Che Guevara

DECLARACIÓN DE AUTORÍA

Declaramos ser autores de la presente tesis y reconocemos a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

Para que así conste se firma la presente a los ____ días del mes de _____ del año _____.

Reinier Asencio Correa	Jeanders Silvio Hinojosa Calzada
_____	_____
Firma del autor	Firma del autor
Ing. Yunier Santana Aldana	Ing. Reynaldo Alvarez Luna
_____	_____
Firma del tutor	Firma del tutor

DATOS DE CONTACTO

Ing. Reynaldo Alvarez Luna: Ing. en Ciencias Informáticas, UCI 2008. Profesor en adiestramiento. Tiene 4 años de experiencia en la producción de software para la salud. Ha participado en numerosos eventos y ha obtenido resultados y publicaciones relacionadas con el tema.
Correo electrónico: rluna@uci.cu

Ing. Yunier Santana Aldana: Ing. en Ciencias Informáticas, UCI 2009. Profesor en adiestramiento. Ha estado vinculado a la producción de software de Gestión y para la Salud con una experiencia de 3 años.
Correo electrónico: ysaldana@uci.cu

Ing. Yusdenys Sánchez Parodin: Ing. en Ciencias Informáticas, UCI 2008. Ha estado vinculado a la producción de software para la Salud con una experiencia de 5 años. Ha participado en numerosos eventos y ha obtenido resultados y publicaciones relacionadas con el tema.
Correo Electrónico: ysanchezp@uci.cu

AGRADECIMIENTOS

De Reinier

A Fidel y la Revolución por la creación de esta Universidad.
A mis padres porque son los más grande que tengo en la vida.
A Duonnis, Yulita, Mirley, Alina por la preocupación constante y el apoyo a lo largo de mí vida de estudiante.
A mis compañeros y amigos Russo, Nela, Asdrubal, Renier, Themis y al piquete de la Universidad por aguantarme durante todos estos años.
A mi tutor Reynaldo que siempre ha estado disponible para ayudarnos y apoyarnos en todo lo que fuera necesario.
A Yusdenis e Irina por toda la ayuda que nos brindaron durante la realización de este trabajo de diploma.
A todas las personas que me quieren y que sin esperar nada de mí de una forma u otra siempre me han ayudado.

De Silvio

A la Revolución y al Ministerio del Interior, sin los cuales no hubiera sido posible este sueño.
A mis padres, guías e inspiración en todo momento.
De forma especial a Rey, por dedicarme tanto tiempo y asesoría constante.
A Yusdenis,Irina y Elvismary, por su asesoría y atención.
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 un profesional.

DEDICATORIA

De Reinier

A las personas que han dedicado parte de su vida a mostrarme lo que realmente es importante, mis padres Alicia Correa y Francisco Asencio, a ellos que siempre me han brindado amor, confianza, dedicación y esmero, les dedico este sueño hecho realidad.
A mis hermanos Duonnis y Yulita, a Alina, a Mirley y a toda mi familia en general.
A mi novia Gretel Fernández-Rubio por apoyarme cada momento y tener tanta paciencia.

De Silvio

A mis padres, por educarme y guiarme siempre por el camino correcto, en especial a mi mama a quien le deseo mucha salud...
A Lisset, el amor de mi vida, sin dudas lo mejor que me ha podido pasar
A mis hermanos, Yoanne, Yamile, Yunibe, Yusdani y Silvio Dylan a los cuales les deseo muchos éxitos...
A mis tíos Jorge, Yamisel, Rolandito, Rosy...
A mis sobrinos y primos, Sayuri, miguelito, Jorgito...
A mi otra familia, Chuchi, Ernesto, Mayela, Sonia, Juan, Mondelo, Madelayne, por dejarme formar parte de su familia y tratarme como un hijo más.
A mis amigas y amigos de siempre, Alejandro, Dianelis, Ludmila, Yaima, Kaly y a todo el piquete

RESUMEN

El Centro Nacional de Genética Médica (CNGM) es una institución que asesora y coordina a nivel Nacional la ejecución de programas de salud. Esta investigación surge a partir de la necesidad de definir la arquitectura para el desarrollo del sistema alasARBOGEN 2.0 para dicha organización, permitiendo agregarle a la versión anterior nuevas funcionalidades, obtenidas como resultado de las necesidades del cliente, además de, integrar el mismo con una versión web. Esta idea se materializa con la definición de una arquitectura regida por el Documento sobre la Arquitectura de Software para los componentes a emplear por el Sistema de Información para la Salud, el cual fue establecido por el Grupo de Arquitectura Ministerio de Salud Pública (MINSAP) - Ministerio de la Informática y las Comunicaciones (MIC).

Para la selección de las herramientas informáticas y tecnologías a utilizar, se tuvo en cuenta que las mismas estuvieran bajo licencia libre. Para lograr un diseño claro y ampliamente aceptado se empleó el patrón arquitectónico Modelo-Vista-Controlador.

La descripción y construcción de la solución propuesta se presenta haciendo uso de las vistas diseñadas por Philippe Kruchten e incorporadas a la metodología RUP, entre las que se encuentran la vista de casos de uso, la vista lógica, la vista de despliegue y la de implementación. Para la evaluación de la propuesta realizada se utilizó el método Taller de Atributos de Calidad (QAW), obteniéndose resultados satisfactorios, pues se cumplieron todos los escenarios definidos para cada atributo de calidad, demostrando la adecuación de la arquitectura propuesta.

PALABRAS CLAVE

alasARBOGEN, Centro Nacional de Genética Médica, Arquitectura de Software.

TABLA DE CONTENIDOS

INTRODUCCIÓN

Las tecnologías de la información y las comunicaciones son elementos fundamentales para la superación y desarrollo de un país. Estos basan su crecimiento en la aplicación y la programación estratégica de las herramientas computacionales, las que han pasado a jugar un rol vital en el desarrollo del sector de la salud, permitiendo reducir de forma antes inimaginable, el tiempo requerido para la gestión de la información y la prestación de servicios.

Uno de los principales retos de la revolución cubana a lo largo de estos 50 años ha sido la atención a las personas que presentan algún defecto de carácter genético en Cuba.

Como parte de los programas de la revolución para fomentar el desarrollo tecnológico, y la mejora de la calidad de vida de esta parte de la población cubana, así como impulsar las investigaciones básicas y aplicadas en el campo de la Genética Médica, surge el 5 de agosto de 2003 el Centro Nacional de Genética Médica (CNGM). El cual conduce el programa nacional para el diagnóstico, manejo y prevención de enfermedades genéticas y defectos congénitos logrando disminuir considerablemente el impacto de estas en la población cubana.

A partir del profundo proceso de transformaciones educacionales y sociales y como parte de los programas de la revolución destinados a la Batalla de Ideas surge la Universidad de las Ciencias Informáticas (UCI), adscrita al Ministerio de la Informática y las Comunicaciones (MIC). En la Universidad se desarrolla de forma simultánea actividades académicas y productivas, permitiendo la formación de un capital humano especializado e innovador para la producción de software, no sólo para Cuba, sino también para el mundo.

La UCI y en especial la facultad 6, de conjunto con el CNGM trabajan en el desarrollo de sistemas que contribuyan a la automatización de muchos de los procesos que en esta entidad se ejecutan. Teniendo en cuenta los estudios que realiza el CNGM con las familias y los resultados de los cruzamientos genéticos, fue desarrollada la primera versión de alasARBOGEN 1.0, aplicación de escritorio que permite obtener una representación gráfica en forma de árbol de una familia, que expone los datos genealógicos de un individuo en forma organizada. La misma es la que se utiliza actualmente, con la limitante de haber sido desarrollada mediante un software propietario.

Como consecuencia del bloqueo económico que impone a Cuba los Estados Unidos se hace casi imposible la adquisición de productos informáticos desarrollados por empresas norteamericanas, además

debe pagarse el costo de la licencia de dicho software por cada copia que se instale en un ordenador diferente, lo que implica altos costos de producción en materia monetaria.

La arquitectura actual del sistema no permite su migración a una plataforma libre, ni la interacción con otros servicios que brinda la Red Nacional de Salud (INFOMED) y que proporcionan información requerida por los genetistas. La aplicación actual no permite almacenar los datos de forma centralizada. Se necesita una nueva estructura que permita efectuar operaciones de consultas, la generación de reportes y análisis estadísticos a los datos desde cualquier centro de genética con conectividad a INFOMED, brindando un flujo de información rápido y seguro.

Teniendo en cuenta los argumentos expuestos anteriormente surge como **problema científico** en la presente investigación *¿Cómo definir la organización estructural del Sistema para la representación de árboles genealógicos "alasARBOGEN" para su nuevo desarrollo en plataforma libre?*

Ante esta interrogante se identifica como **objeto de estudio** *la arquitectura de software para aplicaciones destinadas a la Red Nacional de Salud* y dentro de este se selecciona como **campo de acción** *la arquitectura de software para los sistemas de representación de árboles genealógicos.*

Como **objetivo general** se propone**:** *definir la arquitectura para el desarrollo en plataforma libre de alasARBOGEN 2.0.*

Para dar cumplimiento a este objetivo se definieron las siguientes tareas a cumplir:

1. Estudio de la actual arquitectura del sistema alasARBOGEN 1.0.
2. Estudio de la arquitectura de software establecida para el desarrollo de aplicaciones para el sistema de salud.
3. Análisis de los servicios que brinda INFOMED y el Registro Informatizado de Salud (RIS).
4. Descripción de las principales tendencias del desarrollo de sistemas Web y escritorio en plataforma libre.
5. Descripción de las técnicas y métodos de evaluación de la arquitectura de software.
6. Análisis de los nuevos requerimientos que debe cumplir el sistema.
7. Selección y análisis de las herramientas y tecnologías informáticas a utilizar para el desarrollo de la arquitectura.
8. Definición de la arquitectura para el sistema alasARBOGEN 2.0.
9. Definición y descripción del patrón arquitectónico a emplear.

10. Diseño de la vista de casos de uso, la vista lógica, la vista de procesos, la vista de implementación y la vista de despliegue.
11. Evaluación de la arquitectura definida.

El presente trabajo de diploma se estructura en tres capítulos:

- En el Capítulo 1: **Fundamentación Teórica,** es analizado el concepto de arquitectura de software y otras definiciones estrechamente relacionadas con la misma. Además se identifican los atributos de calidad y los métodos que permiten la evaluación de la arquitectura propuesta.
- En el Capítulo 2: **Definición de la Arquitectura,** se analizan los requisitos que debe cumplir el sistema, se realiza una selección de las herramientas y tecnologías a utilizar. Se definen los principales mecanismos arquitectónicos, se describe la organización del sistema, así como las diferentes vistas de la arquitectura incorporadas a la metodología de desarrollo Proceso Unificado de Desarrollo (RUP).
- En el Capítulo 3: **Evaluación de la Arquitectura Propuesta,** se evalúa la arquitectura propuesta y se analizan los resultados obtenidos.

Capítulo 1: Fundamentación teórica.

INTRODUCCIÓN

Para el desarrollo de todo software, es indispensable una correcta definición su arquitectura, pues la misma brinda una visión clara del producto y permite controlar su desarrollo. La arquitectura de software constituye una guía para el trabajo con el sistema.

En el presente capítulo se abordan los aspectos referentes a los estilos arquitectónicos seleccionados, los patrones, el lenguaje que permitirá la descripción arquitectónica del sistema y el rol que desempeña el arquitecto de software dentro del ciclo de desarrollo del producto, según la metodología de desarrollo RUP, además de los artefactos que son generados por el mismo en cada fase.

El concepto de arquitectura de software varía en dependencia de los disímiles autores y especialistas que lo manejan, así como de las diferentes metodologías de desarrollo de software existentes. A continuación se dan a conocer algunos de los criterios más conocidos, además del ambiente donde será desplegado el sistema.

1.1 DEFINICIÓN DE ARQUITECTURA DE SOFTWARE

Con la evolución de los sistemas informáticos, se fue haciendo necesario organizar los procesos de desarrollo de los mismos. Producto a esto, nuevas metodologías fueron surgiendo dentro de la Ingeniería de Software, y entre ellas la Arquitectura de Software.

En 1969, un año después de la sesión en que se fundara la ingeniería de software, P. I. Sharp expresó comentando las ideas de Dijkstra:

"*Pienso que tenemos algo, aparte de la ingeniería de software: algo de lo que hemos hablado muy poco pero que deberíamos poner sobre el tapete y concentrar la atención en ello. Es la cuestión de la arquitectura de software".*(REYNOSO, 2004)

La arquitectura de software define, de manera abstracta, los componentes que llevan a cabo alguna tarea de computación, sus interfaces y la comunicación entre ellos.

El concepto de arquitectura puede verse de diferentes formas o puntos de vista El Instituto de Ingenieros Eléctricos y Electrónicos (**IEEE) 1471** la define como "el nivel conceptual más alto de un sistema en su ambiente." (LEN BASS, 2004)

Pressman plantea al respecto que la arquitectura: "… es una descripción de los subsistemas y los componentes de un sistema informático y las relaciones entre ellos (...)".(PRESSMAN, 2002)

En el libro **Arquitectura de Software en la Práctica** (**Software Architecture in Practice) 2nd edición** se define como "La estructura de estructuras de un sistema, la cual abarca componentes de software, propiedades externas visibles de estos componentes y sus relaciones". (BASS, 2003)

El **IEEE Std 1471-2000** plantea que la arquitectura "es la organización fundamental de un sistema incorporada en sus componentes, en sus relaciones mutuas y en el entorno, y los principios que guían su diseño y evolución." (IEEE Std 1471-2000, 2000). Este concepto es asumido a nivel internacional como oficial. Después de analizados los diferentes conceptos antes expuestos se asume que para la realización de esta investigación la definición de la arquitectura propuesta por la **Standart IEEE 1471-2000.**

1.2 PATRONES Y ESTILOS ARQUITECTÓNICOS

¿Patrones o estilos arquitectónicos?

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 estilos y patrones arquitectónicos se puede comenzar mencionando que 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.

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". (SHAW, MARY, 1996)

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" (ALLEN, 1997), 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". (ALLEN, 1997).

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” (KICILLOF, 2004).

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. 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.

1.2.1 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. Para ello se aplican patrones de soluciones ya probadas, conocidas como estilos arquitectónicos.

Un estilo arquitectónico define una familia de sistemas de software en términos de su organización estructural. Estos representan los componentes y las relaciones entre ellos con las restricciones de su aplicación y las asociaciones y reglas de diseño para su construcción.

Las primeras definiciones explícitas de estilo fueron propuestas por Dewayne Perry de AT&T Bell Laboratorios 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.” (KICILLOF, 2004)

Posteriormente Mary Shaw y Paul Clements conceptualizan los estilos arquitectónicos como un “conjunto de reglas de diseño que identifican 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”. (LEN BASS, 2003)

Cada estilo describe una categoría del sistema que contiene:

- ✓ Un conjunto de componentes que realizan una función requerida por el sistema.
- ✓ Un conjunto de conectores que posibilitan la comunicación, la coordinación y la cooperación entre los componentes.
- ✓ Restricciones que definen cómo se pueden integrar los componentes que forman el sistema.
- ✓ Modelos semánticos que permiten al diseñador entender las propiedades globales de un sistema para analizar las propiedades conocidas de sus partes constituyentes.

En la presente investigación se adopta la clasificación propuesta por "Shaw y Garlan" (SHAW, MARY Y GARLAN, DAVID, 1996.), quienes define las siguientes clases de estilos:

Estilo de Flujo de datos:

- Tuberías y Filtros.

Estilo centrado en datos:

- Arquitecturas de Pizarra o Repositorio.

Estilo de llamada y retorno:

- Modelo-Vista-Controlador.
- Arquitectura en Capas.
- Arquitectura basada en Componentes.
- Arquitectura basada en Objetos.

Estilo Peer – To – Peer:

- Arquitecturas basadas en Eventos.
- Arquitecturas Orientadas a Servicios (SOA).
- Arquitecturas basadas en recursos.

Estilo heterogéneo:

- Sistemas de control de procesos.
- Arquitecturas basadas en atributos.

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

Estilo Centrado en Datos

Este estilo enfatiza la integrabilidad de los datos. Es utilizado preferentemente en sistemas que se basan en acceso, actualización de datos y en estructuras de almacenamiento de datos.

Estilo de Llamada y Retorno

Este estilo enfatiza la modificabilidad y la escalabilidad. Es el más generalizado en sistemas en gran escala. Como parte de este estilo están las arquitecturas de programa principal y subrutina, los sistemas basados en llamadas a procedimientos remotos, los sistemas orientados a objetos y los sistemas jerárquicos en capas.

Tras analizar los diferentes estilos arquitectónicos antes expuestos se asume que para la realización de esta investigación se utilizará el estilo de llamada y retorno.

1.2.2 PATRONES

Los patrones de software surgieron a partir del año 1977, por Christopher Alexander, "son soluciones reutilizables a los problemas que ocurren durante el desarrollo de un sistema de software o aplicación" (ING. YAMILA VIGIL REGALADO, 2008). Estos proporcionan un proceso consistente o diseño que uno o más desarrolladores pueden utilizar para alcanzar sus objetivos. También proporcionan una arquitectura uniforme que permite una fácil expansión, mantenimiento y modificación de una aplicación.
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. En dependencia del problema que solucionan, estos pueden ser clasificados en: patrones de arquitectura, patrones de análisis, patrones de diseño, entre otros.
A continuación se listan los patrones más conocidos: diseño, arquitectura, organización o proceso, código, interfaz de usuario, prueba, análisis, entre otros.

Patrón de diseño: Son soluciones simples y elegantes a problemas específicos y comunes del diseño orientado a objetos. Conocido también como una técnica para flexibilizar el código haciéndolo satisfacer ciertos criterios. Estos centran su desarrollo en aspectos organizativos, sociales y económicos de un sistema.(GRACIA, 2005)

Patrón de organización o proceso: Es el elemento fundamental para el desarrollo de una nueva forma de pensar. Su objetivo principal consiste en la configuración entre las relaciones de los componentes del sistema, determinando sus características esenciales.(PETER, 2005)

Patrón de análisis: Son un conjunto de clases y relaciones entre ellas, que tienen algún sentido en el contexto de una aplicación. Representan una estructura que puede ser válida para otras aplicaciones. Estos se centran en los aspectos organizativos, sociales y económicos de un sistema, ya que estos aspectos son fundamentales para el análisis de las necesidades, la aceptación y la usabilidad del sistema final.(CRAIG LARMAN, 2003)

Patrón Arquitectónico: Es un patrón de alto nivel que fija la arquitectura global de una aplicación. Estos "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" (BUSCHMANN, 1996). Debido a su utilidad en la actualidad y la importancia del mismo para el desarrollo de múltiples aplicaciones a continuación se exponen algunos de los patrones arquitectónicos más conocidos y utilizados:

Arquitecturas en Capas:

Los sistemas o arquitecturas en capas constituyen uno de los patrones más utilizados en la actualidad. En Garlan y Shaw definen el patrón capas como "una organización jerárquica tal que cada capa proporciona servicios a la capa inmediatamente superior y se sirve de las prestaciones que le brinda la inmediatamente inferior".(KICILLOF, 2004)

Una aplicación básica de este patrón 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:** Conocida como interfaz gráfica del usuario en el sistema, es la encargada de presentar la interfaz del sistema al usuario, comunicar la información y capturar la información del mismo. 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, y con la capa de datos, para solicitar al gestor de base de datos para almacenar o recuperar datos de él.
- ✓ **Capa de datos:** Es donde reside la información persistente del sistema. Está formada por uno o más gestores de bases de datos que realizan todo el almacenamiento de datos, reciben solicitudes de almacenamiento o recuperación de información desde la capa de negocio.

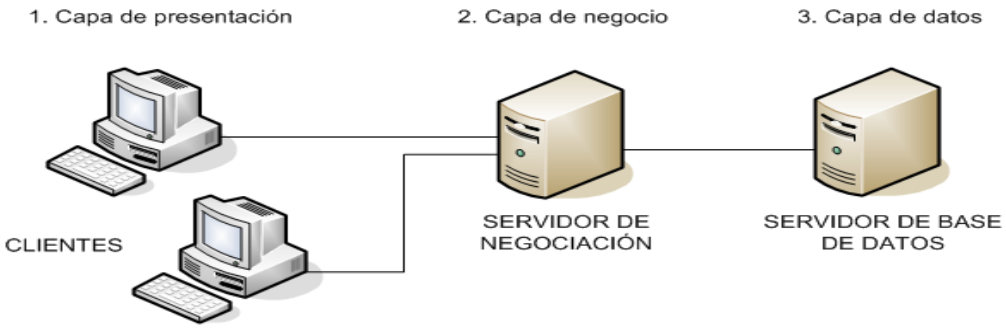


Figura 1. Representación del patrón Arquitectura en Capas. Ejemplo de tres capas y tres niveles.

