Universidad de las Ciencias Informáticas
Facultad 7

Título: Propuesta de arquitectura para sistemas de gestión hospitalaria

Trabajo de Diploma para optar por el título de

Ingeniero en Ciencias Informáticas

Autor(es): Mainoldis Fuentes Suárez

Alain Ramos Medina

Tutor: Lic. Maykell Sánchez Romero.

Asesor: Lic. Julio Antonio Tejera Castillo

Ciudad de La Habana, Junio del 2007

PENSAMIENTO

"Los problemas significativos que enfrentamos, no pueden ser resueltos con el mismo nivel de pensamiento que teníamos cuando lo creamos"

Albert Einstein

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 firmo la presente a los 30 días del mes de junio del año 2007.

Mainoldis Fuentes Suárez

Firma del autor

Alain Ramos Medina

Firma del autor

Lic. Maykell Sánchez Romero

Firma del tutor

DATOS DE CONTACTO

Maykell Sánchez Romero: Graduado de Licenciatura en Ciencias de la Computación de la Universidad de La Habana, profesor instructor. Ha sido Jefe de Departamento de Ciencias de la Especialidad de la facultad 7, arquitecto de la facultad 7, Jefe de Grupo de Investigación y Jefe de Proyecto.
Correo electrónico: maykell@uci.cu.

Julio Antonio Tejera Castillo: Licenciado en Educación. Especialidad Lengua Inglesa Graduado en 1999. Profesor asistente del ISP “Rafael M. de Mendive”, Pinar del Río, Cuba. Vicedecano de Residencia y Extensión de la Facultad 7 de la Universidad de las Ciencias Informáticas.
Correo electrónico: julio@uci.cu.

AGRADECIMIENTOS

A Fidel Castro Ruz y todos los que hicieron posible esta universidad.

De Alain:

A mis padres y hermano por su confianza, labor incansable y ejemplaridad.

A mis abuelitos Olegaria y Vicente, por su amor y por compartir mis sueños y alegrías.

A mi familia, por su apoyo en todos estos años.

A mi amigo Yoandry.

A todos mis amigos por estar ahí siempre, y soportarme.

A todos aquellos que contribuyeron con mi formación y la realización de esta investigación.

A todos y cada uno.

Muchas Gracias.

De Mainoldis:

A mis padres y abuelos, por educarme y hacer de mi lo que soy.

A mi hermano, por estar ahí.

A mi novia.

A mi tío Jorge, por el apoyo incondicional y contribuir en mi educación.

A Juan Antonio y su familia por acogerme en su casa y todo el apoyo que me han dado.

A mis amigos y amigas.

A mis profesores.

A todos los que ayudaron en mi formación.

A todos y cada uno.

Muchas Gracias.

DEDICATORIA

De Mainoldis:

A Keyla, Miguel y Carlos, Juan Antonio, Heisel y familia.

De Alain:

A Juan, Maria Isabel, Yasel y familia en general.

RESUMEN

En la actualidad, los hospitales generan un importante volumen de información, que resulta lento de procesar y necesario para la toma de decisiones. Tampoco cuentan con la completa automatización de sus procesos o utilizan aplicaciones no estandarizadas. Todo esto, provoca: demora, pérdida, inconsistencia, desactualización y redundancia de información.
Por lo que el objetivo de esta investigación, es diseñar una arquitectura de software que posibilite la integración entre los sistemas de gestión de la información en los hospitales. Para dar cumplimiento al mismo, se utilizan las vistas de arquitectura definidas por la metodología RUP. Además se presentan una vista modular y otra global del sistema. Para la modelación de estas, se utilizó Rational Rose haciendo uso de la notación UML.

El diseño de arquitectura propuesto, integra los estilos en Capas, Orientado a Objetos y Orientado a Servicios. En el mismo, se definen estrategias de desarrollo; mejor combinación de herramientas, atendiendo a la razón calidad del producto/tiempo de desarrollo. Además, las vías de comunicación e integración con componentes mediante la utilización de estándares internacionales. También se fundamentan patrones de diseño, arquitectura e idioma; que aportan un valor agregado considerable al sistema. Así como, la arquitectura de seguridad, acceso a datos y negocio.

El diseño propuesto, presenta una arquitectura con base sólida para el desarrollo de sistemas de gestión hospitalaria. Su implementación posibilitará facilidades de mantenimiento e integración con el resto de los componentes informáticos, que viabilizan y mejoran la atención al paciente.

PALABRAS CLAVE

Arquitectura, gestión hospitalaria.

TABLA DE CONTENIDOS

INTRODUCCIÓN

Actualmente, los sistemas de salud requieren un manejo racional y eficaz de la información. En Cuba, es una de las actividades socio–económicas de mayor importancia, y de gran sensibilidad social, por lo que se necesita elaborar sistemas con alto grado de eficiencia y eficacia en el manejo de información. Que pueda respaldar la toma de decisiones, explotando al máximo las innovaciones tecnológicas y el nivel de información de la sociedad.

Los hospitales, como actores destacados en el sistema salud, generan un importante volumen de información, por lo que se requiere: mayor tiempo para gestionar toda esa información y que los procesos estén listos para la toma de decisiones. La tecnología informática es sólo instrumental, a modo de herramienta, perfeccionando el uso de registros en papel (cuadernos, fichas, agendas, biblioratos, libros de registro).

Los esquemas tradicionales de organización jerarquizada y estática se encuentran en un período de transición, hacia una organización más abierta, dinámica, orientada a los procesos y a la información estratégica. Los errores en el diseño de los sistemas de información, la falta de formación del personal y los programas no adecuados a las necesidades son obstáculos que aunque no se presentan a menudo se deben evitarse a toda costa.

Los cambios permanentes en la tecnología de la información generan un horizonte inestable y cambiante, por eso es importante pensar en un modelo que pueda brindar continuidad y viabilidad adaptándose a las estructuras cambiantes y que sea válido frente a los distintos estilos de gestión.

En la actualidad, nuestro país cuenta con una red que brinda servicios al sistema de salud (Infomed), centrando los esfuerzos de todos sus colaboradores, en crear un ecosistema de personas, servicios y fuentes de Información para la salud. Esta red está sostenida por productos y servicios de información, comunicación y colaboración de excelencia, centrados en los usuarios y construidos con su participación activa.

En la actualidad, la informática cubana está enmarcada en la migración y desarrollo de aplicaciones de código abierto con vistas a propiciar un mayor desarrollo en el campo de la producción de software. El avance de las tecnologías de la información, cataliza de las mejoras en metodologías y arquitecturas que se utilizan para automatizar procesos en las diferentes esferas y áreas de la vida. En el caso de empresas e instituciones, se ha definido y personalizado gran cantidad de metodologías y herramientas que aportan ventajas de desarrollo e integración.

En Cuba, los procesos de gestión hospitalarias no están completamente automatizados. La mayoría de estos, se realizan de forma manual, o utilizando aplicaciones que no constituyen un software estándar para todo el país. Esto trae como consecuencia la demora, pérdida e inconsistencia de la información, además de hacer más complejo el trabajo a los empleados. Los datos de los pacientes son guardados en formularios, existentes en formato papel y llevados en algunos casos a las computadoras, que generalmente, no cumplen con los requisitos necesarios para el almacenamiento.

El problema del almacenamiento de los datos, se agrava debido a la cantidad de archivos que deben ser generados y guardados de manera organizada. Al realizarse este proceso de manera física, el aumento de la información provoca que los locales se tornen grandes y los estantes infinitos. Trayendo consigo, que las respuestas a solicitudes de información se atrasen, así como la planificación de las diferentes actividades. Todo esto provoca, duplicidad de información en varios de los departamentos del hospital, en gran parte, por el desconocimiento de la existencia de la misma, puesto que existe un gran flujo de información sin ninguna tecnología que gestione dichos procesos.

El procesamiento manual de la información, tiene muchas desventajas para los trabajadores de salud pública, impide la tenencia de registros actualizados a causa de la demora del llenado de datos y errores de otro tipo que se cometen durante este procedimiento. De igual manera, la cantidad de personas atendidas diariamente es reducida para la demanda de atención existente, en ocasiones se inscriben pacientes en horario no laboral provocando que se corra el riesgo de generar un gran volumen de información repetida, lo que conllevaría a realizar despachos entre las diferentes oficinas y demás trámites. La redundancia de información origina, que en la última historia clínica elaborada a un paciente no existan datos anteriores debido a que se crea una nueva, esto impide el seguimiento de su evolución

en relación a alguna patología. Incluyendo la pérdida de tiempo al llenar nuevamente los mismos formularios; asimismo surgen problemas con la ubicación de su cama en el hospital y malgasto de materiales.

La información sufre desactualización debido a que en algunos casos las personas encargadas de transmitirla deben trasladarse con documentos de un local a otro por lo que demora el proceso de atención al paciente.