Entre las ventajas de este patrón se encuentran:

- ✓ El patrón 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.
- ✓ Admite optimizaciones y refinamientos.
- ✓ Proporciona amplia reutilización.

Entre las desventajas:

- ✓ Muchos problemas no admiten un buen mapeo en una estructura jerárquica.
- ✓ Dificultad de diseñar correctamente la granularidad de las capas.
- ✓ Es ineficiente cuando se utiliza un número de niveles excesivos.
- ✓ Si hay pocos niveles se obtiene un diseño poco organizado. Si por el contrario estos son excesivos el sistema se vuelve muy complejo.

Modelo - Vista - Controlador

Modelo Vista Controlador (MVC) es un patrón de diseño de arquitectura de software que separa los datos de una aplicación, la interfaz de usuario, y la lógica de control en tres componentes distintos. Usado principalmente en aplicaciones que manejen gran cantidad de datos y transacciones complejas donde se requiere una mejor separación de conceptos para que el desarrollo esté estructurado de una mejor manera, facilitando la programación en diferentes capas de manera paralela e independiente.

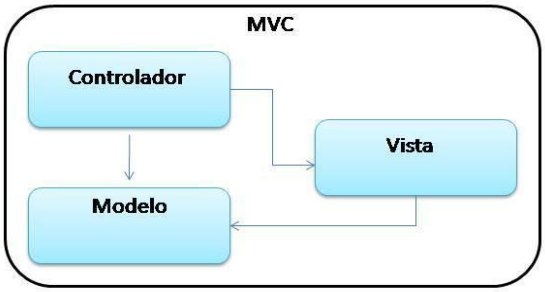


Figura 2. Representación del patrón MVC.

- **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**: Es el componente encargado de manejar y responder las solicitudes del usuario, procesando la información necesaria y modificando el modelo en caso de ser necesario.

Los beneficios de usar el MVC son:

1. Facilita agregar nuevos tipos de datos según sea requerido por la aplicación ya que son independientes del funcionamiento de las otras capas.
2. Facilita el mantenimiento en casos de errores.
3. Permite el escalamiento de la aplicación en caso de ser requerido.
4. Su modularidad.
5. Posee un diseño claro y ampliamente aceptado.
6. Su extensibilidad.
7. Es mucho más sencillo agregar múltiples representaciones de los mismos datos e información.

Las desventajas que ofrece el MVC son:

1. La separación de conceptos en capas agrega complejidad al sistema.
2. La cantidad de archivos a mantener y desarrollar se incrementa considerablemente.

Después de analizados los patrones arquitectónicos antes expuestos se selecciona el patrón arquitectónico Modelo-Vista-Controlador pues este permite que desde la vista pueda accederse a las clases del modelo, lo cual es muy ventajoso en aplicaciones en las que se hace un uso intensivo de componentes visuales.

1.3 LENGUAJES DE DESCRIPCIÓN DE LA ARQUITECTURA

Un Lenguaje de Descripción de la Arquitectura (ADL por sus siglas en ingles), es un lenguaje descriptivo modelado que proporciona características para el modelaje conceptual de un sistema de software, distintiva de la implementación del sistema. Se emplean mayoritariamente para especificar y describir un estilo arquitectónico.

Medvidovic "define un ADL a través de las características de sus componentes, conectores, configuraciones arquitectónicas y soporte de herramientas. Estos no son aspectos definitorios del concepto de ADL, sino de criterios para la evaluación de los ADLs existentes." (CARLOS REYNOSO, 2004). A continuación se definen cada uno de estos elementos:

- ✓ **Componentes**: Representan los elementos computacionales primarios de un sistema. Los componentes exponen varias interfaces en la mayoría de los ADLs. Dichas interfaces definen puntos de interacción entre un componente y su entorno.
- ✓ **Conectores:** Representan interacciones entre componentes. Poseen una especie de interfaz que define los roles entre los componentes participantes en la interacción.
- ✓ **Configuraciones o sistemas:** Se constituyen como grafos de componentes y conectores. Los sistemas también pueden ser jerárquicos: componentes y conectores pueden subsumir la representación de lo que en realidad son complejos subsistemas.
- ✓ **Restricciones:** Representan condiciones de diseño que deben acatarse incluso en el caso que el sistema evolucione en el tiempo.
- ✓ **Propiedades:** Representan información semántica sobre un sistema más allá de su estructura.
- ✓ **Estilos:** Representan familias de sistemas, un vocabulario de tipos de elementos de diseño y de reglas para componerlos. Algunos estilos prescriben un Framework, un estándar de integración de componentes, patrones arquitectónicos o como se lo quiera llamar.
- ✓ **Evolución:** Los ADLs deberían soportar procesos de evolución permitiendo derivar subtipos a partir de los componentes e implementando refinamiento de sus rasgos.
- ✓ **Propiedades no funcionales:** La especificación de estas propiedades es necesaria para simular la conducta de runtime, analizar la conducta de los componentes, imponer restricciones, mapear implementaciones sobre procesadores determinados, etcétera.

Dentro de los principales ADL se encuentran **ADML, Aesop, CHAM, Jacal**.

UML 2.0, no es un lenguaje de descripción de la arquitectura propiamente dicho, pero cumple con casi todas las características para serlo. Este se comporta mejor que la mayoría de los ADLs conocidos y es

de fácil comprensión por parte de los desarrolladores, algo que le da una buena ventaja sobre el resto de los ADLs.
Grady Booch plantea que "UML es un lenguaje gráfico para visualizar, especificar, construir y documentar los artefactos de un software de sistema intensivo. El UML soporta múltiples vistas de un sistema tanto estructurales como de comportamiento" (GRADY BOOCH, DAVID GARLAN, CRIS KOBRYN, VICTORIA STAVRIDOU, 1999).

UML ofrece la ventaja de ser un lenguaje visual utilizado para el modelado de una amplia gama de sistemas de software. Además, es compatible con muchas herramientas. En términos de arquitectura, UML nos permite modelar el sistema de cinco puntos de vista diferentes. Centrándose cada una de ellas en un aspecto en particular del sistema, y un conjunto de diagramas UML se propone el modelo de los aspectos estáticos y dinámicos de la vista. Esta abundancia de capacidades de modelado muestra el poder de UML para el modelado de arquitectura de software. El mismo posibilita además, la reutilización de código, el ahorro de tiempo en el desarrollo del software y la comunicación entre programadores es mucho más fácil.

Tras el análisis realizado de cada uno de los conceptos referentes a los ADL se decide utilizar como lenguaje de descripción de la arquitectura el UML, puesto que los lenguajes anteriormente estudiados son difíciles de entender para los desarrolladores, requieren una extensa capacitación del personal y no son amigables para presentar la arquitectura a personas ajenas a la construcción de software. UML tiene la ventaja de que es soportado por herramientas CASE y permite que se represente de manera semiformal la estructura general del sistema. RUP utiliza UML para la representación de sus diagramas, las 4 + 1 vistas arquitectónicas constituyen diagramas y RUP las utiliza para describir la arquitectura de software. Por tanto esto nos garantiza la utilización de UML como ADL.

1.4 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 la que se han unificado diferentes técnicas de producción de software. Esta se define como un conjunto de actividades necesarias para transformar los requisitos del usuario en un sistema de software". (GRADY BOOCH, IVAR JACOBSON, JAMES RUMBAUGH., 2000)

Dicho proceso presenta tres características fundamentales: Dirigido por casos de uso, centrado en la arquitectura e Iterativo e incremental. RUP se divide en cuatro fases, dentro de las cuales se definen un conjunto de objetivos específicos para cada fase. Cada una de estas fases es desarrollada mediante el ciclo de iteraciones, la cual consiste en reproducir el ciclo de vida en cascada a menor escala. (GRADY BOOCH, IVAR JACOBSON, JAMES RUMBAUGH., 2000). La figura representa lo anteriormente explicado:

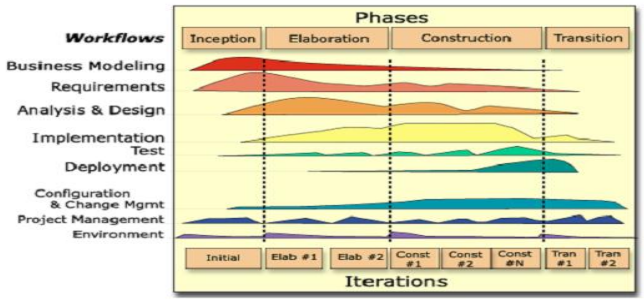


Figura 3. Representación del Flujo de Trabajo de RUP.

Para el Proceso Unificado de Desarrollo, la arquitectura es un elemento primordial. Este abarca decisiones importantes sobre la organización del sistema, los elementos estructurales que corresponden al mismo, sus interfaces, la composición de los elementos estructurales y el comportamiento en subsistemas progresivamente más grandes y como guía de esta organización, los estilos de la arquitectura.

La arquitectura en RUP se desarrolla mediante iteraciones, principalmente en la etapa de elaboración, donde el hito es la línea base de la arquitectura que no es más que donde se encuentran un conjunto de modelos que representan los casos de uso más importantes y sus realizaciones desde el punto de vista de la arquitectura. En RUP la arquitectura 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. (GRADY BOOCH, IVAR JACOBSON, JAMES RUMBAUGH., 2000)

1.5 MÉTODOS DE EVALUACIÓN DE LA ARQUITECTURA.

El cumplimiento de los requisitos de calidad es una tarea fundamental para lograr el éxito de cualquier producto comercial, incluido el software. La definición de la arquitectura del software es una de las primeras y más importantes decisiones a tomar en el proceso de desarrollo, con una gran influencia sobre la calidad final del producto o sistema, dicho resultado no tendría efecto si no se tuviera en cuenta la evaluación de la arquitectura.

¿Qué es una Evaluación?

La evaluación es un "estudio de factibilidad que pretende detectar posibles riesgos, así como buscar recomendaciones para contenerlos" (GUSTAVO ANDRÉS BREY, 2005). Dicha evaluación es realizada antes de la implementación de la solución.

Objetivos de Evaluar una Arquitectura de Software

El objetivo de realizar evaluaciones a la arquitectura, es para analizar e identificar riesgos potenciales en su estructura y sus propiedades, que puedan afectar al software resultante, verificar que los requerimientos no funcionales estén presentes en la arquitectura, así como determinar en qué grado se satisfacen los atributos de calidad.

La evaluación de una arquitectura de software es una tarea no trivial, puesto que se pretende medir propiedades del sistema en base a especificaciones abstractas, como por ejemplo, los diseños arquitectónicos. Por ello, la intención es más bien la evaluación del potencial de la arquitectura diseñada para alcanzar los atributos de calidad requeridos. Las mediciones que se realizan sobre una arquitectura de software pueden tener distintos objetivos, dependiendo de la situación en la que se encuentre el arquitecto y la aplicabilidad de las técnicas que emplea.

A continuación se presentan un conjunto de métodos que son utilizados para evaluar la arquitectura de software:

- ✓ Método de Análisis de Acuerdos de Arquitectura de Software (ATAM)
- ✓ Taller de Atributos de Calidad(QAW)
- ✓ Diseño Determinado por Atributos(ADD)
- ✓ Revisiones Activas para los Diseños Intermedios (ARID)

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

El Método de Análisis de Acuerdos de Arquitectura de Software está inspirado en tres áreas distintas: los estilos arquitectónicos, el análisis de atributos de calidad y el método de evaluación Método de Análisis de Arquitectura de Software (SAAM). El nombre del método ATAM surge del hecho de que revela la forma en que una arquitectura específica satisface ciertos atributos de calidad, y provee una visión de cómo los atributos de calidad interactúan con otros.

El método se centra en la identificación de los estilos arquitectónicos o enfoques arquitectónicos utilizados. Estos elementos representan los medios empleados por la arquitectura para alcanzar los atributos de calidad, así como también permiten describir la forma en la que el sistema puede crecer, responder a cambios, e integrarse con otros sistemas.

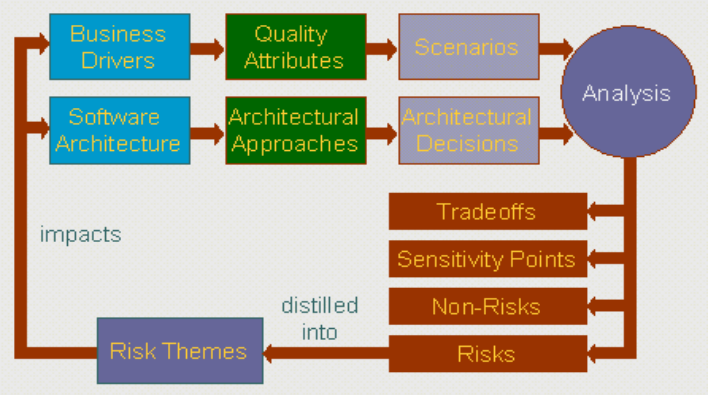


Figura 4.Flujo conceptual del método ATAM

El método de evaluación ATAM comprende nueve pasos, agrupados en cuatro fases (Presentación, Investigación y Análisis, Pruebas y Reporte o Informes).

Taller de Atributos de Calidad (QAW):

QAW fue desarrollado para complementar el Método de Análisis de Compensación de Arquitectura del Instituto de Ingeniería de Software (SEI) (ATAM). Su objetivo consiste en permitir identificar, recolectar y organizar los requisitos relacionados con los atributos de calidad del software que dirigen el proceso de diseño de la arquitectura, por lo que su aplicabilidad es previa al diseño arquitectónico. Tiene como

resultado una lista de factores que van a dirigir el diseño de la arquitectura y una lista de escenarios que servirán para evaluar los atributos de calidad. (MARIO R. BARBACCI, 2003)

Este método proporciona un conjunto de ventajas, las cuales favorecen al desarrollo del software:

- ✓ QAW le ayuda a determinar la calidad para el sistema antes de que sea desarrollado, algo que es crucial para el éxito de sistema y para la satisfacción de sus interesados (stakeholders).
- ✓ Un lugar estructurado y eficiente para comunicarse con los stakeholders, apoyar el análisis y probar el sistema.
- ✓ Refinan el sistema y las exigencias del software.
- ✓ Proporciona un foro para una amplia variedad de stakeholders permitiendo que exista un intercambio de conocimientos.

Diseño Determinado por Atributos (ADD):

El ADD es un método desarrollado por el Instituto de Ingeniería del Software (SEI) para el diseño de arquitectura, que se basa en los atributos de calidad que se quiere que posea el sistema, más que por la funcionalidad de la aplicación.
Una vez que se han capturado escenarios y se han priorizado, se procede a realizar el diseño de la arquitectura siguiendo un enfoque iterativo de descomposición del sistema en componentes cada vez más pequeños. Se van tomando progresivamente escenarios y se van realizando elecciones de diseño (usando soluciones conceptuales llamadas “patrones” y “tácticas”) que permitan satisfacer los requerimientos descritos por los escenarios. Este apoya la construcción de la arquitectura basada en el proceso de diseño en los atributos de calidad que deben ser cumplidos. La salida es una arquitectura conceptual, la cual ayuda a alcanzar los atributos de calidad deseados y proporciona la estructura necesaria para dotar al sistema de la funcionalidad requerida.

Revisiones Activas para los Diseños Intermedios (ARID):

ARID es un método de bajo costo y gran beneficio, este provee una mejor factibilidad de la arquitectura. Es conveniente para realizar la evaluación de diseños parciales en las etapas tempranas del desarrollo. Este es utilizado para la evaluación de diseños detallados de unidades del software como los

componentes o módulos. Las preguntas giran en torno a la calidad y completitud de la documentación y la suficiencia, el ajuste y la conveniencia de los servicios que provee el diseño propuesto.

El método ARID se basa en ensamblar el diseño de los stakeholders para articular los escenarios de usos importantes o significativos, y probar el diseño para ver si satisface los escenarios. Como resultado de la aplicación de dicho procedimiento se obtiene un diseño de alta fidelidad acompañado de una alta familiarización con el diseño de los stakeholders. Este método consta de 9 pasos agrupados en dos fases (Actividades Previas y Evaluación).

Después de analizados los diferentes métodos de evaluación antes expuestos se determinó que para la realización de esta investigación el método que se utilizará para crear una arquitectura estable es el Taller de Atributos de Calidad (QAW), porque el mismo garantiza la calidad del sistema antes de que sea desarrollado, además, este complementa al método ATAM que describe la forma en la que el sistema puede crecer, responder a cambios, e integrarse con otros sistemas.

1.6 ARQUITECTURA DEL SISTEMA NACIONAL DE SALUD

El desarrollo de aplicaciones destinadas a INFOMED, está regido por un grupo de normas y tecnologías definidas por el grupo de arquitectura MINSAP-MIC. Estas normas están dirigidas a garantizar la continuidad y sostenibilidad de los productos que se obtienen y fundamentalmente a contribuir con su integración. A continuación se relacionan algunas:

- Funcionar sobre servidor Linux, distribución Debian (Sarge).
- Funcionar sobre Servidor Web Apache 2.0.
- Consumir o brindar servicios web para interactuar con otros sistemas.

Los sistemas desarrollados deben permitir ser desplegados sobre los servidores disponibles de INFOMED.

Las aplicaciones deben interactuar entre sí consumiendo y/o brindando servicios web según el esquema de seguridad desarrollado por Softel.

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

1.6.1 COMPONENTES DESARROLLADOS POR SISALUD

En la actualidad SISalud presenta varios componentes desplegados que pertenecen al Registro Informatizado de Salud (RIS). Entre ellos podemos mentar el Componente de Seguridad (SAAA), el Registro del Ciudadano (RC), Registro de Unidades de Salud (RUS), Registro de Equipos Médicos (REM), Registro del Personal de la Salud (PRS), Registro de Localidad, entre otros.
Es de interés en esta investigación resaltar el SAAA y el RC.
El **SAAA,** está basado en el modelo de Autenticación, Autorización y Auditoría. La autenticación debe ser la primera acción del usuario en el sistema, para lo cual el usuario cuenta con un nombre de usuario y una contraseña de su único conocimiento. Si la autenticación no es satisfactoria se reporta un error, de lo contrario, se autoriza su acceso, se crea un certificado digital y se retornan todos los datos y permisos del mismo.
El **RC** contiene los datos personales de cualquier ciudadano que trabaje en la salud y/o reciba sus servicios.

CONCLUSIONES

En este capítulo se conceptualizan, describen y analizan elementos relativos a la arquitectura de software, los estilos y patrones arquitectónicos, aspectos a tener en cuenta en el transcurso de este trabajo.
A partir de lo analizado anteriormente, es posible afirmar que en la arquitectura del sistema alasARBOGEN 2.0, se empleará el patrón arquitectónico Modelo Vista Controlador. Además se exponen las pautas de desarrollo para las aplicaciones de la salud propuestas por el grupo de arquitectura MINSAP-MIC, con las cuales se logró posteriormente 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: Definición de la Arquitectura

INTRODUCCIÓN

Para el desarrollo de cualquier sistema de software deben tenerse en cuenta las principales características y funcionalidades que el mismo debe cumplir. Estas funcionalidades dan lugar a los requisitos funcionales y no funcionales por los que debe regirse el desarrollo e influyen en la definición de la arquitectura. Un sistema de software se caracteriza a partir de la forma en que se encuentran organizados sus componentes, así como las herramientas y tecnologías informáticas necesarias para su desarrollarlo y posterior explotación.

En el presente capítulo se describen brevemente los requisitos de software y a partir del análisis de los mismos se selecciona el patrón arquitectónico a utilizar. Además se incluye un análisis del estudio realizado para la selección de las herramientas informáticas y tecnologías propuestas para el desarrollo del sistema.

2.1 ANÁLISIS DE LOS REQUISITOS

El sistema está constituido por dos aplicaciones: aplicación web y aplicación de escritorio. Estas gestionan de forma independiente sus funcionalidades, pero comparten la misma base de datos. El objetivo es poder realizar a través de la aplicación de escritorio los árboles genealógicos correspondientes a cada familia que se desee estudiar, y ejecutar a través de la web, cuando se necesite, consultas sobre los datos almacenados, sin necesidad de disponer de la aplicación de escritorio.