La atención sanitaria de un paciente es la responsabilidad compartida de un grupo de profesionales pertenecientes a diversas disciplinas e instituciones. Como consecuencia, es vital que estas puedan compartir información sobre los pacientes, de una manera sencilla, segura y conservando el significado original de los datos. Actualmente la información sanitaria está repartida en múltiples sistemas de información heterogéneos y autónomos, por tanto, el acceso uniforme a los registros clínicos es una tarea problemática. Todo esto se debe, mayormente, a que la elección de la arquitectura de software para la construcción del sistema que informatiza los distintos servicios del hospital, difiere de la seleccionada por los otros o simplemente posee dificultades de integración.

Para elegir la arquitectura correcta se ha tenido en cuenta primeramente que un modelo de arquitectura de software consiste en tres componentes: elementos, forma y razón. Los elementos son de procesamiento, datos o conexión. La forma se define en términos de las propiedades y las relaciones entre los elementos, es decir, restricciones operadas sobre ellos. La razón proporciona una base subyacente para la arquitectura en términos de las restricciones del sistema, que lo más frecuente es que se deriven de los requerimientos del sistema.
La Arquitectura de Software es considerada como disciplina por mérito propio, resulta beneficiosa como marco de referencia para satisfacer los requerimientos. Además, el arquitecto debe llevar a cabo cuidadosamente y con calidad, el diseño o selección, representación, evaluación, análisis y recuperación de la misma; tiene que garantizar un desarrollo y evolución basados estrictamente en ella.

Se cuenta en estos momentos, con el sistema “Galen Hospital” versión 2.0, desarrollado en plataforma Windows 32 bits (Windows NT y Windows 98) con una configuración cliente/servidor y el uso del gestor de

bases de datos relacional SQL Server 7.0 como reservorio de la información, que aunque posee algunos módulos funcionando, es un sistema que se encuentra desarrollado enteramente utilizando software propietario (Visual Basic 6.0, Delphi y SqlServer), lo que implica costos de desarrollo e instalación, en muchos casos elevados.

Desde el punto de vista arquitectónico dicho sistema, construido bajo el estilo cliente/servidor es un sistema creado como isla de información con beneficios muy limitados; dificultando en gran medida la integración con aplicaciones como: el Sistema de Información Estadística Complementario de Salud(SIE-C salud), el Sistema de nefrología, los cuales deben nutrirse de información proveniente de este sistema y debido a este inconveniente, han sido diseñados y están siendo implementados para que la entrada de datos que necesitan de los hospitales sea procesada manualmente.

Es una aplicación de escritorio por lo que los procesos de despliegue, instalación y mantenimiento se dificultan cuando hablamos de grandes cantidades de ordenadores, pues dichos procesos requieren en la mayoría de los casos, que los cambios sean realizados en cada una de las computadoras clientes.

Después de explicar la situación problémica se determina el siguiente **problema científico:**
¿Cómo diseñar una arquitectura que permita la integración entre los sistemas de gestión de información hospitalarios desarrollados en el Proyecto Hospitales y componentes obtenidos en la informatización del Sistema Nacional de Salud Cubano?

La investigación se enmarca en el **objeto de la investigación:** Procesos de gestión de la información hospitalaria. Además el **campo de acción** determinado se inscribe en los procesos de gestión de la información en los hospitales cubanos.

Para desarrollar la investigación se plantea el siguiente **objetivo general:**
Diseñar una arquitectura de software que posibilite la integración entre los sistemas de gestión de la información en los hospitales cubanos.

Las **tareas de la investigación** son:

1. Investigar el estado del arte de los procesos de gestión de la información hospitalaria.
2. Determinar las funcionalidades que debe tener el sistema.
3. Definir la estrategia de desarrollo con el objetivo de elegir y aplicar la más óptima de acuerdo al tipo de sistema y proyecto.
4. Definir herramientas a utilizar para el desarrollo del proyecto, para lograr la mejor relación tiempo de desarrollo – calidad del producto.
5. Fundamentar la utilización y aplicación de los patrones.
6. Definir las estrategias de desarrollo, comunicación e integración con otros componentes ya existentes, para minimizar los cambios necesarios en las aplicaciones, de manera tal que se logre una integración exitosa, un desarrollo eficaz y veloz del sistema.
7. Diseñar la arquitectura de seguridad, acceso a datos y del negocio.

El presente documento está estructurado en tres capítulos:

- Capítulo 1: presenta conceptos asociados al dominio del problema, estado actual del objeto de estudio, experiencias precedentes de sistemas o Frameworks y un análisis de las tendencias, herramientas y tecnologías de actualidad.
- Capítulo 2: describe los patrones de arquitectura, diseño e idioma presentes en la arquitectura propuesta.
- Capítulo 3: expone la línea base de la arquitectura y el desarrollo del documento de descripción de la arquitectura propuesto por la metodología RUP.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

Existen conceptos asociados a los procesos de gestión de la información hospitalaria que pueden resultar desconocidos y son de vital importancia para la comprensión de la investigación. Ayuda en este tema, el conocimiento del estado actual de estos, el análisis y comparación de experiencias precedentes, tendencias, metodologías, tecnologías y herramientas de actualidad que fundamentan la selección que se presenta en capítulos posteriores.

1.1 Conceptos Asociados

En los *sistemas* como entorno integrado de datos comunes la información se origina en diferentes unidades funcionales independientes y es compartida por todos; o bien como entorno distribuido en el cual cada unidad funcional administra y procesa los datos de interés local, así como los sistemas de uso común. Incluye, centrando en el área de la informática: (a) diseño de sistemas (ingeniería de sistemas), (b) manejo de procesos (usuarios de los sistemas), (c) bases de datos (con gran proyección en la investigación científica y en los programas de salud pública) y (d) diseños de seguridad [1].

La *información* es definida como comunicación o adquisición de conocimientos que permiten ampliar o precisar el conocimiento que se posee sobre una materia determinada; el mismo posee un valor incontable. Debido a la importancia que tiene un manejo eficaz y seguro de esta, sobre todo cuando nos enfocamos en el sector de salud, surgen los *sistemas de información hospitalaria**,*** los cuales basan su razón de ser en la recolección, almacenamiento, procesamiento, recuperación y comunicación de información de atención al paciente y administrativa, para todas las actividades relacionadas con el hospital. Dichos sistemas como sistemas de gestión incluyen además: registros de actividad, indicadores estratégicos (por mes, por año), recursos humanos y recursos materiales [2].

La *Arquitectura de Software (AS*) tiene sus antecedentes en la década de 1960, su historia no ha sido tan continua como la de la ingeniería de software. Después de las tempranas inspiraciones del legendario Edsger Dijkstra, de David Parnas y de Fred Brooks, la AS quedó en estado de vida latente durante unos cuantos años, hasta comenzar su expansión explosiva con los manifiestos de Dewayne Perry de AT&T Bell Laboratories de New Jersey y Alexander Wolf de la Universidad de Colorado. Se considera que Perry y Wolf fundaron la disciplina, y su llamamiento fue respondido en primera instancia por los miembros de lo

que podría llamarse la escuela estructuralista de Carnegie Mellon: David Garlan, Mary Shaw, Paul Clements y Robert Allen.

Se trata entonces, de una práctica joven, de apenas unos doce años de trabajo constante, que en estos momentos experimenta una nueva ola creativa en el desarrollo cabal de sus técnicas en la obra de Rick Kazman, Mark Klein, Len Bass y otros metodólogos del Instituto de Ingeniería de Software (SEI), en la misma universidad. A comienzos del siglo XXI, comienzan a discernirse tendencias, cuyas desavenencias mutuas todavía son leves: al menos una en el sur de California (Irvine y Los Ángeles) con Nenad Medvidovic, David Rosenblum y Richard Taylor, otra en el Instituto de Investigaciones de Software (SRI) de Menlo Park con Mark Moriconi y sus colegas y otra más vinculada a las recomendaciones formales del Instituto de ingenieros eléctricos y electrónicos (IEEE) y los trabajos de Rich Hilliard.

Hoy se percibe también un conjunto de posturas europeas que enfatizan mayormente cuestiones metodológicas vinculadas con escenarios y procuran inscribir la arquitectura de software en el ciclo de vida, comenzando por la elicitación de los requerimientos. Antes de Perry y Wolf, se formularon ideas que serían fundamentales para la disciplina ulterior. [3]

Los años transcurridos desde la primera disertación de AS, han servido para recopilar innumerables definiciones del tema, las cuales se agrupan en cuatro grandes grupos: modernas, clásicas, bibliográficas y las que brindan personas dedicadas al tema, agrupadas en la comunidad del Instituto de Ingeniería de Software “Carnegie Mellon”. La mayoría poseen puntos de coincidencia que giran alrededor de: elementos de software, componentes, propiedades y relaciones. A continuación enumeramos algunas de ellas, sugiriendo fijar la atención en la enunciada por el Instituto de Ingenieros eléctricos y electrónicos:

1. La AS de un programa o sistema computacional es la estructura de estructuras del sistema, la cual comprende: elementos de software, propiedades externamente visibles de esos elementos y relaciones entre ellos [4].
2. La AS es la organización fundamental de un sistema encarnada en sus componentes, las relaciones entre ellos, el ambiente y los principios que orientan su diseño y evolución (ANSI/IEEE Std 1471-2000).[5]

En la ingeniería de software al igual que en la construcción; los estilos arquitectónicos describen categorías, conjunto de componentes, cooperación entre ellos, conectores y modelos, por lo que pudiésemos definir los *estilos arquitectónicos* como conceptos descriptivos que definen una forma de articulación u organización arquitectónica. El conjunto de los estilos cataloga las formas básicas posibles de estructuras de software, mientras que las formas complejas se articulan mediante composición de los estilos fundamentales [6].