El sistema debe permitir el acceso rápido y seguro desde cualquier centro de genética del país, sin las dificultades que supone la instalación y mantenimiento de la aplicación en cada una de las estaciones de trabajo.

Teniendo en cuenta que los servicios de conexión a INFOMED podrían verse interrumpidos, la aplicación de escritorio permitirá guardar los datos de forma local y estos podrán enviarse luego de restablecida la conexión, haciendo uso de cualquiera de las dos aplicaciones.

En el siguiente modelo de dominio (ver Figura 6) se describe el funcionamiento de la aplicación alasARBOGEN1.0, donde se evidencia que no existe ninguna integración del mismo con los componentes de INFOMED. Esta falta de conectividad trae consigo que el proceso de búsqueda de datos de los pacientes sea muy complejo, e incluso en ocasiones haya que introducir o actualizar los datos de forma

manual en la mayoría de los estudios genéticos, además, no permite la persistencia de los datos gestionados por una base de datos.

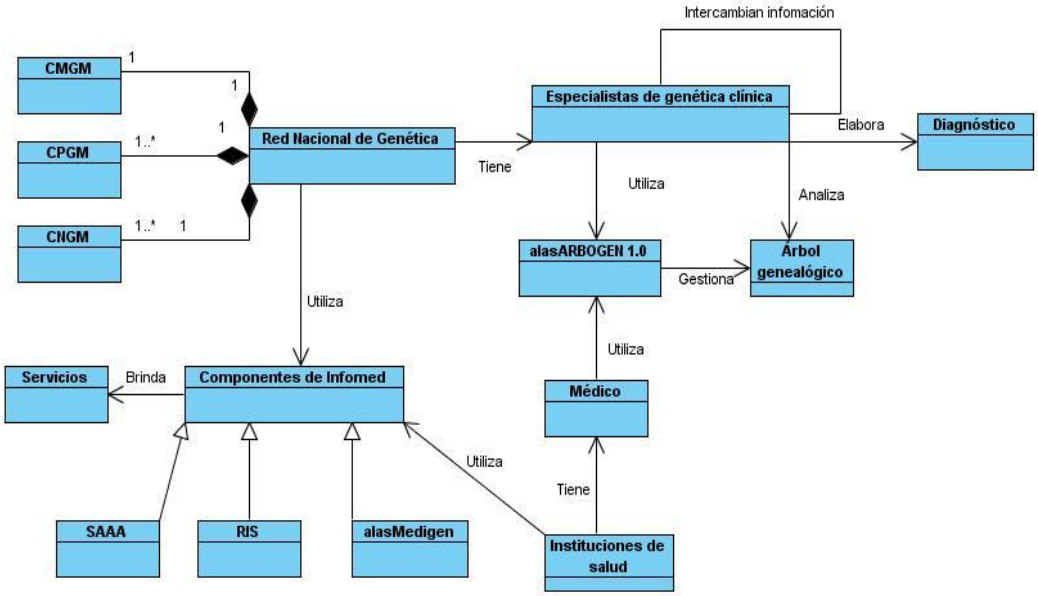


Figura 5. Modelo del Dominio de alasARBOGEN 1.0.

alasARBOGEN 2.0 constituye un sistema informático para la salud en Cuba, por tanto, debe cumplir con las normas establecidas por el Grupo de Arquitectura MINSAP-MIC expuestas en el capítulo anterior.

Dentro de los requerimientos arquitectónicos definidos por el Grupo de Arquitectura MINSAP-MIC, para las aplicaciones informáticas destinadas a la salud se insertan además un grupo de requisitos no funcionales que el sistema debe cumplir. A continuación son definidos cada uno de ellos.

Apariencia e interfaz externa

Las imágenes y colores deben estar identificados con el negocio del sistema, proporcionando un ambiente agradable para el usuario. La interfaz externa debe estar diseñada para verse en cualquier resolución igual o superior a 1024x768.

Usabilidad

El acceso al sistema se realizará de forma fácil y rápida. El mismo contará con un menú que satisfaga las necesidades de los usuarios. Este puede ser usado por cualquier persona que posea conocimientos básicos en el manejo de una computadora y el ambiente de un navegador.

Rendimiento

La velocidad de procesamiento del sistema debe ser rápida, al igual que la capacidad de respuesta.

Soporte

Las necesidades de los usuarios deben quedar satisfechas después de la puesta en explotación del sistema, a través de actualizaciones y mejoras. Para lograr esto se elaboraran una serie de manuales y videos tutoriales, así como la atención en línea.

Seguridad

La información correspondiente a la asociación de los individuos y el parentesco puede ser considerada como confidencial, según las particularidades de las familias que se estudian a través de la aplicación. (Esta proporciona información de los familiares y sus descendencias). Debido a que no pueden efectuarse cambios sobre la información por personas no autorizadas, debe diseñarse un mecanismo de seguridad estricto que regule el acceso al sistema. Esta debe manejarse de manera independiente a cualquier sistema, y sus implicados deben estar bien identificados y comprometidos con el resguardo de la información.

La seguridad será manejada por el SAAA, a través de servicios web. Además, el sistema debe tener un mecanismo propio para gestionar la seguridad a través de niveles de acceso establecidos por los administradores del sistema.

Software

Se requiere para el funcionamiento del sistema disponer de un servidor con el sistema operativo Linux, Apache 2.0, Java Web Start, la Máquina Virtual de Java 5.0 y PostgreSQL o versiones superiores. Los usuarios del sistema deberán contar con un navegador Internet Explorer 5.5, Mozilla Firefox 2.0, o el Opera 8.5, todos ellos en la versión mencionada o una superior.

Hardware

Para el desarrollo y ejecución del sistema se precisó que los requisitos estuvieran en función de los requerimientos que propone J2EE para el desarrollo de aplicaciones en Java.

Para el servidor de aplicaciones:

- Microprocesador Pentium III a 2.8 GHz o superior.
- 1GB de RAM o superior.
- 300 MB de espacio en disco duro para el despliegue de la aplicación web y la salva de la aplicación de escritorio.

Para el cliente:

- Conexión al servidor a través de un modem o tarjeta de red.
- Procesador Pentium III o superior.

- 256 MB RAM como mínimo (Recomendado 512 MB RAM o superior).
- 500 MB de espacio libre en disco (Recomendado 800 MB de espacio libre en disco).

Se hace necesario contar con una impresora para la impresión de los reportes.

Disponibilidad

El sistema debe estar disponible a tiempo completo, y recuperarse rápidamente ante cualquier tipo de fallo. Deben ser creadas copias de respaldo de forma periódica, de manera tal que el sistema pueda restaurarse en caso de fallo crítico o pérdida de información. El sistema debe ser capaz de funcionar, independientemente de que los servicios de los diferentes componentes de SISalud no estén disponibles.

Requisitos legales

Las tecnologías y herramientas que se utilicen para desarrollar el sistema deben estar bajo la licencia de software libre.

Persistencia

La información generada por el sistema debe ser almacenada en bases de datos permanentemente con el objetivo de poder generar reportes y realizar estudios posteriores. Teniendo en cuenta además, que en la aplicación de escritorio la información persistirá en ficheros, en el caso de que la base de datos no se encuentre disponible en un momento determinado.

Portabilidad.

El sistema debe permitir ejecutarse en cualquiera de los sistemas operativos más usados en la actualidad (Windows o Linux cualquiera de sus versiones).

Restricciones de diseño e implementación

El despliegue debe realizarse a través de una aplicación web, para lo que se utilizará la tecnología Java Web Start.

La aplicación debe posibilitar trabajar en un ambiente conectado-desconectado.

2.2 TECNOLOGÍAS Y HERRAMIENTAS

Entre las buenas prácticas para la arquitectura de software está, el brindar soporte con el desarrollo del sistema a través del uso herramientas y tecnologías que faciliten el proceso de construcción. Para ello debe realizarse un buen estudio de aquellas que por sus características se ajusten más a las necesidades de desarrollo que se tienen, teniendo en cuenta sus potencialidades y debilidades.

Se debe analizar además las condiciones o posible repercusión que tendría su uso. Cuba, debido a las restricciones que impone el bloqueo económico de los Estados Unidos tiene como premisa la utilización de herramientas clasificadas como software libre.

2.2.1 CONTROL DE VERSIONES

Se denomina control de versiones a la gestión de los cambios que se ejecutan sobre los elementos que componen un producto o la configuración del mismo. Un sistema de control de versiones es un software que gestiona el acceso a los elementos de un producto y el historial de cambios sucesivos sobre el mismo (ROMMEL, 2009). Es muy útil para guardar cualquier fichero que cambie constantemente, y sobre el cual trabajan múltiples personas, como es el código fuente de un programa.
De los sistemas de control de versiones de Código Abierto existentes, los más utilizados son el Sistema de Versiones Concurrentes (CVS) y el Subversion (SVN).

Sistema de Versiones Concurrentes (CVS)
El CVS es muy popular en el mundo del software libre, y sus desarrolladores difunden el programa bajo la licencia GPL. Es una aplicación informática que utiliza una arquitectura cliente – servidor. (STURM, 2008)
En el servidor se guarda(n) la(s) versión(es) del proyecto y su historial. Los clientes descargan del servidor una copia completa del proyecto, lo que les permite trabajar sobre ella y posteriormente subir los cambios, actualizándose automáticamente los archivos y el registro de cambios.
Originalmente, el servidor utilizaba un sistema operativo similar a Unix, aunque en la actualidad existen versiones de CVS en otros sistemas operativos, incluido Windows. Los clientes de CVS funcionan en cualquier sistema operativo.
CVS presenta como limitación el hecho de que no se permite que los archivos almacenados en el repositorio sean renombrados, estos deben agregarse con otro nombre y posteriormente eliminarse.

Subversión 1.4 (SVN) surge con la necesidad de sustituir y mejorar al popular CVS. SVN mantiene las ideas fundamentales de su predecesor, pero suple sus carencias y suprime sus errores.
Las principales características de SVN y sus mejoras frente a CVS son:

- ✓ Mantiene versiones no sólo de archivos, sino también de directorios y de los metadatos asociados a los mismos.

- ✓ Además de los cambios en el contenido de los documentos, se mantiene la historia de todas las operaciones de cada elemento, incluyendo la copia, cambio de directorio o de nombre.
- ✓ Atomicidad de las actualizaciones. Una lista de cambios constituye una única transacción o actualización del repositorio. Esta característica minimiza el riesgo de que aparezcan inconsistencias entre distintas partes del repositorio.
- ✓ Posibilidad de elegir el protocolo de red. Además de un protocolo propio (svn), puede trabajar sobre HTTP o sobre HTTPS. La capacidad de funcionar con un protocolo tan universal como el http simplifica la implantación (cualquier infraestructura de red actual soporta dicho protocolo) y universaliza las posibilidades de acceso (si se quiere, puede utilizarse a través de Internet).
- ✓ Soporte tanto de ficheros de texto como de binarios.
- ✓ Mejor uso del ancho de banda, ya que en las transacciones se transmiten sólo las diferencias y no los archivos completos.
- ✓ Mayor eficiencia en la creación de ramas y etiquetas que en CVS.
- ✓ Permite selectivamente el bloqueo de ficheros.

Después de analizadas las características de ambos sistemas de control de versiones se selecciona al Subversión 1.4 como una de las herramientas de soporte para el desarrollo de alasARBOGEN 2.0.

TortoiseSVN "es un cliente Subversión, implementado como una extensión al Shell de Windows. Es software libre bajo la licencia GNU GPL. Maneja ficheros y directorios a lo largo del tiempo. TortoiseSVN muestra el estado de una carpeta y fichero versionado a través de iconos sobre impresionados, de esta forma se puede ver en qué estado se encuentra su copia de trabajo. Permite acceder a todos los comandos de Subversión, disponibles desde el menú contextual del explorador. TortoiseSVN" (TORTOISESVNTEAM, 2004-2009).

2.2.3 HERRAMIENTAS CASE

Las herramientas **C**omputer **A**ided **S**oftware **E**ngineering (CASE) son aplicaciones informáticas destinadas a incrementar la productividad en el desarrollo de software, reduciendo el coste de las mismas en tiempo y en dinero. Los sistemas CASE a menudo se utilizan como apoyo al método.
De acuerdo con Kendall la Ingeniería de Sistemas Asistida por Ordenador(CASE) "es la aplicación de tecnología informática a las actividades, las técnicas y las metodologías propias de desarrollo, su objetivo

es acelerar el proceso para el que han sido diseñadas, en el caso de CASE para automatizar o apoyar una o más fases del ciclo de vida del desarrollo de sistemas." (KENDALL, 2008)

Algunas de las herramientas CASE más utilizadas son Platinun Erwin, EasyCASE, Oracle Designer, System Architect, Visual Paradigm, BOUML, Enterprise Arquitect.

De ellas son de uso libre ArgoUML, Visual Paradigm y BOUML.

"ArgoUML es una aplicación de diagramado UML publicada bajo la licencia BSD de Código Abierto e implementada en lenguaje Java, por lo que está disponible en cualquier plataforma soportada por Java. Posee herramientas de ingeniería inversa, se encuentra disponible en 10 lenguajes y exporta los diagramas en formato GIF, PNG EPS, entre otros. A partir de su versión 0.20, no es conforme a los estándares UML y carece de soporte completo por algunos tipos de diagrama incluyendo los de secuencia y colaboración." (TIGRIS.ORG, 2001 - 2008)

Visual Paradigm for UML Community Edition 6.4 (VP-UML CE) es una herramienta CASE de licencia libre. Permite la modelación en UML 2.1, con 13 tipos diferentes de diagramas que pueden ser exportados como imágenes en formato JPG, PNG, entre otros. Permite dibujar todos los tipos de diagramas de clase, invertir código, generar código a partir de diagramas y generar la documentación. Puede ser extendido, pues presenta soporte al diseño personalizado, permitiendo importar imágenes e iconos.

VP-UML CE puede ser instalado en múltiples plataformas. Proporciona una interfaz amigable para los desarrolladores permitiéndoles diseñar un producto con calidad y de forma rápida. (PARADIGM, 2004-2005)

BOUML "es una herramienta CASE gratuita, bajo licencia GPL que permite trabajar con UML 2.0 y soporta gran cantidad de diagramas. Muchos desarrolladores la prefieren, pues es fácil de utilizar, rápida y consume poca memoria. Puede generar código para Java, C++ e IDL (y controlar bastante la generación)".(PAGÈS, 2004-2010). Es capaz de generar la documentación en varios formatos (HTML, XML). Aunque no es Java, también es multiplataforma.

Después de analizadas las características de las diferentes herramientas informáticas, y sus potencialidades, se selecciona como herramienta CASE para modelar el desarrollo de alasARBOGEN 2.0 abarcando los artefactos y sus actividades a Visual Paradigm for UML Community Edition 6.4.

2.2.4 PLATAFORMA Y ENTORNO DE DESARROLLO

Java Enterprise Edition o JEE (anteriormente conocido como Java 2 Platform, Enterprise Edition o J2EE hasta la versión 1.4) es una plataforma de programación, para desarrollar y ejecutar software de

aplicaciones en lenguaje de programación Java. Posee una arquitectura de N niveles distribuidos, y se basa en componentes de software modulares que se ejecutan sobre un servidor de aplicaciones.(RODRÍGUEZ, 2009)

La organización en capas y la creación de componentes reutilizables, posibilita una bajo acoplamiento entre sus partes, así como una fuerte reusabilidad y escalabilidad.

Java "es un lenguaje de programación orientado a objetos desarrollado por Sun Microsystem a principios de los años 90. El lenguaje tiene un modelo más simple que el C y el C++ y elimina herramientas de bajo nivel, que suelen inducir a errores. Es un lenguaje compilado y utiliza para ello la Máquina Virtual de Java, por sus siglas en ingles (JVM), la cual lleva el código java a código intermedio (byte code) y posteriormente a código nativo en dependencia del entorno en que este es ejecutado" (JAVA, 2004-2009). Lo anterior permite que el lenguaje pueda ejecutarse en cualquier sistema operativo que soporte su JVM, lo que hace de Java un lenguaje multipropósito. Java tiene una arquitectura neutra, es multi-hilo y portable.

Eclipse 3.3 es un entorno de desarrollo integrado de código abierto y multiplataforma. Su fortaleza radica en su grado de extensibilidad. Eclipse es una superestructura formada por un núcleo y diversos plugins que van conformando la funcionalidad final. (FOUNDATION, 2004-2010)

Eclipse "permite implementar en lenguajes como JAVA, PHP, C, C++ y otros. Las nuevas versiones incluyen paquetes para desarrollar para la web en HTML y XML.

Presenta un potente editor de código que permite realizar completamiento del mismo, incluso del código contenido en las librerías externas agregadas." (FOUNDATION, 2004-2010).

El **Framework Hibernate 3.3** es un entorno de trabajo que tiene como objetivo facilitar la persistencia de los datos a través de base de datos relacionales y a la vez la consulta de bases de datos para obtener objetos. Es una herramienta de asociación entre modelos jerárquicos de objetos y esquemas relacionales de bases de datos, lo que se denomina comúnmente como una herramienta de Mapeo Relacional de Objetos (ORM). (GONZÁLEZ, 2003)

Hibernate presenta un entorno altamente configurativo a través de XMLs y ficheros o archivos de propiedades, HIBERNATE.PROPERTIES e HIBERNATE.CFG.XML.

Una de sus principales características es que evita que los desarrolladores implementen interfaces propietarias o extiendan clases bases propietarias para poder persistir las clases. En su lugar Hibernate trata con la reflection de Java utilizando la librería CGLIB, que se utiliza para extender clases de Java e implementar interfaces Java en tiempo de ejecución.

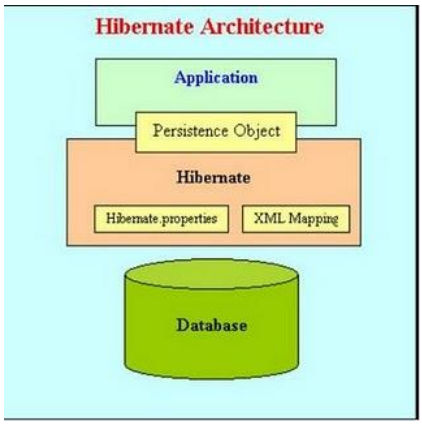


Figura 6: Representación del Framework Hibernate

Otras características importantes a las que expone Jeff Hanson (HANSON, 2004) son:

- ✓ Ocultamiento de objetos: los objetos persistentes se ocultan a través de la sesión a nivel de transacción.
- ✓ No actualización de objetos no modificados: no realiza peticiones innecesarias al servidor de base de datos, manipulando solo la colección de datos si esta fue modificada.
- ✓ Implementación eficiente de la búsqueda a nivel de entidades.

Java TM Web Start es una solución de distribución de aplicaciones basada en tecnología JavaTM. Es la canalización entre Internet y el sistema que permite al usuario ejecutar y gestionar aplicaciones desde la web. Java Web Start proporciona una activación fácil y rápida de las aplicaciones con un único clic y garantiza la ejecución de la última versión de la aplicación, eliminando los complicados procesos de instalación o de modernización.(WAREZ, 2006-2010)

El navegador web automatiza todo el proceso. No existen complicados pasos de descarga, ni de instalación, ni de configuración y tiene la garantía de estar ejecutando siempre la última versión.

La tecnología Java Web Start es una solución de distribución de aplicaciones para la web. El uso de una aplicación con todas las características en lugar de un cliente basado en web tiene diversas ventajas:

- Interfaz de usuario altamente interactiva, comparable a la de las aplicaciones tradicionales, como procesadores de texto u hojas de cálculo.
- Menor necesidad de ancho de banda. Una aplicación no tiene por qué conectarse con el servidor web con cada clic, puede guardar en la ante-memoria la información ya descargada. De esta forma, ofrece una mejor interactividad en conexiones lentas.
- Admite el uso fuera de línea.

Una aplicación basada en la tecnología JavaTM requiere un tiempo de descarga de varios minutos en una conexión de módem habitual. Java Web Start guarda en la ante-memoria todos los archivos descargados en el sistema. De esta forma, aunque el coste de primera activación es más alto para las aplicaciones que para las páginas HTML, las siguientes veces la aplicación se ejecutará de forma casi instantánea, puesto que todos los recursos necesarios estarán disponibles localmente.

Java Web Start está "construido sobre la plataforma Java 2, que proporciona una amplia arquitectura de seguridad. Las aplicaciones ejecutadas con Java Web Start se ejecutarán de forma predeterminada en un entorno restringido ("zona protegida") con acceso limitado a los archivos y a la red" (WAREZ, 2006-2010). Por tanto, la ejecución de aplicaciones mediante Java Web Start mantiene la seguridad e integridad de los sistemas.

Una aplicación puede solicitar acceso sin restricciones al sistema. Java Web Start muestra la información acerca del proveedor que ha desarrollado la aplicación y el usuario erigirá confiar en él o no. Dicha información se basa en la firma digital.

El **Framework Spring 2.0** "es un framework de aplicación de desarrollado por la compañía Interface 21, para aplicaciones escritas en el lenguaje de programación Java. Es considerado un framework liviano y ligero, ya que no requiere de muchos recursos para su ejecución, además, puede ser distribuido completo en un archivo .jar de aproximadamente 1MB. Spring es una aplicación De Código Abierto, por lo que su código fuente está disponible en el paquete de instalación" (JOHNSON, 2005).

No obliga a los desarrolladores a usar un modelo de programación en particular, mas puede ser considerado como sustituto del modelo Enterprise JavaBeans. Además ofrece soluciones bien documentadas y fáciles de usar en prácticas comunes dentro de la industria. Esto se debe a que permite crear componentes reutilizables así como su adaptabilidad e integración con otros frameworks como: Struts, Hibernate, Ibatis, entre otros, funcionalidad que es configurable a través de XML específicos.

Spring fue creado baso en los siguientes principios:

- El buen diseño es más importante que la tecnología subyacente.
- El código debe ser fácil de probar.
- Los JavaBeans ligados de una manera más libre entre interfaces es un buen modelo.

"La arquitectura de Spring está dividida en 7 capas o módulos, integradas por los sub-frameworks que intervienen en su funcionamiento. Esto le permite tomar y ocupar únicamente las partes que interesen para el proyecto y juntarlas con gran libertad." (JOHNSON, 2005). En el presente diagrama se representa la arquitectura de Spring.

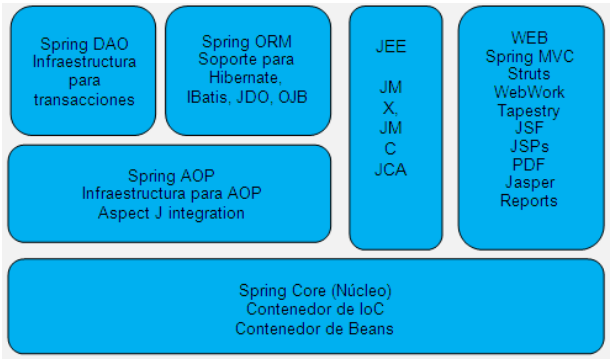


Figura 7: Representación del framework Spring.

Algunos de los elementos más representativos de Spring son:

- Contenedor de inversión de control: La configuración de los componentes de la aplicación y el ciclo de vida es manipulado básicamente por objetos Java comunes.
- Programación orientada a aspectos: El trabajo con las funcionalidades, las cuales no pueden ser implementadas mediante la programación orientada a objetos sin hacer sacrificios de rendimiento y estabilidad.
- Framework de acceso a datos: Trabajo con sistemas de administración objetos-relacionales que proveen soluciones para los desafíos técnicos reusables en multitudes de ambientes basados en Java.
- Manipulación de transacciones: Armonización de varios APIs manipuladores de transacciones orquestadas por objetos Java.
- Modelo-Vista-Controlador: Framework basado en HTTP y Servlets que provee extensiones y personalizaciones.

Un aspecto a destacar en la arquitectura del framework Spring es el sub-framework Contenedor de Inversión de Control (IoC), conocido también como Inyección de Dependencias. En lugar de que el código de la aplicación llame a una clase de una librería, un framework que utiliza IoC llama al código. Es por esto que se le llama ¨Inversión¨, ya que invierte la acción de llamada a alguna librería externa. (JOHNSON, 2005). Este proceso posibilita que las dependencias entre objetos interrelacionados sean mínimas, lo que permite que Spring pueda ser removido sin invocar grandes cambios en el código de la aplicación.

En el Núcleo Contenedor de Spring se encuentran las funcionalidades básicas de este framework. Cuenta con una factoría Beans que constituye el centro de toda aplicación basada en Spring, la cual es

implementada bajo el patrón Factory y que aplica técnicas de IoC para separar la configuración de la aplicación y las dependencias del código de la aplicación.

2.2.5 SERVIDORES WEB

Servidores Web es un programa diseñado para transferir hipertextos, y son indispensables para la publicación y explotación de un sitio web. El programa implementa el protocolo HTTP, perteneciente a la capa de aplicación del modelo de referencia de Interconexión de Sistemas Abiertos (OSI, Open System Interconnection). Estos servidores contienen la lógica para la interpretación de las peticiones de los usuarios a través de los protocolos especificados, lo que hace posible su interacción dinámica entre el cliente y sus acciones sobre el sistema que esté trabajando.

Apache es uno de los servidores web de más popularidad**.** Se caracteriza por ser un servidor ligero, altamente configurable y con una amplia explotación. Encuestas realizadas por la empresa Netcraf, dedicada a la realización de encuestas a nivel global, revelan que el mayor por ciento de los servidores web actuales son servidores Apache.

Entre sus principales características figuran:

- ✓ Altamente configurable y de diseño modular.
- ✓ Tecnología gratuita de código abierto.
- ✓ Multiplataforma, funcionando en la mayoría de los sistemas operativos.
- ✓ Soporte para Interface de Entrada Común (CGI, Common Gateway Interface) es un estándar para poder ejecutar scripts en las páginas web).
- ✓ Soporte para varios lenguajes: PHP, Java, Perl, y librerías ASP.
- ✓ Soporte para el protocolo HTTP.
- ✓ Soporte de host virtuales.
- ✓ Servidor proxy integrado.

Apache Tomcat 6.0 es una aplicación de software de Código Abierto que implementa las especificaciones de Java Servlet 2.5 y Java Server Pages 2.1. Fue desarrollado bajo el proyecto Jakarta en la Fundación de Software Apache (Apache Software Fundation), en un entorno abierto y participativo y publicado bajo la licencia del software de Apache. Puede funcionar como servidor HTTP o conectado a

otro servidor HTTP como Apache HTTP o Servicio de Información de Internet (IIS, Internet Information Service). Permite ejecutar servicios web mediante Apache Axis. (SOFTWARE, 2010)

2.2.6 ADMINISTRACIÓN DE BASE DE DATOS

PostgreSQL es un sistema de Gestión de Bases de Datos Objeto-Relacional (ORDBMS) basado en el proyecto POSTGRES. PostgreSQL es una derivación libre de Código Abierto de este proyecto, y utiliza el lenguaje SQL92/SQL99. Fue diseñado para ambientes de alto volumen de datos como Oracle, Sybase o Interbase, soportando transacciones. Desde su versión 7.0 soporta claves ajenas con comprobación de integridad referencial.(PECOS, 2009)

Es un sistema objeto-relacional, ya que incluye características de la orientación a objetos, como puede ser la herencia, tipos de datos, funciones, restricciones, disparadores, reglas e integridad transaccional, sin embargo, no es un sistema de gestión de base de datos puramente orientado a objetos.(PECOS, 2009)

Posee una gran escalabilidad, haciéndolo idóneo para su uso en sitios web que posean alrededor de 500.000 peticiones por día.

Otras características importantes de PostgreSQL son:

- ✓ Soporta distintos tipos de datos: además del soporte para los tipos base, también soporta datos de tipo fecha, monetarios, elementos gráficos, datos sobre redes (MAC, IP...), cadenas de bits, etc. También permite la creación de tipos propios.
- ✓ Incorpora una estructura de datos de arreglos.
- ✓ Incorpora funciones de diversa índole: manejo de fechas, geométricas, orientadas a operaciones con redes, etc.
- ✓ Soporta el uso de índices, reglas y vistas.
- ✓ Incluye herencia entre tablas (aunque no entre objetos, ya que no existen), por lo que a este gestor de bases de datos se le incluye entre los gestores objeto-relacionales.
- ✓ Permite la gestión de diferentes usuarios, como también los permisos asignados a cada uno de ellos.
- ✓ Permite el Control de Concurrencia Multiversión (MVCC en ingles Multiversión Concurrency Control)
- ✓ Escritura adelantada de registros para evitar pérdida de datos en caso de falla de energía, del sistema operativo o del hardware.

- ✓ Soporte en la mayoría de los sistemas operativos más utilizados: Linux, varias versiones de Unix (AIX, BSD, HP-UX, SGI IRIS , Mac OS X, Solaris, SunOS, Tru64, BeOS y Windows).

Una de sus desventajas es la sobrecarga al sistema por el consumo de un número significativo de recursos.

MySQL es un "sistema de gestión de Bases de Datos relacional, bajo la licencia GPL de GNU. Fue creado por la empresa Sueca MySQL AB, que mantiene el *copyrigth* del código fuente del servidor SQL, así como la marca." (MYSQL, 2008)

MySQL figura entre los gestores de bases de datos más usados del mundo del Software Libre, por las facilidades de uso que proporciona y su gran rapidez. Soporta gran variedad de sistemas operativos y una baja probabilidad de que los datos se corrompan, incluso si los errores no se producen en el propio gestor, sino en el sistema en el que está.

Este gestor cuenta además con infinidad de librerías y otras herramientas que permiten su uso a través de varios lenguajes de programación, además de su fácil instalación y configuración.

Entre sus principales características (MYSQL, 2008) figuran:

- Cada base de datos cuenta con 3 archivos: uno de estructura, uno de datos y uno de índice, soportando hasta 32 bit por tabla.
- Su implementación multi-hilo le permite aprovechar la potencia de los sistemas multiprocesador.
- Posee API's para el soporte de varios lenguajes como: C++, Java, C, PHP, entre otros, lo que le permite tener gran portabilidad entre sistemas.
- Gestión de usuarios y contraseña para el mantenimiento de un excelente nivel de seguridad, incorporando un sistema de privilegios muy flexibles que permite la verificación basada en host.
- Presenta un sistema de reserva de memoria muy rápido apoyado en hilos de ejecución, así como tablas *hash* en memoria, que se utilizan como tablas temporales.
- Incorpora múltiples motores de almacenamiento (MyISAM, Margue, InnoDB, BDB, Memory/heap, MySQL Cluster, Federated, Archive, CSV, Blackhole, y Example en 5.x) que le posibilita al usuario escoger el que sea más adecuado para cada tabla de la base de datos.
- Posee procedimientos de almacenados, disparadores, vistas actualizables y agrupación de transacciones.

MySQL posee además una escalabilidad que posibilita manipular bases de datos enormes del orden de seis mil tablas, alrededor de cincuenta millones de registros y hasta 32 índices por tabla.

2.3 PROPUESTA TECNOLÓGICA

Tras analizar los resultados se puede determinar una propuesta de tecnológica en un entorno de desarrollo libre.

Con el objetivo de cumplir con las funcionalidades antes descritas se decide la implementación de dos aplicaciones, una web y una de escritorio, que gestionarán sus negocios de forma independiente.

Se selecciona a Java como lenguaje de programación a utilizar, teniendo en cuenta que desde la aplicación de escritorio se graficarán los árboles genealógicos, para lo cual Java brinda excelentes librerías; además, los Servlet de Java contienen todo el potencial de J2EE, lo que es un gran respaldo considerando que se pueden integrar fácilmente sistemas ya desarrollados para llevarlos a la web.

La aplicación web, se implementará en este mismo lenguaje para evitar posibles problemas de compatibilidad y facilitar el uso de la tecnología Java Web Start, la cual posibilita que la aplicación de escritorio pueda ser instalada y descargada desde la web. Cuando se utiliza Java como lenguaje de programación, el servidor web más adecuado es el Apache Tomcat, por lo que se decide utilizarlo.

Para el mapeo relacional de los datos, se utilizará el framework Hibernate, el cual proporciona transparencia entre el gestor de bases de datos y el sistema, posibilitando así que el grado de dependencia del sistema con la bases de datos, sea mínimo. El uso del MySQL trae como consecuencia el pago de su licencia, por lo que se utilizará PostgreSQL como gestor de base de datos.

Teniendo en cuenta las dificultades que implica la migración inmediata de todas las bases de datos de Infomed a PostgreSQL, actualmente se valora la utilización de arquitecturas orientadas a servicios, de manera tal que puedan seguir siendo utilizadas de forma satisfactoria e independientemente la tecnología empleada.

PostgreSQL permite eliminar los problemas de concurrencia en el acceso a la base de datos, al eliminar el bloqueo a nivel de tabla. En un sistema tradicional, cada registro que es modificado se bloquea hasta que se confirma la transacción, previniendo lecturas por otros usuarios. MVCC utiliza de modo natural el carácter multiversión de PostgreSQL para permitir que las lecturas continúen leyendo datos consistentes durante la actividad de escritura.

2.4 ORGANIZACIÓN DEL SISTEMA

La arquitectura propuesta para el sistema informático a realizar está basada en el patrón arquitectónico Modelo Vista Controlador, presente en la aplicación de escritorio y en la web a través de la utilización del Framework Spring.

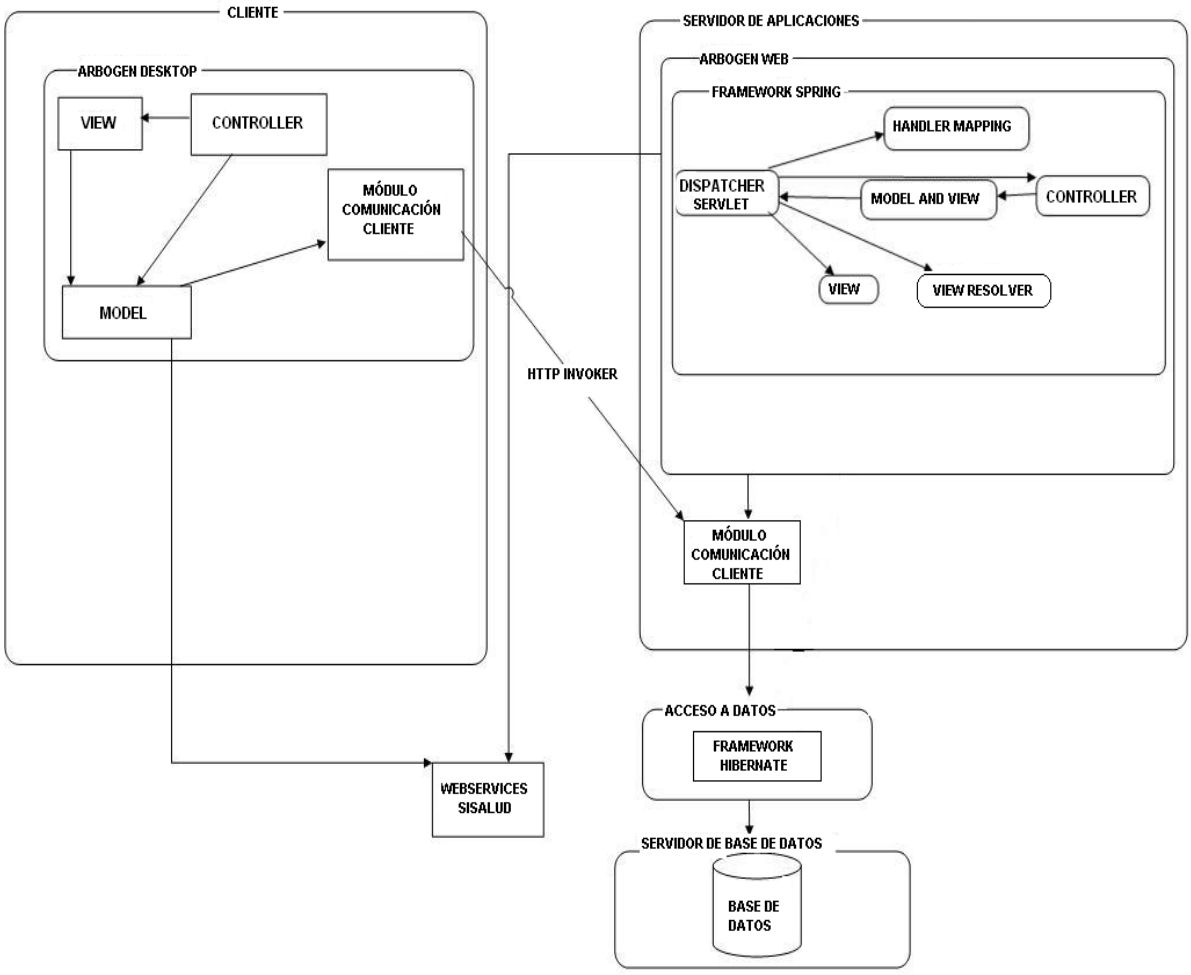


Figura 8. Organización General del Sistema.

Mediante la aplicación de escritorio se genera la información concerniente a los datos genealógicos de la familia que se estudia. Algunos de estos datos son obtenidos como resultado de consumir los servicios que brinda SISalud. La información resultante del estudio es procesada por el componente encargado de transferir los datos al servidor de bases de datos, después de haber sido validados. Esta información se transfiere de forma segura por HTTP INVOKER.

La aplicación web, se encuentra desplegada en el servidor de aplicaciones y, posibilitará la descarga e instalación de la aplicación de escritorio haciendo uso de JavaWebStart.
El sistema, consumirá servicios de SISalud, específicamente el servicio de Registro de Ciudadanos y el Servicio de Autenticación.
El componente de comunicación está formado por dos librerías independientes, una para el lado del cliente y otra para el lado del servidor, en esta última el servidor almacena toda la lógica de validación del XML a través del esquema. Además un conjunto de componentes que constituyen los archivos de configuración, las librerías utilizadas, los archivos necesarios para establecer la seguridad con SSL y los archivos necesarios para garantizar la trazabilidad en el envío. **Ver Anexo 2.**
La capa de acceso a datos contiene la lógica de mapeo de la información de las bases de datos con los objetos de las clases que se deciden hacer persistentes. Esto se logra con la utilización del Framework Hibernate. Este facilita el mapeo de atributos entre la base de datos y el modelo de objetos de la aplicación mediante archivos XML.

2.5 PATRÓN ARQUITECTÓNICO UTILIZADO

El sistema alasARBOGEN en su nueva versión 2.0, está constituido por una aplicación Web y una de Escritorio. Para lograr un diseño claro y ampliamente aceptado se utilizará el patrón arquitectónico Modelo-Vista-Controlador (MVC) donde se separa el modelo, 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.

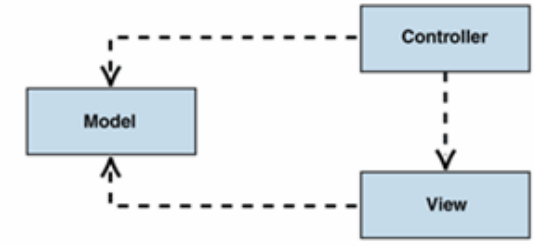


Figura 9. Representación del patrón MVC.

El patrón MVC no es aplicado de la misma forma en estas aplicaciones:
La aplicación web será desarrollada utilizando Spring, framework contenedor liviano basado en la técnica Inversión de Control (IoC) facilitando la separación de la lógica de los controladores de los objetos de negocios de una manera elegante, y en una implementación de desarrollo según el Paradigma de Orientación a Aspectos (AOP). MVC es uno de los módulos del Framework de Spring. Este se encuentra dividido en componentes, donde cada uno de ellos cuenta con una función diferente.(ROD JOHNSON, 2004-2008):

- ✓ **DispatcherServlet**: Es por donde se dirige y se controla el flujo de aplicación, funciona como un controlador de aplicación.
- ✓ **Controller**: Es el encargado de realizar o dirigir el modelo de negocio de la aplicación respondiendo a la petición realizada por el DispatcherServlet mediante el ModelandView.
- ✓ **ModelandView**: Contiene la respuesta en una Vista (View) o el nombre lógico de la misma.
- ✓ **ViewResolver**: Decide buscar el Vista (View) correspondiente al nombre lógico del Vista (View).
- ✓ **View**: Contiene las vistas, que son las que se van a mostrar al cliente con una respuesta dada.