Según señalan Mary Shaw y Neno Medvidovic en sus revisiones de los lenguajes de descripción arquitectónica, el uso que se da en la práctica y en el discurso a conceptos tales como arquitectura de software o estilos suele ser informal y *ad hoc*. En la práctica industrial, las configuraciones arquitectónicas se suelen describir por medio de diagramas de cajas y líneas, con algunos añadidos diacríticos; los entornos para esos diagramas suelen ser de espléndida calidad gráfica, comunican con relativa efectividad la estructura del sistema y siempre brindan algún control de consistencia respecto a que clase de elemento se puede conectar con otro; pero a la larga proporcionan escasa información sobre la computación efectiva representada por las cajas, las interfaces expuestas por los componentes o la naturaleza de sus computaciones.[7]

En la década de 1990 y en lo que va del siglo XXI, se han materializado diversas propuestas para describir y razonar en términos de arquitectura de software; muchas de ellas han asumido la forma de *lenguajes de descripción arquitectónica (ADLs)*. Estos ADLs permiten modelar una arquitectura mucho antes que se lleve a cabo la programación de las aplicaciones que la componen, analizar su adecuación, determinar sus puntos críticos y eventualmente simular su comportamiento. [8]
Además suministran construcciones para especificar abstracciones arquitectónicas y mecanismos para descomponer un sistema en componentes y conectores, especificando de qué manera estos elementos se combinan para formar configuraciones y definiendo familias de arquitecturas o estilos. Contando con un ADL, un arquitecto puede razonar sobre las propiedades del sistema con precisión, pero a un nivel de abstracción convenientemente genérico. Algunas de esas propiedades podrían ser protocolos de interacción, anchos de banda y latencia, localización del almacenamiento, conformidad con estándares arquitectónicos y previsiones de evolución ulterior del sistema. [9]

En la actualidad, la mayoría de las herramientas, que se utilizan para el modelado en la rama informática, cuentan como base o lenguaje de modelado al *lenguaje unificado de modelado (UML)*. Este es un lenguaje de modelado visual que comenzó a desarrollarse a partir de 1994, puesto que existían diversos métodos y técnicas orientadas a objetos, con muchos aspectos en común pero utilizando distintas notaciones. Producto de esto se presentaban inconvenientes para el aprendizaje, aplicación, construcción y uso de herramientas, etc., además de pugnas entre enfoques, lo que generó la creación del UML como estándar para el modelado de sistemas de software, pero con posibilidades de ser aplicado a todo tipo de proyectos [10].

UML es usado para especificar, visualizar, construir y documentar artefactos de un sistema de software, para entender, diseñar, configurar, mantener y controlar la información sobre los sistemas a construir. Este capta la información sobre la estructura estática y el comportamiento dinámico de un sistema, el cual se modela como una colección de objetos discretos que interactúan para realizar un trabajo que, finalmente, beneficia a un usuario externo. El lenguaje de modelado pretende unificar la experiencia sobre técnicas de modelado pasadas e incorporar las mejores prácticas actuales en un acercamiento estándar. UML no es un lenguaje de programación, pues las herramientas pueden ofrecer generadores de código de UML para una gran variedad de lenguajes de programación, así como construir modelos por ingeniería inversa a partir de programas existentes [11].

El término *servicios web* (Web Services) describe una forma estandarizada de aplicaciones usando XML, SOAP, WSDL y UDDI (estándar abierto), sobre el protocolo Internet. XML es usado para etiquetar los datos, SOAP es el encargado de la transferencia de la información, WSDL describe los servicios disponibles y UDDI es el responsable de que los servicios web disponibles sean listados. Fueron usados en sus inicios como vía de comunicación entre negocios y con los clientes, los servicios web permiten a las organizaciones el intercambio de datos sin la necesidad de conocer los sistemas que se encuentran detrás del cortafuego (Firewall).

A diferencia de los modelos tradicionales cliente-servidor, como los sistemas servidor web/página web, los servicios web no proveen al usuario de una interfaz gráfica, en lugar de esa interfaz comparten lógica de

negocio, datos y procesos a través de interfaces programáticas por la red. Los desarrolladores pueden entonces añadir un servicio web a una interfaz gráfica de usuario (páginas web o programas ejecutables), con el objetivo de ofrecer funcionalidades a los usuarios [12]. Llegando a conceptualizar los *servicios web* como el conjunto de aplicaciones o de tecnologías con capacidad para interoperar en la WEB. Estas aplicaciones o tecnologías intercambian datos entre sí, utilizando estándares de comunicación, con el objetivo de ofrecer unos servicios. Los proveedores ofrecen sus servicios como procedimientos remotos y los usuarios solicitan un servicio llamando a estos procedimientos a través de la WEB.

Los navegadores web (browsers) juegan un papel importante dentro de las aplicaciones web ya que desempeñan en papel de clientes. Estos son aplicaciones empleadas para buscar y mostrar páginas web. Dentro de estos, en la actualidad los browsers gráficos impactan rápida y eficazmente, dado el amplio soporte que poseen para mostrar información multimedia, incluyendo sonido y video. Existen hoy día un sin número de navegadores, por mencionar algunos ejemplos podemos destacar a: NetScape Navigator, Internet Explorer, Firefox, Safari, Opera, Netscape y AOL; siendo los dos más populares: Internet Explorer 6 y Firefox.

En informática, las aplicaciones son programas con los cuales se interactúa mediante interfaces. Dentro de estos programas podemos ubicar dos grandes grupos: las aplicaciones desktop o de escritorio y las a*plicaciones web.* Estás últimas son aplicaciones informáticas que los usuarios utilizan accediendo a un servidor web a través de Internet o de una intranet. Las aplicaciones web son populares debido a la practicidad del navegador web como cliente ligero.

La habilidad para actualizar y mantener aplicaciones web sin distribuir e instalar software en miles de potenciales clientes es otra razón de su popularidad. Estas aplicaciones constituyen un caso más complejo de esta plataforma. Actualmente las aplicaciones web cuentan con gran popularidad debido al auge que tuvo Internet en la década pasada, y dado que brindan grandes posibilidades de despliegue en oposición a las aplicaciones de escritorio, por el simple hecho que los usuarios solo necesitan un navegador para hacer uso de ellas. Además, estas aplicaciones permiten el aprovechamiento de todas las características de Internet, estando disponibles desde cualquier parte del mundo donde se tenga acceso

a la Red de Redes. Son fáciles de usar, ya que no requieren conocimientos avanzados por parte de los usuarios.

Las aplicaciones web se ejecutan en el servidor, encargándose de controlar el estado de un negocio o una problemática dada mediante el uso de uno o varios lenguajes de programación. De esta forma se generan páginas dinámicas cuyo contenido la mayor parte de las veces está vinculado con los datos almacenados en bases de datos relacionales Estos servidores donde son almacenadas son comúnmente llamados: *servidores web*, éstos sirven contenido estático a un navegador, carga un archivo y lo sirve a través de la red al navegador de un usuario. Este intercambio es mediado por el navegador y el servidor que hablan el uno con el otro mediante HTTP. Se pueden utilizar varias tecnologías en el servidor para aumentar su potencia más allá de su capacidad de entregar páginas HTML; estas incluyen scripts CGI, seguridad SSL y páginas activas del servidor (ASP).

De forma sencilla, un *sistema de gestión de bases de datos* (SGBD) se puede definir como una colección de datos interrelacionados y un conjunto de programas para acceder a esos datos. Adoración de Miguel lo define como “conjunto coordinado de programas, procedimientos, lenguajes, etc. que suministra, tanto a los usuarios no informáticos como a los analistas, programadores o al administrador, los medios necesarios para describir, recuperar y manipular los datos almacenados en la base, manteniendo su integridad, confidencialidad y seguridad [13].

En la actualidad los SGBD se pueden agrupar en dos grandes modelos:

- Sistemas de Gestión de Bases de Datos Relacionales (SGBDR)

  Las bases de datos que generan se construyen con información muy estructurada (datos) acerca de una organización o empresa determinada. Cuando un usuario realiza una consulta en una base de datos relacional, el sistema presenta como resultado la respuesta exacta a lo que se busca. A este tipo de bases de datos se les denomina *bases de datos relacionales,* y a los sistemas que las gestionan, Sistemas de Gestión de Bases de Datos Relacionales (SGBDR) [14].