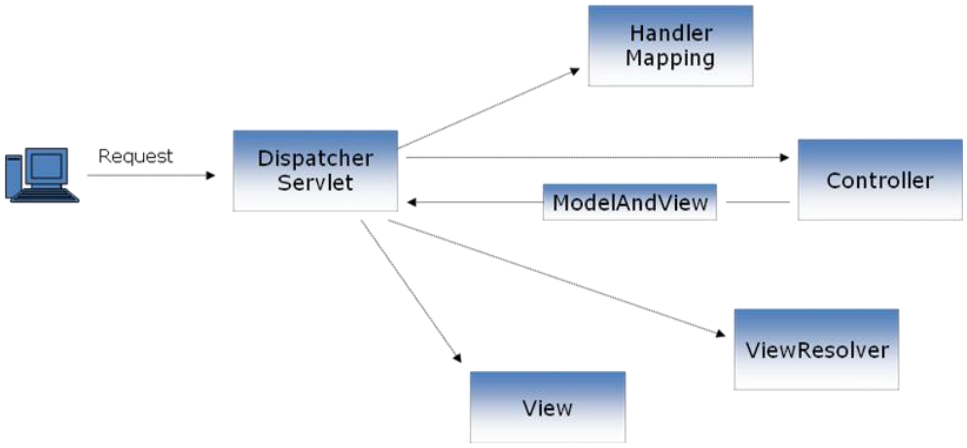


Figura 10. Representación de la arquitectura de Spring.

Este patrón favorece a las aplicaciones escritorio realizadas en java, ya que su modularidad, permite que desde la vista se pueda tener acceso a las clases del modelo, dando gran ventaja a aplicaciones como estas, en las que se hace uso intensivo de componentes visuales y requieren de una actualización constante de las propiedades que se modifican.

2.6 DESCRIPCIÓN DE LA ARQUITECTURA.

La arquitectura de software muestra una visión común del sistema que incluye los componentes principales del mismo, en la que el equipo de proyecto y los usuarios deben estar de acuerdo. Un elemento importante dentro de la arquitectura de software es que debe definirse a través de vistas, que representan las diferentes perspectivas del diseño, muestran el diseño estructural de una aplicación de forma general centrándose en aspectos concretos y describiendo las principales partes del sistema. (GRADY BOOCH, IVAR JACOBSON, JAMES RUMBAUGH., 2000).
RUP permite representar la arquitectura mediante varias vistas, que se centran en aspectos concretos y describen las principales partes del sistema. Estas se describen detalladamente a continuación.

2.7 VISTA DE CASOS DE USO

En la vista de casos de uso se representan los casos de uso (CU) arquitectónicamente significativos, los que engloban las funcionalidades fundamentales y críticas del sistema, o que implementan algún requisito importante que debe ser desarrollado con prontitud dentro del ciclo de vida del sistema.

En la siguiente tabla se muestran los actores involucrados en los CU arquitectónicamente significativos:

Genetista	El especialista en genética es el que interactúa con el sistema y el único encargado de realizar el árbol genealógico.
Administrador	Es el encargado de la administración del los usuarios y su gestión, así como de los nomencladores. No necesariamente debe ser una especialista en genética.
SAAA	Componente de seguridad con el cual debe interactuar el sistema para acceder a cualquier otro componente de SISalud.
Registro de Ciudadano	Componente de SISalud que el proporciona al sistema información primaria de los pacientes.

Tabla 1. Actores del Sistema

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

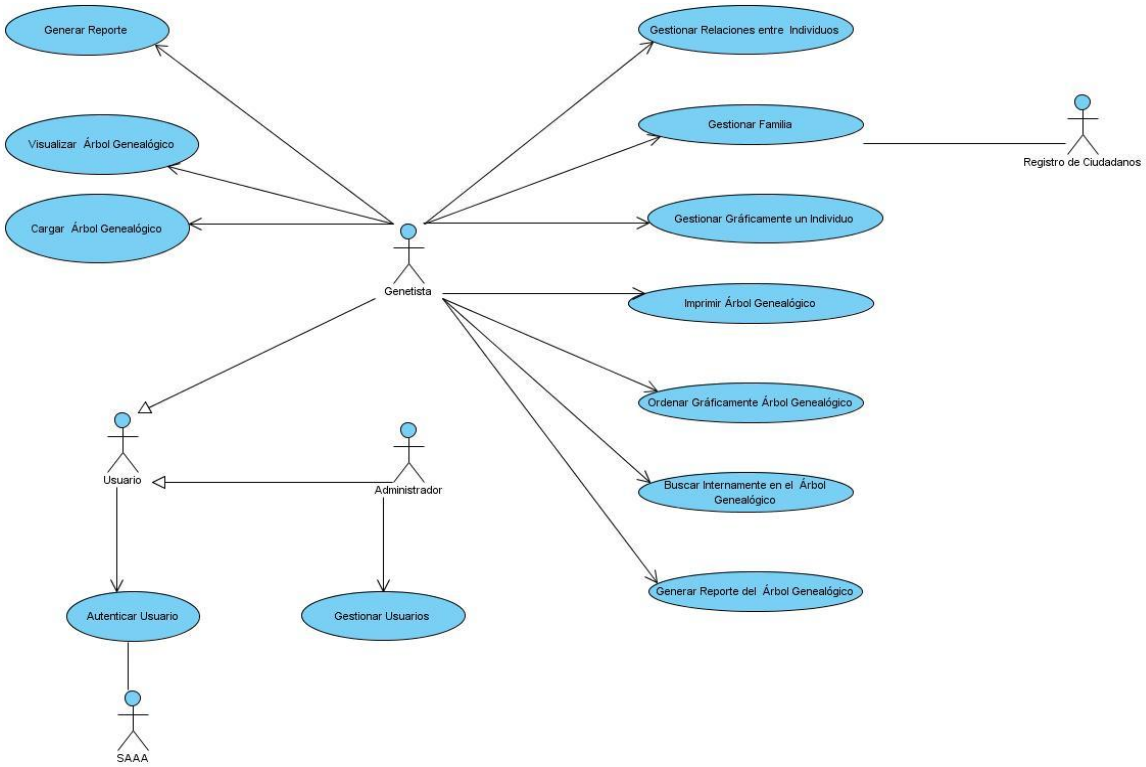


Figura 11. Vista de casos de uso del sistema.

2.7.1 BREVE DESCRIPCION DE LOS CASOS DE USO ARQUITECTÓNICAMENTE SIGNIFICATIVOS.

Describir un CU consiste en proporcionar de forma textual y mediante un lenguaje natural, la secuencia de eventos y acciones que este engloba, especificando las precondiciones y poscondiciones de ese flujo de eventos. A continuación se describen los casos de uso arquitectónicamente significativos.

2.7.1.1 BUSCAR EN EL ÁRBOL GENEALÓGICO.

El caso de uso comienza cuando el genetista realiza una búsqueda en el árbol genealógico. El sistema muestra una ventana de diálogo, para que el genetista seleccione el criterio de búsqueda por el que desea realizar la búsqueda en el árbol genealógico. Estos criterios pueden ser intento de suicidio, muestras de laboratorio, o enfermedades genéticas. El caso de uso termina cuando se efectúa la búsqueda correspondiente al criterio o el genetista decida cancelar la búsqueda.

Precondiciones: El sistema tiene que estar iniciado y el árbol genealógico debe haber sido creado con anterioridad.
Poscondiciones: Se muestran los resultados de la búsqueda.

2.7.1.2 ORDENAR GRÁFICAMENTE ÁRBOL GENEALÓGICO.

El caso de uso se inicia cuando el genetista da clic en la opción ordenar árbol. Finaliza una vez que se ha ordenado el árbol genealógico.
Precondiciones: Para ordenar el árbol genealógico, este debe estar creado. La aplicación debe estar iniciada.
Poscondiciones: El árbol genealógico queda ordenado jerárquicamente.

2.7.1.3 IMPRIMIR ÁRBOL GENEALÓGICO.

El caso de uso se inicia cuando el genetista va a imprimir el árbol genealógico de la persona. El sistema le muestra la ventana de diálogo correspondiente a la opción. El caso de uso finaliza una vez que se imprima el árbol genealógico, o el genetista decida cancelar la opción.
Precondiciones: La aplicación debe estar iniciada. Para imprimir el árbol genealógico, este debe estar creado.
Poscondiciones: Se obtiene el árbol impreso.

2.7.1.4 GENERAR REPORTE DEL ÁRBOL GENEALÓGICO.

El caso de uso se inicia cuando el genetista procede a generar un reporte del árbol genealógico. El sistema le muestra una ventana de diálogo para que seleccione el directorio en el que desea guardar el reporte generado. El caso de uso termina cuando se guarde el reporte generado, o el genetista seleccione la opción cancelar.
Precondiciones: La aplicación debe estar iniciada. El árbol genealógico debe estar creado.
Poscondiciones: Se obtiene un reporte de acuerdo a los parámetros seleccionados.

2.7.1.5 AUTENTICAR USUARIO.

El caso de uso se inicia cuando el usuario desea autenticarse en la aplicación, el sistema muestra la interfaz correspondiente y el usuario introduce sus datos. El sistema habilita los privilegios, y finaliza caso de uso.

Precondiciones: La aplicación debe estar iniciada.

Poscondiciones: Se habilitan las funcionalidades según los privilegios.

2.7.1.6 GESTIONAR USUARIO.

El caso de uso se inicia cuando el Administrador decide realizar alguna de las siguientes operaciones:

- Adicionar usuario.
- Modificar usuario.
- Buscar usuario.
- Eliminar usuario.

El sistema mostrará la interfaz correspondiente a la opción seleccionada, donde se introducirán los datos requeridos que luego se validarán. El sistema verificará que el usuario exista, en caso de que se desee adicionar, modificar y eliminar, no así para la búsqueda, donde mostrará un mensaje indicando un error en caso de que ocurra.

Precondiciones: El usuario debe haberse autenticado anteriormente como administrador del sistema.

Poscondiciones: En dependencia de la acción del Administrador:

- Se adiciona un nuevo usuario.
- Se modifica los datos de un usuario existente.
- Se elimina un usuario.

2.7.1.7 VISUALIZAR ÁRBOL GENEALÓGICO.

El caso de uso se inicia cuando el genetista desea visualizar el árbol genealógico de un paciente, el sistema muestra la interfaz correspondiente. Si no hay resultados en la búsqueda, el sistema muestra un mensaje indicándolo.

Precondiciones: El usuario debe estar autenticado como genetista.

Poscondiciones: El sistema visualiza un árbol genealógico.

2.7.1.8 CARGAR ÁRBOL GENEALÓGICO.

El caso de uso se inicia cuando el genetista desea cargar a la aplicación un árbol genealógico de un paciente. El sistema muestra la interfaz correspondiente y el genetista busca la dirección donde se encuentra el archivo, lo carga y selecciona la opción guardar. El sistema guardará el archivo y mostrará un mensaje indicando que se realizó el proceso de forma correcta, de lo contrario mostrará un mensaje de error.

Precondiciones: El usuario debe estar autenticado como genetista.

Poscondiciones: El sistema guarda un árbol genealógico.

2.7.1.9 GENERAR REPORTE.

El Caso de Uso se inicia cuando el genetista desea generar un reporte de acuerdo a una serie de parámetros específicos. El sistema muestra la interfaz correspondiente y el genetista introduce los datos.

Precondiciones: El usuario debe estar autenticado como genetista.

Poscondiciones: Se genera un reporte con los datos que se seleccionaron en el formato deseado.

2.7.1.10 GESTIONAR GRÁFICAMENTE UN INDIVIDUO.

El caso de uso se inicia cuando el genetista va a realizar alguna de las siguientes operaciones relacionadas con la gestión gráfica del individuo.

- Insertar un individuo.
- Modificar datos de un individuo.
- Mostrar Datos de un individuo.
- Eliminar un individuo.

El sistema mostrará la interfaz correspondiente según su solicitud y ejecuta las acciones necesarias. El caso de uso finaliza cuando se ejecuta alguna de las funciones seleccionadas.

Precondiciones: Debe estar iniciada la aplicación. Para modificar datos y eliminar los datos, el individuo debe haber sido creado con anterioridad.

Poscondiciones: El sistema inserta, modifica, elimina, muestra o traslada un individuo.

2.7.1.11 GESTIONAR RELACIONES ENTRE INDIVIDUOS.

El caso de uso se inicia cuando el genetista va a realizar alguna de las siguientes operaciones relacionadas con la gestión de relaciones entre individuos.

- Crear relación entre individuos.
- Eliminar relación entre individuos.

El sistema mostrará la interfaz correspondiente según su solicitud y ejecuta las acciones necesarias. El caso de uso finaliza cuando se ejecuta alguna de las funciones seleccionadas.

Precondiciones: Debe estar iniciada la aplicación, al menos un individuo representado y para eliminar una relación tiene que haber al menos dos individuo relacionados.

Poscondiciones: El sistema crea o elimina una relación entre individuos.

2.7.1.12 GESTIONAR FAMILIA.

El caso de uso se inicia cuando el genetista va a realizar alguna de las siguientes operaciones relacionadas con la gestión de la familia.

- Crear una nueva familia.
- Buscar una familia.
- Modificar datos de una familia.
- Guardar familia.
- Exportar familia.

El sistema mostrará la interfaz correspondiente según su solicitud y ejecuta las acciones necesarias. El caso de uso finaliza cuando se ejecuta alguna de las funciones seleccionadas.

Precondiciones: Para crear una familia es necesario haber iniciado la aplicación. Para buscar, modificar, guardar y exportar es necesario que exista al menos una familia creada con anterioridad.

Poscondiciones: El sistema crea, busca, modifica, guarda y exporta una familia.

2.8 VISTA LÓGICA.

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

Como se expuso en el capítulo anterior, el patrón arquitectónico a utilizar es el Modelo Vista Controlador, por cuanto la estructura organizacional del sistema alasARBOGEN 2.0, 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.

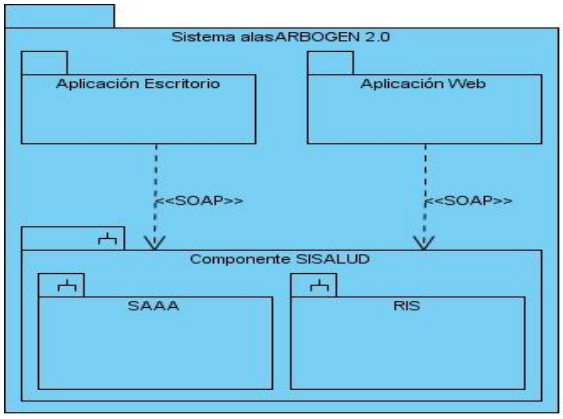


Figura 12. Vista lógica del sistema.

El sistema alasARBOGEN 2.0 se compone de dos subsistemas, una aplicación web y una de escritorio, por lo que se procederá a la descomposición del diagrama anterior, para una mayor compresión de sus partes.

El subsistema Componente SISalud representa los componentes de SISalud que consumirá el sistema, el SAAA para la autenticación y el RIS para el registro de ciudadanos.

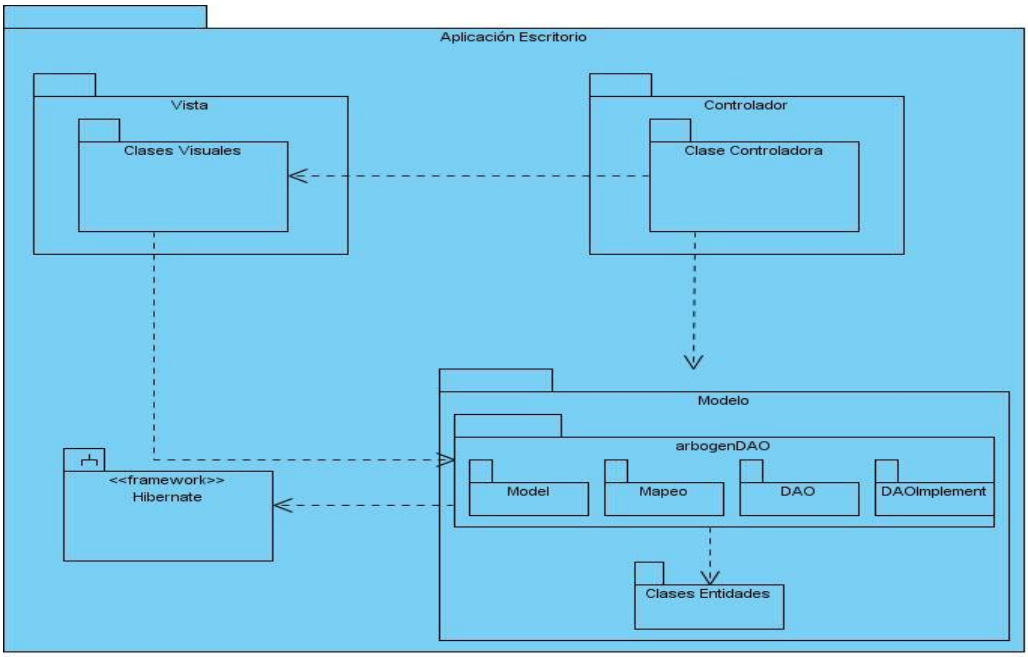


Figura 13. Vista lógica de la aplicación de escritorio.

En la aplicación de escritorio, el paquete vista, contiene las clases visuales que proporcionan las interfaces necesarias para trabajar e interactuar con la misma. El paquete controlador agrupa las clases que implementan la lógica de la aplicación y el modelo contiene las clases que encapsulan la lógica del dominio, y se encargan del acceso a los datos.

El paquete Clases Entidades contienen las clases persistentes para la aplicación, estas utilizan arbogenDAO para el modelado y mapeo relacional de los datos.

arbogenDAO contiene los paquetes Model, Mapeo, DAO, y DAOImplement. A continuación se describen los mismos:

Model: Contiene las clases del dominio de la aplicación.

Mapeo: Almacena los archivos hbm o xml, que posibilitan mapear las clases del dominio con las tablas de la base de datos.

DAO: Contiene las interfaces del acceso a datos que permiten realizar las consultas a la base de datos.

DAOImplement: Contiene la implementación de las interfaces del DAO.

Subsistema Hibernate: Nos provee la clase *HibernateDaoSupport* para brindarle soporte a arbogenDAO.

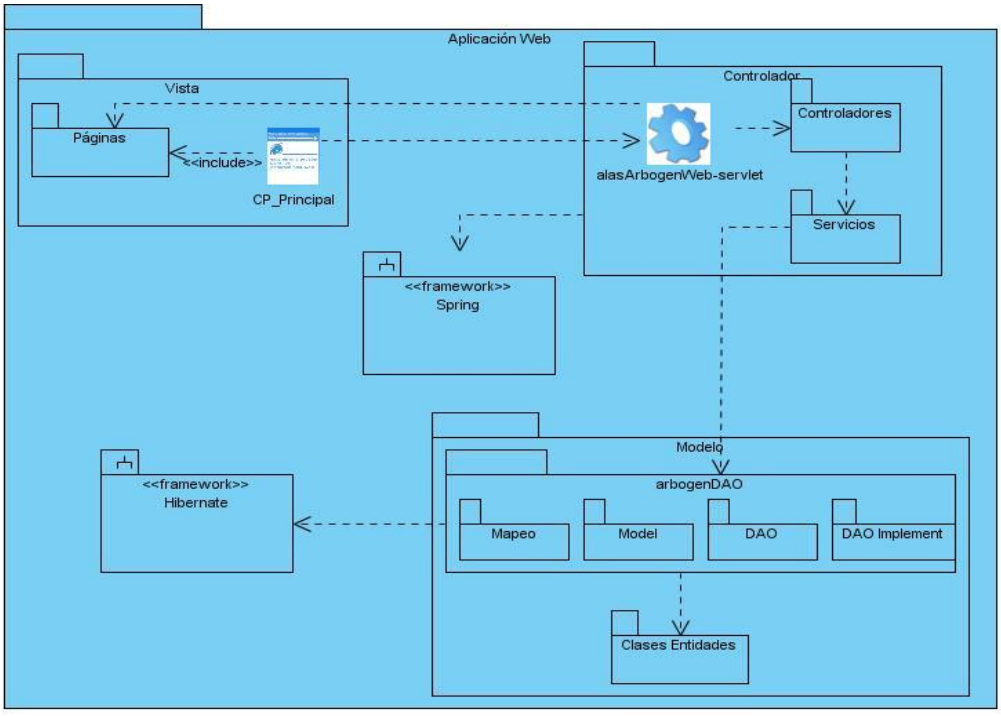


Figura 14. Vista lógica de la aplicación web.