- Sistemas de Gestión de Bases de Datos Documentales (SGBDD) o Sistemas de Recuperación de Información (SRI)

  Las bases de datos que generan se construyen con información no estructurada tipo texto (documentos) sobre uno o varios temas. Cuando un usuario realiza una consulta en una base de datos documental, el sistema presenta como resultado, no una respuesta exacta, sino documentos útiles para satisfacer la pregunta del usuario.

  A este tipo de bases de datos se les denomina *bases de datos documentales*, y a los sistemas que las gestionan, Sistemas de Gestión de Bases de Datos Documentales (SGBDD) o Sistemas de Recuperación de Información (SRI). [15]

En años recientes, la disponibilidad de las bases de datos y de las redes de computadoras ha promovido el desarrollo de un nuevo campo denominado bases de datos distribuidas. Una *base de datos distribuida* es una base de datos integrada la cual se construye por encima de una red de computadoras en lugar de una sola computadora. Las bases de datos distribuidas ofrecen diversas ventajas a los diseñadores y usuarios de bases de datos. Entre las más importantes se encuentra la transparencia en el acceso y localización de información. Sin embargo, el diseño y administración de bases de datos distribuidas constituye un gran desafío que incorpora problemas no encontrados en bases de datos centralizadas. Por ejemplo, los esquemas de fragmentación y localización de información, el manejo de consultas a sitios distribuidos y los mecanismos de control de concurrencia y confiabilidad en bases de datos distribuidas [16].

Sería válido también dar algunas razones tecnológicas que justifican la descentralización de la información: Permitir autonomía local y promover la evolución de los sistemas y los cambios en los requerimientos de usuario; proveer a los sistemas de una arquitectura simple, flexible y tolerante a fallas; la tercera razón es la de ofrecer buenos rendimientos; necesario en los sistemas de gestión dada la cantidad de información que se maneja y la rapidez de respuesta que se requiere.

Cuando los proyectos que se van a desarrollar son de mayor envergadura, ahí si toma sentido el basarnos en una *metodología de desarrollo*, y empezamos a buscar cual sería la más apropiada para nuestro caso. Lo cierto es que muchas veces no encontramos la más adecuada y terminamos por hacer o diseñar

nuestra propia metodología, algo que por supuesto no está mal, siempre y cuando cumpla con el objetivo [17]. En la actualidad se pueden identificar una gran gama dentro de las cuales se pueden destacar tres, siendo a consideración de disímiles autores las mas importantes: RUP, XP, MSF.

La *metodología* Rational Unified Process (RUP), divide en 4 fases el desarrollo del software: inicio, elaboración, construcción y transición. Cada una de estas etapas es desarrollada mediante el ciclo de iteraciones, la cual consiste en reproducir el ciclo de vida en cascada a menor escala. Los objetivos de una iteración se establecen en función de la evaluación de las iteraciones precedentes. Vale mencionar que el ciclo de vida que se desarrolla por cada iteración, es llevada bajo dos disciplinas [18].

La Disciplina de Desarrollo**,** la cual circunscribe la ingeniería de negocios, los requerimientos, el análisis y diseño, implementación y pruebas y la Disciplina de Soporte que engloba la configuración y administración del cambio, administración del proyecto, administración del ambiente de desarrollo y la distribución del producto.

Además esta metodología consta de los siguientes elementos: actividades, trabajadores y artefactos. Una particularidad de esta es que, en cada ciclo de iteración, se hace exigente el uso de artefactos, siendo por este motivo, una de las metodologías más importantes para alcanzar un grado de certificación en el desarrollo del software.

Desde la etapa más temprana de concepción del software, encontramos *patrones*, estos abundan en un orden de magnitud de tres dígitos, llegando a cuatro y gana más puntos quien más variedades enumere. Los sub-estilos se han congelado alrededor de la veintena, agrupados en cinco o seis estilos mayores y se considera mejor teórico a quien los subsume en el orden más simple. Aún cuando los foros de discusión abundan en pullas de los académicos por el desorden proliferante de los patrones y en quejas de los implementadores por la naturaleza abstracta de los estilos arquitectónicos, desde muy temprano hay claras convergencias entre ambos conceptos, aunque se reconoce que los patrones se refieren más bien a prácticas de re-utilización y los estilos conciernen a teorías sobre la estructuras de los sistemas a veces más formales que concretas [19].

Los patrones pueden dividirse o clasificarse en *patrones de arquitectura* son los relacionados con la interacción de objetos dentro o entre niveles arquitectónicos, problemas arquitectónicos, adaptabilidad a requerimientos cambiantes, rendimiento, modularidad y acoplamiento. Los *patrones de diseño* fueron construidos dado los problemas con la claridad de diseño, multiplicación de clases, adaptabilidad a requerimientos cambiantes, proponiendo como solución a estos problemas el comportamiento de factoría, clase-responsabilidad-contrato (CRC*),* etc. Los *patrones de análisis* son usualmente específicos de aplicaciones. Los *patrones de proceso u organización* tratan todo lo concerniente con el desarrollo o procesos de administración de proyectos, técnicas y estructuras de organización y finalmente, los *patrones de idioma* reglan la nomenclatura en la cual se escriben, se diseñan y desarrollan nuestros sistemas.

1.2 Estado actual de los procesos de gestión de la información hospitalaria

En la década de los 70's, se perfilan los primeros sistemas de información médica que, posteriormente, habrán de dar lugar a los sistemas de información hospitalaria, tan indispensables en la actualidad. Su impacto en las instituciones de salud es notable, ya que buscan elevar la calidad de la atención del paciente, de los servicios brindados y aplicar la información obtenida a las áreas de la investigación, la clínica, la docencia, la administración y desde luego abatir costos y elevar la productividad [20].

De los sistemas de información hospitalaria (HIS) existen disímiles productos y versiones pertenecientes a varias empresas de desarrollo de software. En Cuba, se han hecho intentos, aunque no existe la suficiente difusión e intercambio de experiencias como para realizar y estandarizar un producto que cubra las expectativas de los centros de salud existentes, ejemplo de estos intentos tenemos el “Galen Hospital” y el “Medisys”, ambos sistemas implantados y funcionando, pero con posibilidades remotas de integración y estandarización.

De estos sistemas de información hospitalaria podemos mencionar algunos que figuran en las listas mundiales, estos son: “Hosix–V” **,** **“**Care2x”, “MedFile 5.x”, “Request for proposal (RFP)”, “C-DAC Sushrut”, “Armed VMD”, “Archimed”, “Proyecto Hospital Sin Paredes”, “Debian-Med”, “Hosxp”, “Vitality Medical”, “The Capterra Medical Scheduling Software”, “Biocom”, “Caduceo – rips”, “FGKP Software”, “IGM”, “INFORMED” y “Cerner Millennium”.

En la actualidad, aunque no se cuenta con una definición exacta de arquitectura, si poseemos estilos arquitectónicos bien definidos que podemos utilizar para lograr una integración entre los sistemas existentes y dar paso a la construcción de sistemas bien centralizados que reutilicen sistemas que en la actualidad constan con una reputación aceptable, pero dado los inconvenientes de una arquitectura diseñada, en muchos casos para garantizar una dependencia absoluta del sistema y de la firma, obligan a reinventar la rueda. Se habla de lograr una combinación exitosa entre estilos de arquitectura: en capas, orientados a objetos, basados en componentes, *peer to peer* y orientados a servicios; recomendando un especial detenimiento en el ultimo estilo propuesto, dado que este último es la mejor propuesta para fundamentar la integración entre sistemas.

Estos sistemas, en su mayoría diseñados e implementados usando el estilo arquitectónico cliente/servidor en tres capas, han sido desarrollados utilizando software privativo. Lo que eleva en gran medida los costos de producción e implantación, además de imposibilitar la reutilización de estas experiencias en nuevas versiones, atentando entre otras cosas contra la estandarización. Se puede decidir dentro de estas posibilidades, por aplicaciones web y de escritorio, aunque dentro de estas se presentan algunos impedimentos.

Los sistemas que poseen interfaces web, presentan en algunos casos incompatibilidades con los navegadores que en la actualidad encabezan la lista de los más utilizados, o aquellos que posibilitarían la explotación de la aplicación en clientes que usen sistemas operativos basados en la plataforma GNU-LINUX. Sumándose aquellos sistemas consistentes en aplicaciones de escritorio compatibles solo sistemas operativos Windows.
Las inconformidades de los clientes surgen al estar restringidos en su mayoría a utilizar sistemas operativos que acarrean costos adicionales en licencias. Además, de todo lo que implica un cambio de sistema operativo y tecnología: “MedFile 5.x”, “Galen Hospital”, “MediSys” y “Archimed”, los casos mencionados cuentan con escasas posibilidades de integrarse con otros sistemas existentes o por desarrollar.