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

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

La clase **CP_Principal** constituye la vista principal de la aplicación y desde ella se accede a todo el conjunto de funcionalidades que la misma proporciona.

Paquete Páginas: Agrupa todas las plantillas o páginas de cada módulo que se encargan de visualizarlas a partir de los datos especificados en el controlador.

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

La clase controlador frontal **alasARBOGENWeb-servlet** recibe y gestiona todas las peticiones, dirigiéndolas hacia los controladores que contienen la especificidad de la petición realizada, recibiendo posteriormente la respuesta de los mismos, la cual es visualizada mediante las páginas.

Paquete Controladores: Engloba los controladores específicos de cada uno de los módulos, los cuales reciben las peticiones del controlador frontal y se encargan de direccionar la misma al servicio correspondiente.

Paquete Servicios: Es una capa intermedia entre los controladores y los DAO que proporcionan interfaces para el acceso a los métodos de las clases entidades.

Paquete 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. Cada uno de los paquetes que lo componen (Paquete Clases Entidades, Model, Mapeo, DAO, DAOImplement), realizan las mismas funciones descritas en la aplicación de escritorio.

Subsistema Hibernate: Nos provee la clase *HibernateDaoSupport* para brindarle soporte a arbogenDAO.

Subsistema Spring: Representan las clases del framework Spring que serán utilizadas durante el funcionamiento de la aplicación, tales como componentes de seguridad y configuración.

2.9 VISTA DE DESPLIEGUE.

La vista de despliegue modela la configuración física sobre la que se encuentra desplegado el software**.** En esta se muestran 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. Estos nodos son elementos de hardware sobre los cuales pueden ejecutarse los elementos de software.

A continuación se representa la vista de despliegue propuesta para el sistema alasARBOGEN 2.0.

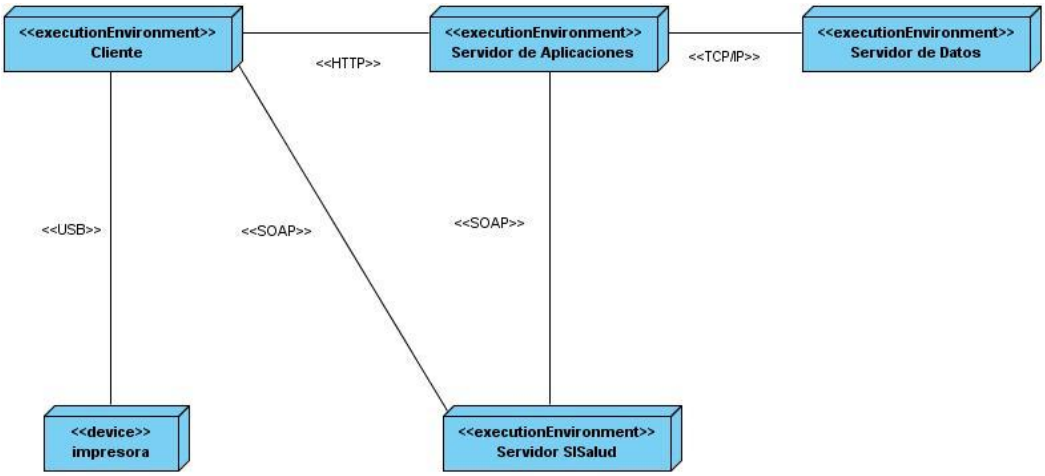


Figura 15. Vista de despliegue

En el nodo **Cliente** es donde se desarrollan los procesos de interacción del usuario con el sistema. En él se encuentra almacenada la aplicación de escritorio y se visualiza la aplicación web. Desde el mismo se generan los reportes que se imprimirán en el nodo **impresora,** el cual se encuentra conectado al cliente por USB.

Desde el **Cliente** se consumen los servicios que brinda el nodo **Servidor de SISlaud** a través del protocolo SOAP. Este nodo proporciona los servicios de autenticación por SAAA y el Registro Informatizado de Ciudadanos.

El nodo **Servidor de Aplicaciones** contiene a la aplicación web, y al igual que el nodo **Cliente** utiliza los servicios del **Servidor de SISalud** a través del protocolo SOAP.

Los datos que se generan en el nodo **Cliente** son enviados de forma segura al **Servidor de Aplicaciones** a través del protocolo HTTP, y desde ahí al **Servidor de Datos,** donde será almacenada dicha información.

En el nodo **Servidor de Datos** es donde se almacena la totalidad de la información que se genera como resultado de la interacción con el sistema. El mismo alberga la base de datos centralizada de alasARBOGEN 2.0. Al mismo se accede a través del protocolo TCP/IP.

2.10 VISTA DE IMPLEMENTACIÓN.

La vista de implementación representa los componentes y subsistemas principales del sistema y sus dependencias, 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 (GUSTAVO TOROSSI, 2005).

El sistema alasARBOGEN 2.0 está formado por dos subsistemas, que los componen la aplicación de escritorio y la aplicación web. A continuación se desglosarán ambos subsistemas en sus respectivas vistas de implementación, en pos de lograr una mayor comprensión de los mismos.

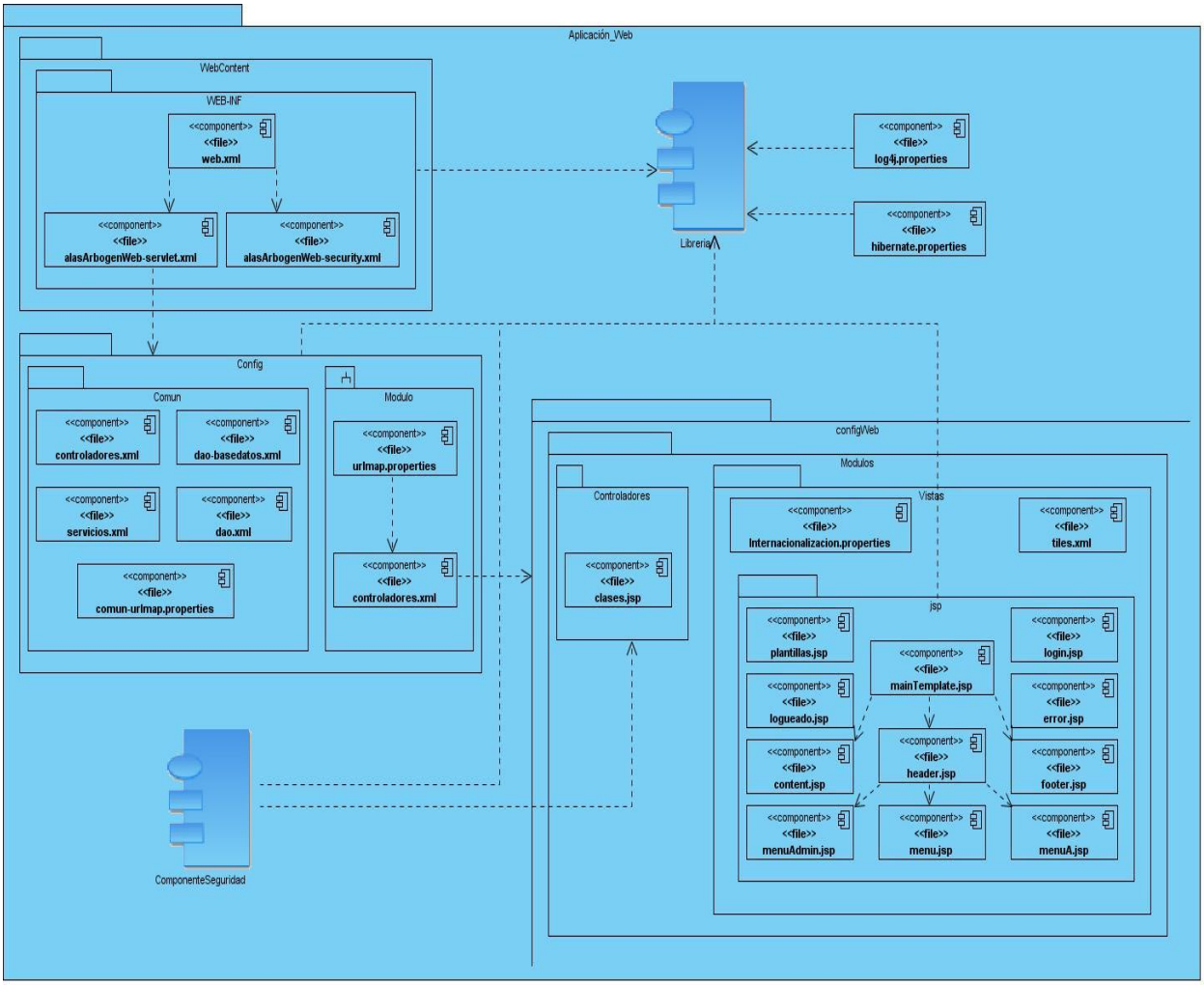


Figura 16: Vista de implementación. Aplicación web.

La vista se conforma de 4 paquetes que engloban los principales componentes y subsistemas de esta aplicación.

El **WEB-INF** contiene el componente **web.xml**, el cual controla hacia donde serán direccionadas todas las solicitudes, además de cual será el controlador frontal de la aplicación, que en este caso es el componente **alasARBOGENWeb-servlet.xml**. El componente **alasARBOGENWeb-security.xml**, contiene la configuración que permite regular los niveles de acceso a las páginas en dependencia del rol del usuario que interactúa con la aplicación.

En el paquete **Config** engloba **común** y **Módulo**. El primero contiene los archivos de configuración comunes para todos los módulos que se desarrollen, los cuales determinan la lista de **controladores**, los **servicios**, entre otros, relacionados con el mapeo relacional de datos. Módulo permite dividir las diferentes funcionalidades en subsistemas. Cada uno posee un archivo **urlmap.properties** y un **controlador.xml,** que especifican cuales serán las clases que se mostrarán, y que se encuentran dentro de **Vistas** en el paquete **configWeb**.

Vistas contiene las clases jsp que se desplegarán en dependencia del módulo correspondiente. Este paquete almacena los componentes **internacionalización.properties**, que permite cargar mensajes de texto como base para la internacionalización de la aplicación, o sea, mostrar la misma en otros lenguajes sin necesidad de cambiar el código y el componente **tiles.xml**, que proporciona información sobre cuales serán las páginas jsp que se cargarán cuando se ejecute ese módulo.

El paquete Librería, almacena las librerías que utiliza la aplicación. En el diagrama se representan solamente las principales, entre ellas figuran; **Spring.jar, jnlp-servlet.jar, postgresql-8.3-603.jdbc3**.

spring.jar: Contiene las clases del núcleo de Spring.

Postgresql-8.3-603.jdbc3: Permite realizar la conexión con la Base de Datos.

La configuración necesaria para la utilización de la librería **Hibernate**, se establece en el componente **hibernate.properties**, que contiene además los parámetros para la conexión a la base de datos. **arbogenDAO**, es el resultado del mapeo relacional de los datos existentes en la bases de datos del sistema.

El componente **log4.properties**, se utiliza para especificar donde se almacenarán las trazas de la aplicación y de cuales elementos de la misma se llevará un registro. Para su utilización se requiere de la librería **log4.jar.**

El **Componente de Seguridad** es el encargado de que la información viaje de forma segura a través de la red. Es utilizado por ambas aplicaciones, por lo cual cumple el mismo objetivo para la aplicación de escritorio.

La aplicación de escritorio queda constituida por un conjunto de componentes y subsistemas, los cuales representan las carpetas que se encuentran dentro del directorio de la aplicación y los ficheros que son modelados en términos de componentes. La siguiente figura así lo representa.

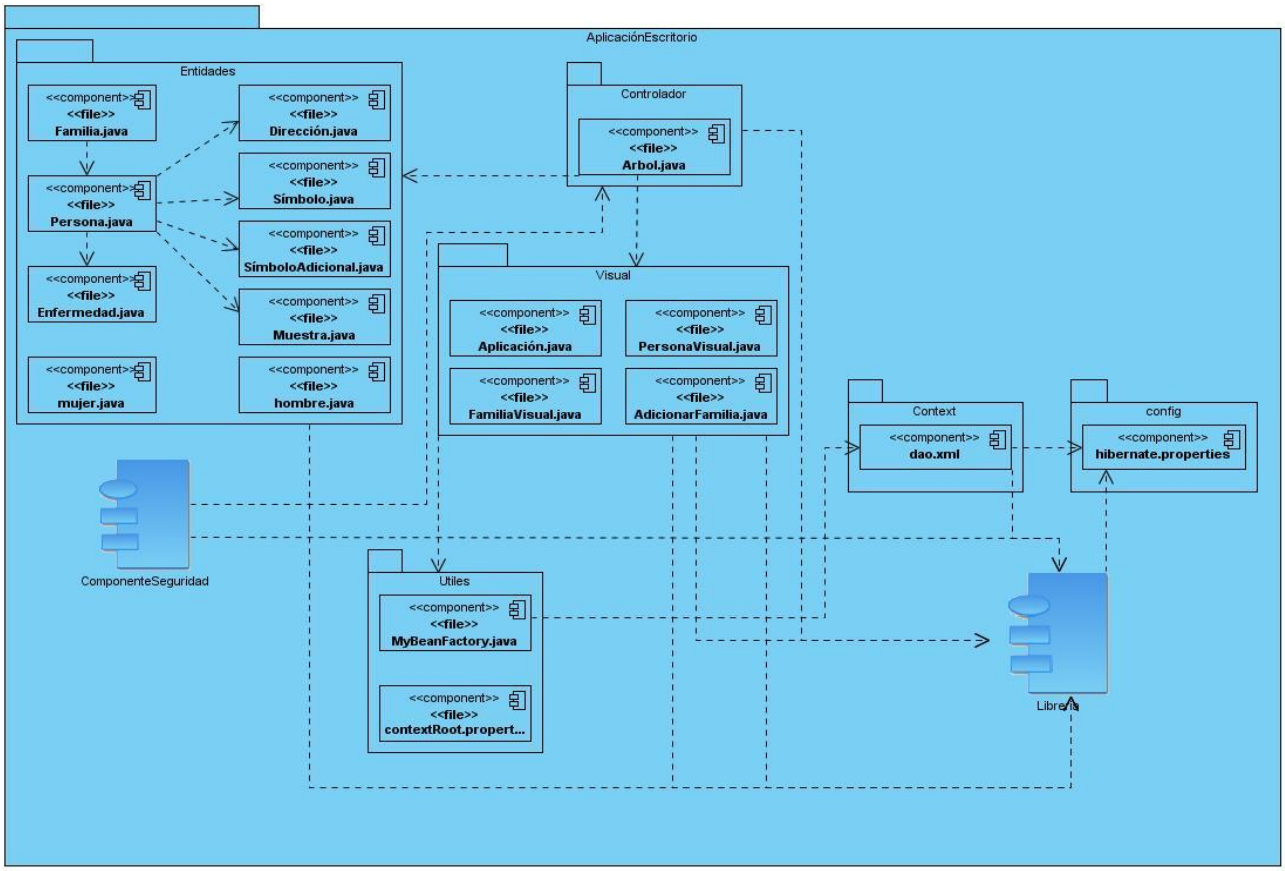


Figura 17: Vista de implementación. Aplicación de escritorio.

Paquete Entidades: Contiene las clases persistentes de la aplicación, tales como Familia, Persona y Enfermedad, Muestra, Símbolo, etc.

Paquete Controlador: Contiene la clase principal de la aplicación, este es el encargado de controlar todo el flujo de eventos y acciones que son realizadas en la misma.

Paquete Visual: Contiene un conjunto de clases que permiten visualizar la información de la aplicación. Las cuales utilizan el Property para mostrar la información que sea generada de la persona o la familia en general.

Paquete Context: Contiene la clase DAO.xml la cual devuelve cada uno de los objetos que se encuentran dentro de la librería ArbogenDAO.jar.

Paquete Config: Posee el fichero de configuración de Hibernate donde se especifican cada uno de los valores de entrada, la dirección del servidor, el puerto, el nombre de la BD, el usuario y la contraseña que desea realizar una determinada acción.

Paquete Útiles: Engloba un conjunto de clases que brindan información de interés específico en la aplicación.

MyBeanFactory: Es la clase encargada de devolver los objetos de un tipo específico.

ContextRoot.properties: Es la clase que contiene el directorio donde se encuentra el dao.xml.

Paquete Librería: Contiene un conjunto de librerías que permiten el funcionamiento de la aplicación.

ArbogenDAO.jar: Permite obtener el resultado del mapeo relacional de los datos existentes en la bases de datos del sistema.

Autenticación.jar: Permite la autenticación directa con el SAAA.

Postgresql-8.3-603.jdbc3: Permite realizar la conexión con la Base de Datos.

Hibernate3.jar: Contiene las clases principales para el uso de Hibernate.

Log4j.jar: Permite hacer trazas de nuestras aplicaciones.

Property: Provee un menú el cual permite mostrar al usuario los datos de cada una de las personas.

CONCLUSIONES

En este capítulo se presentaron los requisitos del sistema, los cuales influyeron en gran medida en la selección del patrón arquitectónico Modelo-Vista-Controlador, el cual posibilita que el sistema presente una alta independencia entre sus elementos.

Se analizaron y definieron las principales tecnologías y herramientas informáticas a emplear para el desarrollo de alasARBOGEN 2.0. Entre estas se incluyen a Java Enterprise Edition como plataforma de desarrollo, Java como lenguaje de programación y la tecnología Java TM Web Start. El Apache Tomcat 6.0 como servidor web, PostgreSQL como Sistema Gestor de Bases de Datos, y Java como lenguaje de programación.

Se definió, diseñó y describió la arquitectura planteada para el sistema alasARBOGEN2.0, donde se redefinieron las vistas arquitectónicas propuestas por la metodología utilizada con el propósito de lograr un sistema con gran modularidad e interoperabilidad.

Capítulo 3: Evaluación de la arquitectura.

Seleccionar una arquitectura indebida y evaluarla 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 de software propuesta es la más factible según las necesidades del cliente. (CERVANTES, 2010)

En el presente capítulo se describe el método de evaluación propuesto para llevar a cabo este proceso, definiendo elementos de vital importancia para el desarrollo de dicha evaluación, entre los que se encuentran los requisitos estrechamente relacionados con los atributos de calidad del software.

3.1 ASPECTOS RELEVANTES A TENER EN CUENTA PARA EVALUAR LA ARQUITECTURA

Evaluar la arquitectura de software de un sistema permite detectar aquellos atributos de calidad que serán de vital importancia para el desarrollo del mismo, además de, obtener las metas específicas de calidad ante las cuales la arquitectura será juzgada; 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 la arquitectura es un proceso iterativo e incremental que se puede realizar al final de cada ciclo. Existen dos variaciones útiles: temprana y tardía.(YOAN ARLET CARRASCOSO PUEBLA, 2009)

1. La evaluación temprana es aquella que no tiene porque esperar a que la arquitectura esté totalmente especificada. Esta puede ser utilizada en cualquier etapa del proceso de creación de la arquitectura, para examinar las decisiones arquitectónicas ya tomadas y decidir entre las opciones que están pendientes.
2. La evaluación tardía toma lugar no solo cuando hay suficiente información de la arquitectura como para justificarlo, sino también cuando la implementación está completa.

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.

3.2 EVALUACIÓN DE LA ARQUITECTURA PROPUESTA.

Para evaluar la arquitectura propuesta se tomaron en cuenta los atributos de calidad definidos por la norma ISO 9126: Funcionalidad, Confiabilidad, Eficiencia, Usabilidad, Mantenibilidad y Portabilidad; además de otros elementos como requisitos y restricciones que influyan en el desarrollo de la arquitectura. Como método de evaluación el Taller de atributos de calidad (QAW), que permite determinar la calidad para el sistema antes de ser desarrollado y la descripción de los requisitos, algo que es crucial para el éxito del sistema y para la satisfacción de los interesados, permitiendo además, ahorrar dinero y evitar realizar dichos procesos nuevamente en el futuro.

EL QAW establece 8 pasos de vital importancia (UNIVERSITY, 2010), los cuales se relacionan a continuación:

1. **Presentación e Introducción:** Los facilitadores de QAW describen la motivación del mismo y explican cada paso del método.
2. **Negocios / presentación programática:** Un representante de la comunidad de las partes interesadas presentan el negocio y/o conductores de programación para el sistema.
3. **Presentación del Plan de Arquitectura**: Un interesado técnico presenta el sistema de planos de arquitectura en su forma actual con respecto a los primeros documentos, tales como descripciones de los sistemas de alto nivel y artefactos que describen algunos de los detalles técnicos del sistema.
4. **Identificación de los conductores de Arquitectura**: Los conductores de arquitectura suelen incluir requisitos de alto nivel empresarial, se refiere a la misión, metas, objetivos, y varios atributos de calidad. Durante este paso, el grupo de desarrolladores y los interesados en el sistema llegan a un consenso acerca de que los conductores son la clave para el mismo.
5. **Escenario de tormenta de ideas**: Los interesados generan escenarios del mundo real para el sistema. Estos constituyen un estímulo relacionado, una condición ambiental, y una respuesta. Los facilitadores garantizan que al menos un escenario aborde cada uno de los controladores de arquitectura que identificó en el Paso 4.
6. **Escenario de Consolidación**: Los facilitadores deben pedir a las partes interesadas identificar los escenarios que son muy similares en contenido. Estos escenarios similares son agrupados siempre y cuando las personas que intervienen en esta selección estén de acuerdo.
7. **Escenario Priorización**: Los interesados dan prioridad a los escenarios a través de un proceso de votación.

8. **Escenario de refinamiento:** Después de la prioridad, en función de la cantidad de tiempo restante, los primeros cuatro o cinco escenarios son refinados con mayor detalle.

Durante el desarrollo del sistema se detallaron elementos como la descripción del negocio y de la arquitectura, la descripción del método de evaluación que se utilizaría, como consecuencia de esto se omitieron los pasos 1, 2, 3; procediéndose directamente a definir los conductores de Arquitectura, y a partir de ahí continuar con cada uno de los pasos que propone este método de evaluación (QAW).

3.2.1 IDENTIFICACIÓN DE LOS CONDUCTORES ARQUITECTÓNICOS.

La identificación de los conductores arquitectónicos se basa principalmente en la selección de los requisitos a tener en cuenta, la cual es una tarea clave y de gran prioridad para el éxito en la construcción de la estructura del sistema. Si se incluyen demasiados requisitos se pierde la propiedad de configuración mínima. Si por el contrario no se incluyen todos los necesarios puede influir sobre el éxito del proyecto.
A continuación se exponen los atributos de calidad, las restricciones y los requisitos funcionales más importantes, los cuales conforman los requisitos del sistema.

Atributos de calidad:

- ✓ ATRIBUTO FUNCIONALIDAD
- ✓ ATRIBUTO CONFIABILIDAD
- ✓ ATRIBUTO EFICIENCIA
- ✓ ATRIBUTO USABILIDAD
- ✓ ATRIBUTO MANTENIBILIDAD
- ✓ ATRIBUTO PORTABILIDAD
- ✓

Restricciones:

- ✓ RESTRICCIONES DE DISEÑO E IMPLEMENTACIÓN

Requisitos funcionales:

- ✓ VISUALIZAR ÁRBOL GENEALÓGICO.
- ✓ Gestionar Gráficamente un Individuo.

3.2.2 ESCENARIO TORMENTA DE IDEAS.

Con el objetivo de cerciorarse de que no estuviesen exentos ningunos de los requisitos se utilizó la lluvia de ideas, en la cual participan el grupo de desarrolladores, los clientes y los arquitectos. Esta es una técnica efectiva para la selección de requisitos. Luego de definidos los posibles requisitos a utilizar en el sistema, es necesario generar los escenarios para cada uno de estos atributos de calidad, dicha actividad es considerada un paso clave en el método QAW.

3.2.2.1 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, exactitud, seguridad de acceso y la interoperabilidad.
A continuación se exponen los escenarios propuestos para evaluar este atributo de calidad:

Escenario #1: El sistema debe cumplir con los requisitos funcionales solicitados por el cliente.

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

Escenario #3: El sistema debe poseer una infraestructura física que limite el acceso a los datos del sistema por parte de usuarios maliciosos.

3.2.2.2 ATRIBUTO CONFIABILIDAD.

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

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

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

3.2.2.3 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 expone el escenario propuesto para evaluar este atributo de calidad:

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

3.2.2.4 ATRIBUTO USABILIDAD.

La usabilidad expresa la capacidad que tiene el software para ser entendido, aprendido y operado por el usuario.

A continuación se expone el escenario propuesto para evaluar este atributo de calidad:

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

3.2.2.5 ATRIBUTO MANTENIBILIDAD

La mantenibilidad expresa la facilidad con que una modificación puede ser realizada.

A continuación se expone el escenario propuesto para evaluar este atributo de calidad:

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

3.2.2.6 ATRIBUTO PORTABILIDAD

La portabilidad de un sistema lo hacen sumamente potente. Dentro de este atributo de calidad se enmarcan características como la adaptabilidad con otros sistemas, la coexistencia y la capacidad de ser reemplazado del sistema. Este se define como la forma rápida y segura que posee un software para ejecutarse bajo cualquier plataforma, sin que estos cambios afecten las funcionalidades.

A continuación se expone el escenario propuesto para evaluar este atributo de calidad:

Escenario #1: El sistema debe permitir ser ejecutado en cualquier plataforma.

3.2.2.7 RESTRICCIONES DE DISEÑO E IMPLEMENTACIÓN.

Escenario #1: El despliegue de la aplicación de escritorio debe realizarse a través de una aplicación web.

Escenario #2: El sistema debe funcionar independientemente del estado de conexión con el RIS

3.2.2.8 VISUALIZAR ÁRBOL GENEALÓGICO.

Escenario #1: El sistema debe visualizar los árboles genealógicos de los pacientes.

3.2.2.9 GESTIONAR GRÁFICAMENTE UN INDIVIDUO.

Escenario #1 El sistema debe diseñar, visualizar y modificar los árboles genealógicos de los pacientes.

3.3 ESCENARIO DE CONSOLIDACIÓN.

Después de llevar a cabo la lluvia de ideas, los escenarios que fueron seleccionados se consolidan, de manera que no existan escenarios con similar contenido. Durante la selección se detectó la similitud de tres escenarios diferentes, los cuales se presentan a continuación con sus respectivos requisitos:

3.3.1 ATRIBUTO FUNCIONALIDAD

Escenario #1: El sistema debe cumplir con los requisitos funcionales solicitados por el cliente.

3.3.2 VISUALIZAR ÁRBOL GENEALÓGICO.

Escenario #1: El sistema debe visualizar los árboles genealógicos de los pacientes.

3.3.3 GESTIONAR GRÁFICAMENTE UN INDIVIDUO.

Escenario #1 El sistema debe diseñar, visualizar y modificar los árboles genealógicos de los pacientes.
Los requisitos expuestos coinciden en que son de tipo funcional, es decir su objetivo principal es lograr la funcionalidad del sistema. De acuerdo a un análisis realizado por los arquitectos y el grupo de interesados en el sistema, se determinó unificar los requisitos (Visualizar Árbol Genealógico y Gestionar Gráficamente un Individuo) en el escenario #1 presente en el atributo de funcionalidad, ya que este aborda de forma general los escenarios propuestos para dichos requisitos.

3.4 PRIORIZACIÓN DE ESCENARIOS.

Una vez que se tienen los escenarios por cada uno de los requisitos se procede a priorizarlos mediante la técnica de los 100 puntos (NANCY R. MEAD, 2006-2008). Se le da la posibilidad a cada implicado presente en el taller de distribuir 100 puntos entre todos los requisitos que fueron detectados dándole mayor puntuación a los más importantes teniendo en cuenta su criterio personal.
A continuación se muestran aquellos escenarios que se le determinarán sus prioridades:

	Escenarios	Aplicaciones
R1	El sistema debe cumplir con los requisitos funcionales solicitados por el cliente.	AMBAS
R2	Se debe garantizar el acceso a las acciones de forma segura en todo momento.	AMBAS
R3	El sistema debe poseer una infraestructura física que limite el acceso a los datos del sistema por parte de usuarios maliciosos.	AMBAS
R4	Los datos introducidos por el usuario deben ser los más correctos posible.	AMBAS
R5	Los registros de SISalud no responden a las solicitudes del sistema.	AMBAS
R6	El sistema debe ser entendible y fácil de usar.	AMBAS
R7	El sistema puede cambiar alguna de sus partes fácilmente.	AMBAS
R8	El sistema debe permitir ser ejecutado en cualquier plataforma.	AMBAS
R9	Los tiempos de respuesta a las peticiones de los usuarios deben ser mínimos.	AMBAS
R10	El despliegue de la aplicación Escritorio debe realizarse a través de una aplicación web.	AMBAS
R11	El sistema debe funcionar independientemente del estado de conexión con el RIS.	AMBAS

Tabla #2: Tabla de Escenarios Propuestos

Una vez realizada la votación se procede a determinar el nivel de prioridad de cada requisito a través de la matriz de cálculo de prioridad mostrada en la siguiente tabla:

# Personas	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11
Desarrollador 1	18	10	8	4	10	8	6	9	10	5	12
Desarrollador 2	20	7	9	7	8	8	8	7	8	8	10
Genetista	22	10	5	7	8	9	8	7	10	4	10
Prioridad Final	20,00	9,00	7,30	6,00	8,70	8,30	7,3	7,70	9,30	5,70	10,70

Tabla #3: Tabla de Priorización

Teniendo en cuenta el siguiente resultado se ordena de manera decreciente cada uno de los requisitos, de forma tal que la prioridad será dada de acuerdo al orden en que fueron ubicados. La siguiente tabla representa lo explicado anteriormente:

R1	20	5,00
R11	10,7	4,55
R9	9,3	4,09
R2	9	3,64
R5	8,7	3,18
R6	8,3	2,73
R8	7,7	2,27
R3	7,3	1,82
R7	7,3	1,36
R4	6	0,91
R10	5,7	0,45

Tabla #4: Tabla de valor de Prioridad

3.5 ESCENARIO DE REFINAMIENTO.

Después de conocer cada uno de los escenarios con su respectivamente prioridad, se procede a refinar los primeros cinco escenarios con mayor detalle. En la presente investigación se decidió refinarlos todos, teniendo en cuenta su prioridad.

A continuación se refinan los escenarios propuestos para el proceso de la evaluación de la arquitectura:

Escenario #1: El sistema debe cumplir con los requisitos funcionales solicitados por el cliente.

La arquitectura propuesta satisface los requisitos funcionales planteados por los clientes. Apoyándose en los servicios brindados por el RIS, el sistema posibilita gestionar gráficamente un individuo, gestionar familia, gestionar enfermedades, gestionar muestras, entre otros, además, posibilita la gestión de reportes sobre los datos generados. El sistema se concreta a las necesidades del usuario, pues no gestionará información fuera del negocio.

Escenario #11: El sistema debe funcionar independientemente del estado de conexión con el RIS.

Una de las funciones que posee el sistema es que consume servicios del RIS, mediante el cual se adquiere toda la información necesaria para el funcionamiento del sistema. Con el objetivo de lograr un sistema que se encuentre disponible en todo momento se analizaron las siguientes situaciones:

- Conectado: El sistema se encuentra conectado a la Red Nacional de Salud, de donde obtiene los datos del individuo, después de que el usuario se ha autenticado. Estará conectado además, al servidor de base de datos que permite almacenar toda la información sin ningún inconveniente.
- No Conectado: El sistema brinda la posibilidad de trabajar sin conexión. El usuario podrá introducir los datos del individuo manualmente. Toda la información generada a partir de ese momento, se guardará en un archivo dentro de la aplicación de escritorio garantizando la persistencia de la información. Luego de restablecerse la conexión, esta información es enviada al servidor de Base Datos, realizando una comparación entre los datos existentes y los que son enviados, de forma tal que no existan réplicas de la misma.

Escenario #9: Los tiempos de respuesta a las peticiones de los usuarios deben ser mínimos.

El sistema presenta un tiempo de respuesta a las peticiones relativamente rápido, gracias al mecanismo Spring AOP Cache proporcionado por Spring para interceptar las invocaciones de métodos y almacenar en memoria el resultado de operaciones como una consulta a una base de datos, reduciendo así los tiempos de respuesta a las peticiones.

El mínimo tiempo de respuesta dependerá además de las características de la red y el tipo de conexión que utilice el usuario. Se obtendrá un nivel de respuesta más lento si la conexión se establece a través del acceso telefónico a razón de 56 kbs.

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.

El comportamiento del sistema ante el intento de acceso de un usuario a una acción depende de las credenciales de este. 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.

- Si el usuario está autenticado entonces podrá realizar la operación que desee, ya sea el acceso a la aplicación Web o cualquier acción que se realice a través de la aplicación de escritorio.
- Si el usuario no está autenticado, se muestra un mensaje de error.

Escenario #5: Los registros de SISalud no responden a las solicitudes del sistema.

Si el sistema perdiera el acceso a los registros de autenticación y de ciudadano, el genetista podrá insertar los datos del individuo de forma manual, y posteriormente corregir los mismos, al verificar su autenticidad cuando se hayan restablecidos los servicios del RIS.

El objetivo es que si existiera alguna falla en estos servicios, el sistema pueda seguir operando normalmente.

Escenario #6: El sistema debe ser entendible y fácil de usar.

El sistema presenta al usuario una interfaz atractiva e interactiva, en ambas aplicaciones. La aplicación web posee una barra de herramientas que guía el desarrollo de las operaciones que se deseen realizar, así como la web presenta un menú general que facilita el proceso de búsqueda de la información requerida. Se utilizaron patrones de diseño garantizando una estructura similar en ambas aplicaciones e iconos y colores claros que estuvieran acorde con el sistema. A medida que el usuario interactúe con el sistema, más fácil y entendible será para el mismo.

Escenario #8: El sistema debe permitir ser ejecutado en cualquier plataforma.

Se definió que para implementar el sistema se utilizará como lenguaje de programación Java, ya que aparte de ser una fuente abierta, es multiplataforma lo que posibilita que el producto pueda ser instalado en cualquier cliente o servidor con sistema operativo Linux, Windows, Apple, etc.

Escenario #3: El sistema debe poseer una infraestructura física que limite el acceso a los datos del sistema por parte de usuarios maliciosos.

Los únicos archivos que pueden ser accedidos de forma directa por el cliente son los que se encuentran en el directorio de carpetas que fue instalada la aplicación de escritorio y aquellos reporte o imágenes que sean descargadas mediante la aplicación Web. Los datos del sistema pueden ser cambiados en el cliente pero existe un mecanismo de seguridad a la hora de enviar directamente esos datos al servidor, en caso de no contar con los permisos necesarios no podrá realizar dicha acción.

Escenario #4: Los datos introducidos por el usuario deben ser los más correctos posible.

Para lograr una mayor confianza sobre los datos con que opera el sistema se lleva a cabo una validación de los mismos. El funcionamiento básico de este mecanismo consiste en que si el usuario introduce datos no válidos, al enviar la información se producirá un mensaje de error, dándole la posibilidad al usuario de rectificarlo.

Escenario #10: El despliegue de la aplicación de escritorio debe realizarse a través de una aplicación web.

El despliegue de la aplicación de escritorio se realizará haciendo uso de la tecnología Java Web Start, la cual permite que esta aplicación pueda ser actualizada, instalada y descargada desde la Web.

Después de haber seleccionado cada uno de los escenarios por cada atributo de calidad, se realizó la evaluación, obteniéndose como resultado que las funcionalidades deseadas por el sistema fueron satisfactoriamente cumplidas, demostrando la eficacia de la arquitectura propuesta. El método propuesto dio como resultado una lista de atributos relacionados con la seguridad, la funcionalidad, la portabilidad, etc., y una lista de escenarios por cada uno de estos atributos, que se utilizarían como guía para el desarrollo de la arquitectura del sistema alasARBOGEN en su nueva versión.

CONCLUSIONES

En el presente capítulo se analizaron los elementos fundamentales relacionados con la evaluación de la arquitectura de software, propiciando la profundización del método propuesto para evaluar la arquitectura y de los atributos de calidad seleccionados para realizar dicho proceso de evaluación.

Para realizar la evaluación fueron propuestos 11 escenarios, los cuales fueron detallados y puestos en práctica durante el funcionamiento del sistema. Durante el desarrollo de la evaluación se determinaron 5 escenarios como los más relevantes (Escenario #1, Escenario #11, Escenario #9, Escenario #2, Escenario #5). Estos exponen situaciones relacionadas con los requisitos funcionales, la seguridad (sub-atributo de funcionalidad), con las restricciones de diseño e implementación, la eficiencia, confiabilidad y la portabilidad. Los atributos tenidos en cuenta durante la evaluación comprenden un conjunto de sub-atributos propuestos por la ISO 9126, que aunque no se hayan mencionado de forma directa se encuentran inmersos dentro de los escenarios expuestos anteriormente.

Se obtuvo como resultado de la evaluación que las funcionalidades deseadas fueron satisfactoriamente cumplidas, demostrando la eficacia de la arquitectura propuesta.

CONCLUSIONES

Con el objetivo de definir la arquitectura de software de alasARBOGEN 2.0, fueron especificados los requerimientos arquitectónicos del mismo, facilitando posibilitando la selección del patrón arquitectónico y los atributos de calidad para el sistema. Mediante la selección de las tecnologías y herramientas para dar soporte al desarrollo, se garantizó que el mismo se realizará en un entorno de desarrollo confiable y seguro.

Se describió la arquitectura propuesta para el sistema alasARBOGEN 2.0, así como las vistas arquitectónicas requeridas por la metodología RUP.

Para la evaluación de la arquitectura propuesta se utilizó el método de evaluación Taller de Atributos de Calidad (QAW), en el cual se determinaron 11 escenarios para la evaluación, resultando 5 de ellos los más relevantes para el desarrollo de la arquitectura del sistema, posibilitando así que se cumpliera con todos los requisitos especificados.

RECOMENDACIONES

- ✓ Aplicar otro método de evaluación tardía de la arquitectura con el objetivo de fortalecerla.
- ✓ Desarrollar middleware que posibiliten la interacción con los servicios del Registro Informatizado de la Salud garantizando la integridad con las distintas aplicaciones desarrolladas en el sector de la salud.

REFERENCIAS BIBLIOGRÄFÍCAS

1. ALLEN, R. CMU-CS-97-144. A formal approach to Software Architecture. s.l. : Technical Report. 1997, nº
2. BASS, C., KAZMAN. ***Software Architecture in Practice***. Editado por: Clements, K., Bass. 2003.
3. BUSCHMANN, F., MEUNIER, R., ROHNERT, H., SOMMERLAD, P., STAL, M. PATTERN. *Oriented Software Architecture. A System of Patterns. Inglaterra.* Hardcover, 1996. vol. volumen 1,
4. CARLOS REYNOSO, N. K. *De Lenguajes de descripción arquitectónica de Sofware (ADL)* version 1.0. ed. Disponible en: http://www.willydev.net/descargas/prev/ADL.pdf.
5. CERVANTES, H. *Arquitectura de Software* Software Guru, Disponible en: http://www.sg.com.mx/content/view/1004.
6. CRAIG LARMAN, P. H. *UML y Patrones: Una introducción al análisis y diseño orientado a objetos y al proceso unificado.* Segunda edición ed. 2003.
7. FOUNDATION, E. *Eclipse* Disponible en: http://www.eclipse.org/org/.
8. GONZÁLEZ, H. S. *Manual Hibernate* Disponible en: http://www.javahispano.org/contenidos/es/manual_hibernate/;jsessionid=42E8AAC25F861C19B3AB9CE9EF0B65C1.
9. GRACIA, J. *Patrones de diseño* Disponible en: http://www.ingenierosoftware.com/analisisydiseno/patrones-diseno.php.
10. GRADY BOOCH, D. G., CRIS KOBRYN, VICTORIA STAVRIDOU. *OPSLA, IS UML AN ARQUITECTURAL DESCRIPCION LANGUAGE.* Rational Software Corporation., Disponible en: http://www.sigplan.org/oopsla/oopsla99/2_ap/tech/2d1a_uml.html.
11. GRADY BOOCH, I. J., JAMES RUMBAUGH. El Proceso Unificado de Desarrollo de Software. 2000, vol. 1, nº
12. GUSTAVO ANDRÉS BREY, G. E., NICOLAS PASSERINI Y JUAN ARIAS. *Arquitectura de Proyectos de IT. Evaluación de Arquitecturas* Buenos Aires : Universidad Tecnológica Nacional: Facultad Regional de Buenos Aires- Departamento de Sistemas, Disponible en: http://apit.wdfiles.com/local--files/start/05_apit_evaluacion.pdf.
13. GUSTAVO TOROSSI, G. B., IVAR JACOBSON. Introducción a UML. 2005, nº Disponible en: http://www.slideshare.net/pegasus83/juti-introduccin-a-uml.
14. HANSON, J. *Simplify Java Object Persistence with Hibernate* Disponible en: http://www.devx.com/Java/Article/22652.
15. IEEE1471-2000. *Recommended Practice for Architectural Description of Software-Intensive Systems.* Disponible en: http://standards.ieee.org/reading/ieee/std_public/description/se/1471-2000_desc.html. .
16. ING. YAMILA VIGIL REGALADO, I. E. F. La arquitectura de software como disciplina científica. 2008, nº Disponible en: http://www.gestiopolis.com/administracion-estrategia/arquitectura-de-sofware-como-disciplina-cientifica.htm.
17. JAVA. *Tutorial Java* Disponible en: http://java.sun.com/docs/books/tutorial/.
18. JOHNSON, R. *Rob Johnson* Disponible en: http://catarina.udlap.mx/u_dl_a/tales/documentos/lis/sanchez_r_ma/capitulo3.pdf.
19. KENDALL. *Analisis y Diseño de Sistemas.* 3ra Edición ed. 2008.
20. KICILLOF, C. R., NICOLÁS. *Estilos y Patrones en la Estrategia de Arquitectura de Microsoft.*
21. versión 1.0. ed. Disponible en: http://carlosreynoso.com.ar/archivos/arquitectura/Estilos.PDF.