1.3 Experiencias precedentes

La meta es construir un HIS estandarizado donde el paciente sea el más beneficiado, y los profesionales de la salud encuentren en estos sistemas un recurso idóneo, amigable y flexible que responda a las necesidades de información de la institución hospitalaria o de salud y que además no solo posibilite su integración con sistemas existentes sino que otros sistemas puedan ser diseñados e implementados contando con las funcionalidades que brinda.

Un ejemplo fehaciente de un sistema que ha dado los primeros pasos en el sentido propuesto, es el Care2X, sistema implementado utilizando herramientas libres y en el cual podemos encontrar un adelanto a la solución que se persigue, este sistema utiliza el protocolo de intercambio de datos *Health Exchange Protocol* (HXP), este protocolo está siendo usado por Care2x para comunicarse transparentemente con otras aplicaciones independientemente de sus plataformas. HXP hace intercambio de datos entre aplicaciones para la salud, además de ser simple de implementar y de comprender, independiente de plataforma, libre o de muy bajo costo, confiable, seguro con autenticación, encriptado con protocolo SSL, flexible, transparente, libre de restricciones geográficas y código libre[21]. Consiste en un mensaje en formato XML y un diccionario de llamadas a procedimientos que cuenta con la descripción de la llamada al procedimiento, el propósito, descripción de los parámetros, tipo y valores necesarios, tipos de dato y valores de la respuesta, descripción de los mensajes de error y su significado.

El *estándar HCE ENV13606* utilizado en la construcción de Historias Clínicas Electrónicas, permite la formalización de la información clínica (semántica clara): integración semántica, aislar a los usuarios (profesionales/maquinas) de la complejidad de las fuentes de datos, que los profesionales sanitarios puedan trabajar con conceptos propios de su dominio (arquetipos), crear sistemas flexibles de búsqueda de información, facilita la investigación clínica, la creación de almacenes de datos (datawarehouse, datasmart, repositorios temporales), dichos datos son almacenados como árboles con referencias con nodos etiquetados, lo cual lo convierte en un modelo compatible con XML.

Un ejemplo interesante del uso de este estándar es *Pangea***:** Middleware de integración de datos médicos basado en un modelo dual de arquitectura de información para la comunicación de historias clínicas electrónicas, que ofrece una vista global integrada de la información sobre la salud de una persona

repartida por múltiples sistemas compatible con la prenorma ENV13606 del Comité Europeo de Normalización(CEN), sirve la información (extractos de historias clínicas electrónicas) como mensajes a través de varios servicios web y un mecanismo de petición/respuesta síncrono basado en XML. Este middleware puede ser utilizado a la par por varias aplicaciones permitiendo la integración entre ellas.

1.4 Tendencias, Herramientas y Tecnologías

1.4.1 Estilos arquitectónicos

La relación de estilos arquitectónicos, que a continuación se muestra, no incluye a todos los que existen. Se mencionan, aquellos los más utilizados en la actualidad y que han sido escogidos para conformar la arquitectura que se propone.

Los estilos se agrupan en clases, las que engloban una serie de estilos arquitectónicos que comparten características.

- Estilos de flujo de datos
  - Tubería y filtros: El sistema tubería-filtros se percibe como una serie de transformaciones sobre sucesivas piezas de los datos de entrada. Los datos entran al sistema y fluyen a través de los componentes. Los componentes son programas independientes; el supuesto es que cada paso se ejecuta hasta completarse antes que se inicie el paso siguiente [22].
- Estilos centrados en datos
  - Arquitecturas de pizarra o repositorio: Utiliza una estructura de datos para representar el estado actual y una colección de objetos independientes que operan sobre él como componentes principales. Estos sistemas se han usado en aplicaciones que requieren complejas interpretaciones de proceso de señales (reconocimiento de patrones, reconocimiento de habla, etc.), o en sistemas que involucran acceso compartido a datos con agentes débilmente acoplados [23].
- Estilos de llamada y retorno
  - Model-View-Controller (MVC): El patrón conocido como Modelo-Vista-Controlador (MVC) separa el modelado del dominio, la presentación y las acciones basadas en datos ingresados por el usuario en tres clases diferentes, el modelo que se encarga de administrar el comportamiento y los datos del dominio de aplicación, responde a requerimientos de información sobre su estado y responde a

instrucciones de cambiar el estado (habitualmente desde el controlador); la vista que maneja la visualización de la información y el controlador que interpreta las acciones del ratón y el teclado, informando al modelo y/o a la vista para que cambien según resulte apropiado.

- Arquitecturas en capas: Definida por Garlan y Shaw 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 [24].

  Las interacciones entre las capas ocurren generalmente por invocación de métodos, por definición los niveles mas bajos no deben poder utilizar funcionalidad ofrecida por las capas superiores. Este estilo facilita la modularidad del sistema, la localización de errores y mejora considerablemente el soporte del sistema.
- Arquitecturas orientadas a objetos: Los componentes del estilo se basan en principios orientados a objetos (OO) como encapsulamiento, herencia y polimorfismo. Son asimismo las unidades de modelado, diseño e implementación, y los objetos y sus interacciones son el centro de las incumbencias en el diseño de la arquitectura y en la estructura de la aplicación; las interfaces están separadas de las implementaciones; en cuanto a las restricciones, puede admitirse o no que una interfaz pueda ser implementada por múltiples clases. Entre las cualidades de este estilo, la más básica concierne a que se puede modificar la implementación de un objeto sin afectar a sus clientes. Asimismo es posible descomponer problemas en colecciones de agentes en interacción. Además, por supuesto (y esa es la idea clave), un objeto es ante todo una entidad reutilizable en el entorno de desarrollo [25].
- Arquitecturas basadas en componentes: Los sistemas de software basados en componentes se basan en principios definidos por una ingeniería de software específica [26]. Los componentes son las unidades de modelado, diseño e implementación, las interfaces están separadas de las implementaciones, y las interfaces y sus interacciones son el centro de incumbencias en el diseño arquitectónico, los componentes soportan algún régimen de introspección, de modo que su funcionalidad y propiedades puedan ser descubiertas y utilizadas en tiempo de ejecución.

- Estilos de Código Móvil
  - Arquitectura de máquinas virtuales: La arquitectura de máquinas virtuales se ha llamado también intérpretes basados en tablas o sistemas basados en reglas, este estilo incluye un seudo-programa a interpretar y una máquina de interpretación. Ambas variedades abarcan, sin duda, un extenso espectro

que va desde los llamados lenguajes de alto nivel hasta los paradigmas declarativos no secuenciales de programación.

- Estilos heterogéneos
  - Sistemas de control de procesos: Desde el punto de vista arquitectónico, mientras casi todos los demás estilos se pueden definir en función de componentes y conectores, los sistemas de control de procesos se caracterizan no sólo por los tipos de componentes, sino por las relaciones que mantienen entre ellos [27].

    En uno de los pocos tratamientos arquitectónicos para modelos cibernéticos. La ventaja señalada para este estilo radica en su elasticidad ante perturbaciones externas.
  - Arquitecturas basadas en atributos: La arquitectura basada en atributos o ABAS fue propuesta por Klein y Klazman. La intención de estos autores es asociar a la definición del estilo arquitectónico un framework de razonamiento (ya sea cuantitativo o cualitativo) basado en modelos de atributos específicos.
- Estilos Peer-to-Peer
  - Arquitecturas basadas en eventos: Las arquitecturas basadas en eventos se han llamado también de invocación implícita. Las arquitecturas basadas en eventos se vinculan históricamente con sistemas basados publicación-suscripción. Los conectores de estos sistemas incluyen procedimientos de llamada tradicionales y vínculos entre anuncios de eventos e invocación de procedimientos. La idea dominante en la invocación implícita es que, en lugar de invocar un procedimiento en forma directa un componente puede anunciar mediante difusión uno o más eventos [28].
  - Arquitecturas orientadas a servicios (SOA): Es lo suficientemente flexible, elegante y ágil garantizando las soluciones que las empresas han anhelado siempre. Es una arquitectura de software construye toda la topología de la aplicación como una topología de interfaces, implementaciones y llamados a interfaces; es una relación entre servicios y consumidores de servicios, ambos lo suficientemente amplios como para representar una función de negocio completa. [1]
  - Arquitecturas basadas en recursos: Define recursos identificables y métodos para acceder y manipular el estado de esos recursos. El caso de referencia es nada menos que la World Wide Web, donde los URIs identifican los recursos y HTTP es el protocolo de acceso. El argumento central es que HTTP mismo, con su conjunto mínimo de métodos y su semántica simplísima, es suficientemente general

[1] Consultar Anexo no. 1 para profundizar en las características del estilo arquitectónico (SOA).

para modelar cualquier dominio de aplicación. De esta manera, el modelado tradicional orientado a objetos deviene innecesario y es reemplazado por el modelado de entidades tales como familias jerárquicas de recursos abstractos con una interfaz común y una semántica definida por el propio HTTP.

1.4.2 *Health Level Seven* (HL7)

HL7 es una organización internacional, iniciada en los Estados Unidos en 1987, que pretende promover el desarrollo y evolución del estándar para el formato de datos e intercambio de información entre diferentes sistemas de información de salud.

Está abierta a todos los diferentes actores del ámbito de las tecnologías de la información y la salud (prestadores de servicios de salud, desarrolladores de software, consultores, usuarios finales, organizaciones gubernamentales, universidades y otras organizaciones) y desarrolla estándares por consenso en un entorno abierto [29].

En la actualidad, los sistemas informáticos de los servicios de salud de las comunidades, promocionan el uso del estándar *XML/HL7***.** Esta tecnología, garantiza la transferencia de información clínica entre comunidades autónomas y entre hospitales, desde los resultados de una prueba diagnóstica hasta el historial completo de un paciente. Ejemplo de esto: Software AG promueve el uso de XML/HL7, un estándar basado en XML específico para el sector sanitario, que hace posible la interoperabilidad entre los sistemas informáticos de distintos servicios de salud. De esta forma, cualquier médico o ciudadano tiene la posibilidad de acceder a su historia clínica única si necesita de atención especializada o requiere de una intervención quirúrgica fuera de su comunidad.

Bajo este protocolo de intercambio de información estándar se facilita la interconexión entre los distintos sistemas informáticos de las comunidades autónomas para garantizar así un flujo rápido y seguro de la información clínica del paciente [30].

1.4.3 Lenguaje de Estilo Extensible (XSL)

XSL es una tecnología XML de hojas de estilos que sirve para darles formato de presentación a documentos de ese tipo. Permite transformar documentos XML en otros, y la manipulación de su información.
En un documento de este tipo se describe un conjunto de reglas para aplicarse en documentos XML, encaminadas a la presentación dichos documentos.

1.4.4 Transformaciones XSL (XSLT)

Es un estándar de la organización W3C que presenta una forma de transformar documentos XML en otros e incluso a formatos que no son XML. Las hojas de XSLT realizan la transformación del documento utilizando una o varias reglas de plantilla: unidas al documento fuente a transformar, esas reglas de plantilla alimentan a un procesador de XSLT, que realiza las transformaciones deseadas colocando el resultado en un archivo de salida o, como en el caso de una página web, directamente en un dispositivo de presentación, como el monitor de un usuario. Actualmente, XSLT es muy usado en la edición web, generando páginas HTML o XHTML. La unión de XML y XSLT permite separar contenido y presentación, aumentando así la productividad. [31]

XSLT es la alternativa cuando uno quiere adaptar un contenido descrito con XML a diferentes clientes. Lo que consiguen las hojas de estilo es separar la información de su presentación, usando en cada caso las transformaciones que sean necesarias para que el contenido aparezca de la forma más adecuada en el cliente. Es más, se pueden usar diferentes hojas de estilo, o incluso la misma, para presentar la información de diferentes maneras dependiendo de los deseos o de las condiciones del usuario [32].

1.4.5 Lenguaje de Marcado Extensible (XML)

Es un lenguaje de meta marcaje que proporciona un formato para describir datos estructurados. Facilita declaraciones más precisas de contenido y resultados de búsquedas, con más significado entre muchas plataformas. Además, el XML habilitará una nueva generación de aplicaciones manipulación y

visualización de datos basadas en la WEB. El número de empresas que migrado sus páginas web a esta especificación se han incrementado de forma vertiginosa.

La idea que subyace bajo la creación del XML, es la de crear un lenguaje muy general, que pueda tener muchos usos. Mientras que HTML está diseñado para presentar información directamente a los usuarios, aunque es un lenguaje complejo de procesar para los programas informáticos. El HTML no indica lo que está representando, se preocupa principalmente de que el contenido tenga determinadas características visuales, pero no especifica lo que está mostrando, si es el título de un libro o el precio de un artículo. Contrario a esto, XML describe el contenido de lo que etiqueta.

Cada documento XML tiene una estructura tanto lógica como física:

• Físicamente, el documento está compuesto de unidades llamadas entidades. Una entidad puede referirse a otras entidades con el fin de causar su inclusión en el documento. Un documento comienza en una "raíz" o entidad documento.

• Lógicamente, el documento está compuesto de declaraciones, elementos, comentarios, referencias de carácter e instrucciones de proceso. Todas estas estructuras lógicas son indicadas mediante las correspondientes marcas.

Para que un objeto de texto sea considerado documento XML debe estar “bien formado” de acuerdo con la especificación XML. Un documento se considera “bien formado” si, tomado como un todo, el documento XML cumple la especificación de documento, es decir, tiene un prólogo, un elemento raíz (dentro del cual pueden ir otros elementos) y puede terminar con comentarios, instrucciones de procesamiento o espacios en blanco, cumple con los requerimiento de la especificación del lenguaje, todas las entidades a las que referencia estén bien formados, debe tener asociado un tipo de documento XSchema o DTD.

1.4.6 Esquema XML (XSchema)

XSchema es un lenguaje de definición de clases de documentos XML. Realmente un documento de este tipo es un documento XML con unas marcas determinadas. A los documentos XML derivados de un documento XSchema se les suele denominar “documentos instancia”.

XSchema dispone de una amplia gama de tipos básicos predefinidos para la definición de atributos y elementos. Estos tipos básicos están compuestos en su mayoría por tipos simples, aunque también podemos encontrar un tipo lista y un tipo unión. También podemos construir nuestros propios tipos de datos, extendiéndolos de los existentes o creando estructuras nuevas.
XSchema es un lenguaje muy potente para definir documentos XML y nos será de gran ayuda para validar los documentos XML que comuniquemos.

1.4.7 Protocolo de Acceso Simple a Objetos (SOAP)

SOAP es un protocolo creado por Microsoft, IBM y otros, actualmente está bajo el auspicio de la W3C que define cómo dos objetos en diferentes procesos pueden comunicarse por medio de intercambio de datos XML. SOAP es uno de los protocolos utilizados en los servicios web.

A diferencia de DCOM y CORBA, que son binarios, SOAP usa el código fuente en XML. Esto es una ventaja pues facilita su lectura por parte de los usuarios, pero también es un inconveniente que los mensajes resultantes son más largos. El intercambio de mensajes se realiza mediante componentes. El término object en el nombre significa que se adhiere al paradigma de la programación orientada a objetos.

SOAP es un marco extensible y descentralizado que permite trabajar sobre múltiples pilas de protocolos de redes informáticas. Los procedimientos de llamadas remotas pueden ser modelados en la forma de varios mensajes SOAP interactuando entre sí.

1.4.8 Lenguaje de Descripción de Servicios Web (WSDL)

Se utiliza para definir la interfaz web de cualquier función RPC expuesta en el extremo HTTP y para describir la funcionalidad de los lotes SQL para el extremo. El cliente puede solicitar una respuesta WSDL de una instancia de SQL Server y utilizarla para generar solicitudes de lotes RPC y SQL que enviará al servidor a través de los extremos HTTP configurados para asegurar la compatibilidad con los tipos WSDL. La respuesta WSDL es un documento XML generado dinámicamente a partir de las funciones RPC asociadas al extremo en el momento de generar la solicitud [33].

Un documento WSDL proporciona la información necesaria al cliente para interaccionar con el servicio Web. WSDL es extensible y se pude utilizar para describir, prácticamente, cualquier servicio de red, incluyendo SOAP sobre HTTP e incluso protocolos que no se basan en XML como DCOM sobre UDP.

Dado que los protocolos de comunicaciones y los formatos de mensajes están estandarizados en la comunidad WEB, cada día aumenta la posibilidad e importancia de describir las comunicaciones de forma estructurada. WSDL afronta esta necesidad definiendo una gramática XML que describe los servicios de red como colecciones de puntos finales de comunicación capaces de intercambiar mensajes. Las definiciones de servicio de WSDL proporcionan documentación para sistemas distribuidos y sirven como fórmula para automatizar los detalles que toman parte en la comunicación entre aplicaciones [34].

1.4.9 Descripción, Descubrimiento e Integración de Servicios WEB (UDDI)

La iniciativa UDDI surgió como respuesta a estas preguntas. Varias empresas, incluidas Microsoft, IBM, Sun, Oracle, Compaq, Hewlett Packard, Intel, SAP y unas trescientas más (para obtener un listado completo, consulte UDDI Community), unieron sus esfuerzos para desarrollar una especificación basada en estándares abiertos y tecnologías no propietarias que permitiera resolver los retos anteriores. El resultado, cuya versión beta se lanzó en diciembre de 2000 y estaba en producción en mayo de 2001, fue un registro empresarial global alojado por varios nodos de operadores en el que los usuarios podían realizar búsquedas y publicaciones sin coste alguno [35].

UDDI es un registro público diseñado para almacenar de forma estructurada información sobre empresas y los servicios que éstas ofrecen así como la integración entre estos.