22. LEN BASS, P. C., RICK KAZMAN. *IEEE Recommended Practice for Arcitectural Descripcion of Software-Intensive Systems-Description* de 2004]. Descripción de la arquitectura. Disponible en: http://standards.ieee.org/reading/ieee/std_public/description/se/14'.
23. *Software Architecture in Practice.* 2ª edición ed. 2003.
24. MARIO R. BARBACCI, R. E., ANTHONY J. LATTANZE, JUDITH A. STAFFORD, CHARLES B. WEINSTOCK, WILLIAM G. WOOD. *Quality Attribute Workshops (QAWs),*
25. Third Edition ed. Architecture Tradeoff Analysis Initiative, 2003.
26. MYSQL, C. *M. MySQL* Disponible en: http://dev.mysql.com/doc/refman/5.0/en/.
27. NANCY R. MEAD, S. E. I. *Requirements Prioritization Introduction* Disponible en: https://buildsecurityin.us-cert.gov/bsi/articles/best-practices/requirements/545-BSI.html.
28. PAGÈS, B. *BOUML* Disponible en: http://bouml.free.fr/.
29. PARADIGM, C. V. ***oos Productivity with Innovative and Intuitive Technologies*** Disponible en: http://www.visual-paradigm.com/product/vpuml/editions/community.jsp.
30. PECOS, D. *PostGreSQL* Disponible en: http://danielpecos.com/docs/mysql_postgres/x15.html.
31. PETER, G. D. *PATRÓN Y ORGANIZACIÓN* Disponible en: http://www.eumed.net/libros/2009b/542/PATRON%20Y%20ORGANIZACION.htm.
32. PRESSMAN, R. *Ingeniería de Software. Un Enfoque Práctico.* Editado por: Pressman. 2002.
33. REYNOSO, C. B. *Introducción a la Arquitectura* Versión 1.0. ed. Buenos Aires: Describe la Arquitectura abordando conceptos y temas de gran importancia para el desarrollo de la arquitectura. Disponible en: http://carlosreynoso.com.ar/archivos/arquitectura/Introduccion.PDF.
34. ROD JOHNSON, J. H., ALEF ARENDSEN, COLIN SAMPALEANU, ROB HARROP, THOMAS RISBERG, DARREN DAVISON, DMITRIY KOPYLENKO, MARK POLLACK, . *The Spring Framework.* Version 2.5.4 ed. 2004-2008.
35. RODRÍGUEZ, C. R. Introducción al Java y J2EE. *Curso formativo Plataforma de formación*, 2009, nº Disponible en: http://www.docstoc.com/docs/23971437/%28Microsoft-PowerPoint---01-1-Intro-y-presentaci363n-J2EEppt%29/.
36. ROMMEL. *Tutorial. Git, Sistema de control de versiones distribuido.* Disponible en: http://www.aulati.net/?p=936.
37. SHAW, M. *"Some Patterns for Software Architecture" en Pattern Languages of Program Design.* 1996,
38. SHAW, M. Y. G., DAVID. Software Architecture: Perspectives on an Emerging Discipline. 1996., nº
39. SOFTWARE, A. F. *Apache* Disponible en: http://www.apache.org/.
40. STURM, R. *CVS Concurrent Versioning System* Disponible en: http://www.cvshome.org/eng/.
41. TIGRIS.ORG. *Open Source Software Engineering Tools.* Enterprise Edition. ed. Disponible en: http://argouml.tigris.org/.
42. TORTOISESVNTEAM, T. *TortoiseSVN The coolest Interface to (sub) Version Control.* 2004-2009, Disponible en: http://tortoisesvn.net/docs/release/TortoiseSVN_es/tsvn-intro-features.html.
43. UNIVERSITY, C. M. *Taller de atributos de calidad* Disponible en: http://www.sei.cmu.edu/architecture/consulting/qaw/.
44. WAREZ, A. *Java start web* Disponible en: http://www.argentinawarez.com/programas-gratis/688048-java-start-web.html.

45. YOAN ARLET CARRASCOSO PUEBLA, E. C. G. Y. A. C. V. Procedimiento para la evaluación de arquitecturas de software basadas en componentes 2009, nº Disponible en: http://www.gestiopolis.com/administracion-estrategia/procedimiento-para-la-evolucion-de-las-arquitecturas-de-software.htm#mas-autor.

BIBLIOGRAFÍA

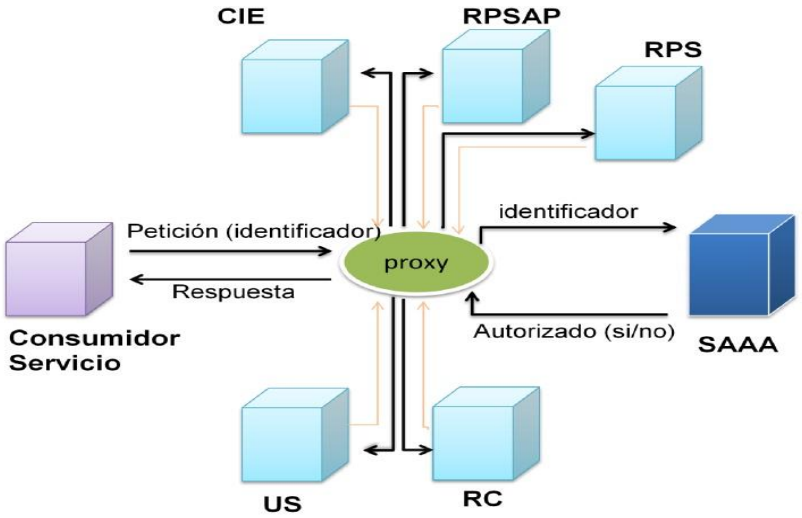
1. M. MySQL 5.0. [En línea] 2008. http://dev.mysql.com/doc/refman/5.0/en/.
2. Análisis Y Diseño De Sistemas. s.l. : 3ª. Edición Kendall & Kenda.
3. apache. [En línea] http://www.apache.org/.
4. Aplicaciones Web. [En línea] http://www.desarrolloweb.com.
5. BOUML. *BOUML.* [En línea] SOURCEFORGE.NET. http://bouml.free.fr/index.html.
6. eclipse. *eclipse.* [En línea] http://ww.eclipse.org) .
7. java. *java.* [En línea] http://java.sun.com/docs/books/tutorial/).
8. Java. [En línea] http://java.sun.com/javase/technologies/escritorio/javawebstart/1.2/es/docs/Readme_es.html.
9. **Tigris.org.** Open Source Software Engineering Tools. *Open Source Software Engineering Tools.* [En línea] COLLABNET.Enterprise Edition, 2001 - 2008. http://argouml.tigris.org/.
10. **The TortoiseSVN team.** TortoiseSVN The coolest Interface to (sub) Version Control. *TortoiseSVN The coolest Interface to (sub) Version Control.* [En línea] 2004-2009. http://tortoisesvn.net/docs/release/TortoiseSVN_es/tsvn-intro-features.html.
11. **Allen, Robert.** *CMU-CS-97-144. A formal approach to Software Architecture.* s.l. : Technical Report, 1997.
12. **Bachmann, F., y otros.** *The Architecture Based Design Method. Pittsburgh.* Carnegie Mellon University : Software Engineering Institute. CMU/SEI-2000-TR-001 ADA375851.
13. **Bass, Clements, Kazman.** Software Architecture in Practice. *Software Architecture in Practice (2nd edition).* 2003.
14. **BUSCHMANN, F., MEUNIER, R., ROHNERT, H., SOMMERLAD, P., STAL, M. Pattern.** Oriented Software Architecture. *A System of Patterns.* Inglaterra : s.n., John Wiley & Sons, 1996.
15. **Doménech, Ricardo Borillo.** *Metodologías, UML y patrones de diseño.*
16. **Grady Booch, David Garlan, Victoria Stavridou.** OPSLA, IS UML AN ARQUITECTURAL DESCRIPCION LANGUAGE. *OPSLA, IS UML AN ARQUITECTURAL DESCRIPCION LANGUAGE.* [En línea] http://www.sigplan.org/oopsla/oopsla99/2_ap/tech/2d1a_uml.html.
17. **Gustavo Andrés Brey, Gastón Escobar, Nicolas Passerini y Juan Arias.** *Arquitectura de Proyectos de IT. Evaluación de Arquitecturas.* Buenos Aires : Universidad Tecnológica Nacional. Facultad Regional de Buenos Aires- Departamento de Sistemas, 2005.
18. **Hanson, Jeff.** DevX.com. *Simplify Java Object Persistence with Hibernate.* [En línea] 8 de December de 2004. http://www.devx.com/Java/Article/22652.
19. **IEEE.** Recommended Practice for Architectural Description of Software-Intensive Systems. *Recommended Practice for Architectural Description of Software-Intensive Systems.* [En línea] http://standards.ieee.org/reading/ieee/std_public/description/se/1471-2000_desc.html. ANSI/IEEE Standart 1471-2000.
20. **Infomed.** Red Telemática de Salud en Cuba, MISAP-MIC. *Red Telemática de Salud en Cuba, MISAP-MIC.* [En línea] MISAP-MIC. http://www.sld.cu/red.
21. **Jaccobon, Buschman.** Architect Patterns. *Architect Patterns.* [En línea] 13 de Marzo de 2000. [Citado el: 06 de 02 de 2008.] http://www.rationalrose.com/models/architectural_patterns.htm..
22. **Johnson, Rob.** *Spring Framework 2005.*
23. **Kicillof, Carlos Reynoso – Nicolás.** Estilos y Patrones en la Estrategia de Arquitectura de Microsoft. *Estilos y Patrones en la Estrategia de Arquitectura de Microsoft.* [En línea] Marzo de 2004. http://carlosreynoso.com.ar/archivos/arquitectura/Estilos.PDF. versión 1.0.

24. **LEN BASS, PAUL CLEMENTS, RICK KAZMAN.** IEEE Recommended Practice for Arcitectural Descripcion of Software-Intensive Systems-Description. *IEEE Recommended Practice for Arcitectural Descripcion of Software-Intensive Systems-Description.* [En línea] 2004. http://standards.ieee.org/reading/ieee/std_public/description/se/14'.
25. **Wolf, Alexander.** ACM SIGSOFT Software Engineering Notes . *"Succeedings of the Second International Software Architecture Workshop".* 1997.
26. **UCI.** *Conferencia Ingenieria de Software I. Arquitectura y Patrones de diseño.* Ciudad de la Habana, UCI : s.n., 2007-2008.
27. **Torossi, Gustavo.** *El Proceso Unificado de desarrollo de software.*
28. **Shaw, Mary y Garlan, David.** *software Architure: Perspectives on an Emerging Discipline.* s.l. : Prentice Hall Publishing, 1996.
29. —. *Software Architecture: Perspectives on an Emerging Discipline.* s.l. : s.l. : Prentice Hall Publishing, 1996.
30. **Shaw, Mary.** *"Some Patterns for Software Architecture" en Pattern Languages of Program Design.* 1996.
31. **Shaw, Mary y Garlan, David.** *"Characteristics of Higher-Level Languages for Software Architecture".* Carnegie Mello University : Technical Report CMU-CS-94-210.
32. **Reynoso, Carlos Billy.** Introducción a la Arquitectura Versión 1.0 . *Introducción a la Arquitectura Versión 1.0 .* [En línea] BUENOS AIRES, Marzo de 2004. http://carlosreynoso.com.ar/archivos/arquitectura/Introduccion.PDF.
33. **Reynoso, Carlos y Kiccillof, Nicolás.** Estilos y Patrones en la Estrategia de Arquitectura de Microsoft. *Estilos y Patrones en la Estrategia de Arquitectura de Microsoft.* [En línea] Microsoft Developer Network, Marzo de 2004. http://www.microsoft.com/spanish/msdn/arquitectura/roadmap_arq/style.asp..
34. **R., Pressman.** Ingeniería de Software. Un Enfoque Práctico. [aut. libro] Pressman R. *Ingeniería de Software. Un Enfoque Práctico.* 2002.
35. **Bachmann, F., y otros.** *The Architecture Based Design Method.*
36. **Bass, L. y Clements, P. & Kazman, Addison Wesley Longman.** *Software Architecture in Practice. Reading.*
37. **Bass, L., Klein, M., Bachmann, F.** *Quality Attribute Design Primitives.* s.l. : CMU/SEI. CMU/SEI-2000-TN-017.
38. **Bass, L, Klein, M., Moreno, G.,.** *Applicability of General Scenarios to the Architecture Tradeoff Analysis Method.* CMU/SEI-2001-TR-014.
39. **Helm, R. y Johnson, R. & Vlissides, . Reading, MA: Addison Wesley Longman,.** *J. Design Patterns.*
40. **Hofmeister, C., Nord, R., Soni.** *Applied Software Architecture.* s.l. : Addison Wesley, 2000.
41. **Klein, M., y otros.** *Attribute-Based Architectural Styles 225-243, Proceedings of the First Working IFIP Conference on Software Architecture (WICSA1)).* San Antonio : s.n.
42. **Maceda, Humberto Cervantes.** Evaluación de la metodología de diseño arquitectural propuesta por el SEI. [En línea] http://mcyti.izt.uam.mx/Maestria_archivos/proyectos/ProyectoHumberto2.pdf.
43. **grupo de investigación de software.** La importancia de la arquitectura en el desarrollo de software de calidad . [En línea] 15 de 2004 de febrero. http://www.eafit.edu.co/NR/rdonlyres/223A8F47-27B5-4EB8-B695-4097F745D701/0/Arquitectura.pdf.

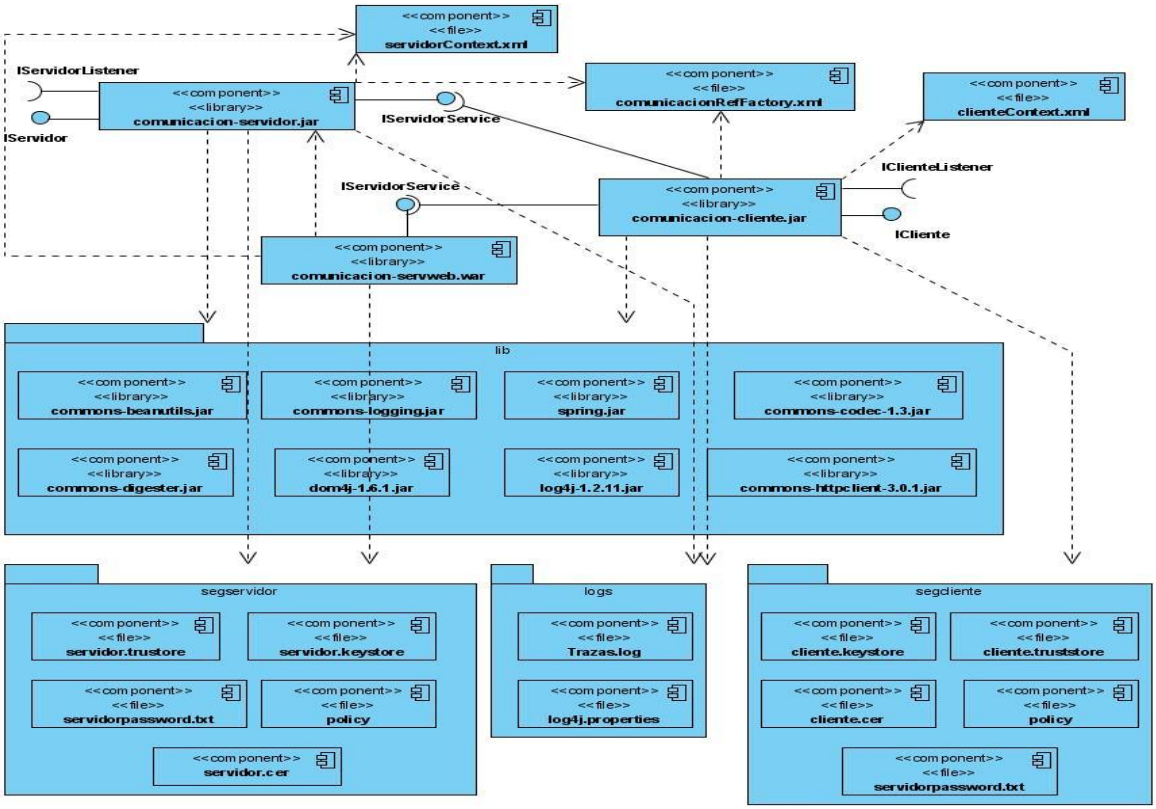
44. **Gómez, Ing. Yoan Arlet Carrascoso Puebla e Ing. Enrique Chaviano.** Propuesta de arquitectura orientada a servicios para el módulo de inventario del ERP cubano. [En línea] 15 de 05 de 2009. http://www.gestiopolis.com/administracion-estrategia/erp-arquitectura-orientada-a-servicios.htm.
45. **Erika Camacho, Fabio Cardeso.** *ARQUITECTURAS DE SOFTWARE.* Cuba : s.n., 2004.
46. **Mario R. Barbacci, Robert Ellison, Anthony J. Lattanze, Charles B. Weinstock.** *Quality Attribute Workshops (QAWs), Third Edition.* s.l. : CMU/SEI-2003-TR-016, 2003. TR-016.
47. YOAN ARLET CARRASCOSO PUEBLA, E. C. G. Y. A. C. V. Procedimiento para la evaluación de arquitecturas de software basadas en componentes 2009, nº Disponible en: http://www.gestiopolis.com/administracion-estrategia/procedimiento-para-la-evolucion-de-las-arquitecturas-de-software.htm#mas-autor.
48. RODRÍGUEZ, C. R. Introducción al Java y J2EE. *Curso formativo Plataforma de formación*, 2009, nº Disponible en: http://www.docstoc.com/docs/23971437/%28Microsoft-PowerPoint---01-1-Intro-y-presentaci363n-J2EEppt%29/.
49. NANCY R. MEAD, S. E. I. *Requirements Prioritization Introduction* Disponible en: https://buildsecurityin.us-cert.gov/bsi/articles/best-practices/requirements/545-BSI.html.
50. UNIVERSITY, C. M. *Taller de atributos de calidad* Disponible en: http://www.sei.cmu.edu/architecture/consulting/qaw/.
51. WAREZ, A. *Java start web* Disponible en: http://www.argentinawarez.com/programas-gratis/688048-java-start-web.html.

ANEXOS

Anexo 1



Anexo 2



GLOSARIO

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.

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

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.

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.

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.

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.

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.

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.

Infomed: Red Telemática de Salud en Cuba.

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.

PostgreSQL: Sistema Gestor de Base de Datos.