1.4.10 .Net Framework

Microsoft .NET Framework versión 2.0 instala los archivos asociados de .NET Framework necesarios para ejecutar aplicaciones desarrolladas sobre dicho framework.
Mejora la escalabilidad y el rendimiento de aplicaciones gracias a características mejoradas como el almacenamiento en caché, el desarrollo de aplicaciones y la actualización con ClickOnce; además, es compatible con la gama más amplia de exploradores y dispositivos con servicios y controles ASP.NET 2.0.

Fue diseñado para cumplir con objetivos como: proporcionar un entorno coherente de programación orientada a objetos, en el que el código de los objetos se pueda almacenar y ejecutar de forma local, ejecutar de forma local pero distribuida en Internet o ejecutar de forma remota; proporcionar un entorno de ejecución de código que reduzca lo máximo posible la implementación de software y los conflictos de versiones, la ejecución segura del mismo, incluso del creado por terceras personas desconocidas o que no son de plena confianza, que elimine los problemas de rendimiento de los entornos en los que se utilizan secuencias de comandos o intérpretes de comandos; basar toda la comunicación en estándares del sector para asegurar que el código de .NET Framework se puede integrar con otros tipos de código. .NET tiene dos componentes principales: Common Language Runtime (CLR) y la biblioteca de clases de .NET framework (BCL - Basic Class Library).

1.4.11 El Common Language Runtime (CLR)

El CLR administra memoria, ejecución de subprocesos, ejecución de código, comprobación de la seguridad del código, compilación y demás servicios del sistema. Podríamos decir que esta es la máquina virtual de .NET. El código destinado al motor de tiempo de ejecución se denomina *código administrado*, a diferencia del resto de código, que se conoce como *código no administrado.*[2]

El CLR aporta características de vital importancia, algunas de estas son: seguridad avanzada: En el CLR los componentes administrados reciben niveles de confianza diferentes en función de una serie de factores entre los que se incluye su localización, origen (fabricante) y permisos o roles del usuario que lo este ejecutando; seguridad de tipos: el CLR impone la robustez del código mediante la implementación de un sistema estricto de comprobación de tipos y código denominada Sistema de Tipos Común (CTS). Gestión de memoria: el CLR mantiene el control de la ubicación de los objetos, administra las referencias a éstos y los libera cuando ya no se utilizan; integración de lenguajes: los programadores pueden crear aplicaciones en el lenguaje que prefieran y seguir sacando todo el provecho del motor de tiempo de ejecución, la biblioteca de clases y los componentes escritos en otros lenguajes por otros colegas; interoperabilidad con código antiguo, soporte multihilo, ejecución multiplataforma(Windows 95/98/ME/NT 4.0/2000/XP/2003/CE, proyecto Mono).

[2] Consultar el Anexo no. 6: Estructura de sistemas con .NET.

1.4.12 La Biblioteca de Clases de .NET

La biblioteca de clases de .NET Framework es una colección de tipos reutilizables que se integran estrechamente con el CLR. La biblioteca de clases está orientada a objetos, lo que proporciona tipos de los que su propio código administrado puede derivar funciones. Esto ocasiona que los tipos de .NET Framework sean sencillos de utilizar y reduce el tiempo asociado con el aprendizaje de las nuevas características de este. Además, los componentes de terceros se pueden integrar sin dificultades con las clases.

Esta librería está escrita en Lenguaje Intermedio de Microsoft (MSIL-Microsoft Intermediate Language) que es un conjunto de instrucciones independiente de la Unidad central de procesamiento que se pueden convertir de forma eficaz en código nativo. Por tanto esta librería puede ser utilizada desde cualquier lenguaje cuyo compilador genere MSIL.

1.4.13 Mono

Por definición, Mono es una plataforma de desarrollo de código abierto basada en el .NET Framework. Es una plataforma para ejecutar y desarrollar aplicaciones modernas basadas en los estándares ECMA/ISO. Mono puede ejecutar aplicaciones hechas para los frameworks .NET y Java.
Permite a los desarrolladores construir aplicaciones GNU-Linux y multiplataforma de alta productividad. La implementación cumple los estándares ECMA para C# y la infraestructura de lenguaje común (Common Language Infraestructure, CLI).

Acoge varias licencias libres (X11, LGPL y GPL) y tiene una comunidad de desarrolladores activa, está patrocinado por Novell y es la base para muchas aplicaciones.

Mono incluye compiladores, un intérprete compatible con el CLR de ECMA y un conjunto de librerías. Las librerías cubren la compatibilidad con Microsoft .NET (incluyendo ADO.NET, System.Windows.Forms y ASP.NET). Mono posee librerías adicionales y otras de terceras partes. Para el desarrollo de aplicaciones

gráficas se incluye la librería GTK# que es un "binding" sobre GTK+ y GNOME para permitir el desarrollo de aplicaciones nativas para Gnome utilizando Mono y cualquiera de los lenguajes soportados. El diseño de interfaces gráficas se puede realizar con Glade. Además está Monodevelop que es un IDE (entorno de desarrollo integrado) que integra la gestión de proyectos, la construcción de aplicaciones, depurado y toda la documentación (APIs, especificaciones, etc.)

1.4.14 Analizador de migración a plataforma Mono (Mono Migration Analyser MOMA)

MOMA ayuda al programador a identificar requisitos que debe cumplir al migrar las aplicaciones hechas en .Net a Mono, analizando los ensamblados sobre los cuales se basa el funcionamiento de la aplicación.

Existen cuatro puntos sobre los cuales MOMA fija su atención.
Missing Methods: Se reportan cuando son detectados métodos pertenecientes a los ensamblados que no se encuentran implementados en MONO.
MonoTodo: Los métodos marcados con esta clasificación pueden o no causar problemas en la aplicación ya que puede ser que no estén implementados completamente, necesiten refinamiento, o que simplemente solo cuentan con la declaración, por lo que no realizarán la función que se espera de ellos.
NotImplementedException: Sucede cuando existe alguna llamada a un método que puede provocar el lanzamiento de una excepción de este tipo, por lo que nunca se podrá llegar a determinar la verdadera razón del error.
P/ Invokes: Los métodos marcados con esta categoría son aquellos que realizan alguna llamada a funciones específicas de la plataforma (SO), por lo que son un punto interesante a considerar.

Aunque MOMA garantice resultados confiables, debemos recordar siempre que la verdadera prueba se debe hacer ejecutando la aplicación sobre MONO.

1.4.15 XML y Javascript Asíncrono (AJAX)

AJAX parece ser la palabra de moda en el mundo del desarrollo de aplicaciones web. No es una tecnología, sino la unión de varias, que pueden lograr cosas realmente impresionantes, incorporando: presentación basada en estándares usando XHTML y CSS; exhibición e interacción dinámicas usando el

Document Object Model; intercambio y manipulación de datos usando XML and XSLT; recuperación de datos asincrónica usando XMLHttpRequest; y Javascript para unir todo el contenido.

Es el acrónimo para Asynchronous Javascript + XML y se resume en: cargar y renderizar una página, luego mantenerse en esta, mientras scripts y rutinas van al servidor buscando, en *background*, los datos que son usados para actualizar la página solo re-renderizando la página y mostrando u ocultando porciones de la misma [36].

1.4.16 Lenguajes del lado del servidor

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

1.4.16.1 C#

C# es un lenguaje de programación orientado a objetos desarrollado y estandarizado por Microsoft como parte de su plataforma .NET, que después fue aprobado como un estándar por la ECMA e ISO.
Su sintaxis básica deriva de C/C++ y utiliza el modelo de objetos de la plataforma .NET el cual es similar al de Java aunque incluye mejoras derivadas de otros lenguajes (más notablemente de Delphi y Java). C# fue diseñado para combinar el control a bajo nivel de lenguajes como C y la velocidad de programación de lenguajes como Visual Basic.

C#, como parte de la plataforma .NET, está normalizado por ECMA desde diciembre de 2001 (ECMA-334 "Especificación del Lenguaje C#"). El 7 de noviembre de 2005 acabó la beta y salió la versión 2.0 del lenguaje que incluye mejoras tales como tipos genéricos, métodos anónimos, iteradores, tipos parciales y

tipos anulables. Ya existe la versión 3.0 de C# en fase de beta destacando los tipos implícitos y el LINQ (Language Integrated Query).

Aunque es posible escribir código para la plataforma .NET en muchos otros lenguajes, C# es el único que ha sido diseñado específicamente para ser utilizado sobre ella. Por lo que programar usando C# es mucho más sencillo e intuitivo que hacerlo con cualquiera de los otros lenguajes, disponibles en .NET, carece de elementos heredados innecesarios.

Las principales características que hacen del lenguaje C# una elección inteligente:

- Sencillez e independencia de tipos: El tamaño de los tipos de datos básicos es fijo e independiente del compilador, sistema operativo o máquina para quienes se compile (no como en C++), lo que facilita la portabilidad de código,
- Modernidad: C# incorpora en el propio lenguaje elementos que se han mostrado útiles para el desarrollo de aplicaciones y que en otros lenguajes como Java o C++,
- Orientación a objetos: Como todo lenguaje de programación, C# es un lenguaje orientado a objetos, no admite ni funciones ni variables globales sino que todo el código y datos han de definirse dentro de definiciones de tipos de datos, lo que reduce problemas por conflictos de nombres y facilita la legibilidad del código,
- Orientación a componentes: La propia sintaxis de C# incluye elementos propios del diseño de componentes que otros lenguajes tienen que simular mediante construcciones más o menos complejas. Es decir, la sintaxis de C# permite definir cómodamente propiedades (similares a campos de acceso controlado), eventos (asociación controlada de funciones de respuesta a notificaciones) o atributos (información sobre un tipo o sus miembros),
- Seguridad de tipos,
- No se pueden usar variables no inicializadas,
- Versionable: C# incluye una política de versionado que permite crear nuevas versiones de tipos sin temor a que la introducción de nuevos miembros provoquen errores difíciles de detectar en tipos hijo previamente desarrollados y ya extendidos con miembros de igual nombre a los recién introducidos;
- Eficiente,

- Compatible: Para facilitar la migración de programadores, C# no sólo mantiene una sintaxis muy similar a C, C++ o Java que permite incluir directamente código escrito en C# fragmento de código escritos es estos lenguajes, sino que el CLR también ofrece, a través de los llamados Plataform Invocation Services (Pinvoke) [37].

Aunque C# forma parte de la plataforma .NET, esta es una interfaz de programación de aplicaciones; mientras que C# es un lenguaje de programación independiente diseñado para generar programas sobre dicha plataforma.

Para el lenguaje C#, existen los siguientes compiladores e IDEs:

- Microsoft .NET *framework* SDK incluye un compilador de C#, pero no un IDE.
- Microsoft Visual C#, IDE por excelencia de este lenguaje, versión 2002, 2003 y 2005.
- SharpDevelop, es un IDE libre para C# bajo licencia LGPL, muy similar a Microsoft Visual C#.
- Mono, es una implementación GPL de todo el entorno .NET desarrollado por Novell. Como parte de esta implementación se incluye un compilador de C#.
- Delphi 2006, de Borland Software Corporation.
- dotGNU Portable.NET, de la Free Software Foundation.

1.4.16.2 ASP.NET

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

- Mejor rendimiento, eficacia y flexibilidad**.** ASP.NET es un código de Common Language Runtime compilado que se ejecuta en el servidor. A diferencia de sus predecesores, ASP.NET puede aprovechar las ventajas del enlace anticipado, la compilación just-in-time, la optimización nativa y los servicios de caché desde el primer momento. Esto supone un incremento espectacular del rendimiento antes de siquiera escribir una línea de código. La biblioteca de clases de .NET Framework, la mensajería y las soluciones de acceso a datos se encuentran accesibles desde el web de manera

uniforme. Es también independiente del lenguaje, por lo que puede elegir el lenguaje que mejor se adapte a la aplicación o dividir la aplicación en varios lenguajes.

- Compatibilidad con herramientas de primer nivel. El marco de trabajo de ASP.NET se complementa con un diseñador y una caja de herramientas muy completos en el entorno integrado de programación (Integrated Development Environment, IDE) de Visual Studio. La edición WYSIWYG, los controles de servidor de arrastrar y colocar y la implementación automática son sólo algunas de las características que proporciona esta eficaz herramienta.
- Facilidad de uso**.** ASP.NET emplea un sistema de configuración jerárquico, basado en texto, que simplifica la aplicación de la configuración al entorno de servidor y las aplicaciones Web. Debido a que la información de configuración se almacena como texto sin formato, se puede aplicar la nueva configuración sin la ayuda de herramientas de administración local. No se requiere el reinicio del servidor, ni siquiera para implementar o reemplazar el código compilado en ejecución.
- Escalabilidad y disponibilidad**.** ASP.NET se ha diseñado teniendo en cuenta la escalabilidad, con características diseñadas específicamente a medida, con el fin de mejorar el rendimiento en entornos agrupados y de múltiples procesadores. Además, el motor de tiempo de ejecución de ASP.NET controla y administra los procesos de cerca, por lo que si uno no se comporta adecuadamente (filtraciones, bloqueos), se puede crear un proceso nuevo en su lugar, lo que ayuda a mantener la aplicación disponible constantemente para controlar solicitudes.
- Seguridad. Con la autenticación de Windows integrada y la configuración por aplicación, se puede tener la completa seguridad de que las aplicaciones están a salvo [38].

1.4.16.3 Perl

Es un lenguaje de propósito general originalmente desarrollado para la manipulación de texto y que ahora es utilizado para un amplio rango de tareas incluyendo administración de sistemas, desarrollo web, programación en red y desarrollo de interfaces graficas de usuario.

Se utiliza regularmente para construir aplicaciones CGI para la web. Es software libre, un lenguaje de programación interpretado y extensible a partir de otros lenguajes.

1.4.16.4 PHP

Es un lenguaje de programación usado generalmente para la creación de contenido para sitios web.
PHP es un acrónimo recursivo para "PHP Hypertext Pre-processor" (inicialmente PHP Tools, o, Personal Home Page Tools), y se trata de un lenguaje interpretado, usado para la creación de aplicaciones para servidores, o creación de contenido dinámico para sitios web. Últimamente también para la creación de otro tipo de programas incluyendo aplicaciones con interfaz gráfica usando la biblioteca GTK+4. En PHP los scripts del lado del servidor se insertan dentro del código HTML. Es gratuito, multiplataforma y está ampliamente difundido en el mundo entre la comunidad de programadores.

1.4.16.5 Java Server Pages (JSP)

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

1.4.16.6 Common Gateway Interface (CGI)

Permite a un cliente (navegador web) solicitar datos de un programa ejecutado en un servidor web. CGI especifica un estándar para transferir datos entre el cliente y el programa. Es un mecanismo de comunicación entre el servidor web y una aplicación externa. Las aplicaciones CGI fueron una de las primeras maneras prácticas de crear contenido dinámico para las páginas web. En una aplicación CGI, el servidor web pasa las solicitudes del cliente a un programa externo. La salida de dicho programa es enviada al cliente en lugar del archivo estático tradicional. Se convirtió en un estándar y tomó un gran auge debido a la posible implementación de nuevas funcionalidades en las páginas Web.

1.4.17 Lenguajes del lado del cliente.

Se ejecutan en el cliente o navegador, son los encargados de darle dinamismo a la página sin necesidad de realizar viajes al servidor. Estos lenguajes son interpretados, pueden acceder a la información HTML que se muestra en un navegador pudiendo modificarla y actualizarla según las necesidades de los programadores. Se encuentran entre los más populares JavaScript y Visual Basic Script.

1.4.17.1 Java Script

Javascript es un lenguaje interpretado, es decir, que no requiere compilación, utilizado principalmente en páginas web, con una sintaxis semejante a la del lenguaje Java y el lenguaje C. Al contrario que Java, Javascript no es un lenguaje orientado a objetos propiamente dicho, ya que no dispone de herencia, es más bien un lenguaje basado en prototipos, ya que las nuevas clases se generan clonando las clases base (prototipos) y extendiendo su funcionalidad. Todos los navegadores interpretan el código JavaScript integrado dentro de las páginas web. Estadísticas brindadas por la w3shools ratifican que en el presente año 2007, el 94% de los sitios muestreados utilizan este lenguaje del lado del cliente.

1.4.17.2 Visual Basic Script

Es un lenguaje que, a diferencia de JavaScript, es solamente compatible con Internet Explorer, aunque posee toda la funcionalidad que brinda JavaScript.

1.4.18 Servidores WEB

1.4.18.1 Apache

En lo que respecta a las tecnologías por parte del servidor sobresale Apache. El servidor HTTP Apache es un producto libre, de código abierto para plataformas Unix (BSD, GNU/Linux, etcétera), Windows y otras. Implementa el protocolo HTTP/1.1. Presenta entre otras características mensajes de error altamente configurables, bases de datos de autenticación y negociado de contenido.

Es un servidor de red para el protocolo HTTP, elegido para poder funcionar como un proceso independiente, sin que solicite el apoyo de otras aplicaciones o directamente del usuario. Para poder hacer esto, Apache, una vez que se haya iniciado, crea unos subprocesos (que normalmente vienen llamados "*children processes*") para poder gestionar las solicitudes: estos procesos, sin embargo, no podrán nunca interferir con el proceso mayor.

Tiene amplia aceptación en la red: en el 2005 fue el servidor HTTP más usado, en el 70% de los sitios web en el mundo y creciendo aún su cuota de mercado (estadísticas históricas y de uso diario proporcionadas por Netcraft).

1.4.18.2 Internet Information Services (IIS)

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

1.4.19 Navegadores WEB

En internet, existen un número considerable de navegadores, pero vale aclarar que no todos son capaces de asimilar el mismo tipo de contenido. Nuestra decisión ha sido la de crear aplicaciones compatibles, inicialmente, con los navegadores IE6 y Firefox, dándole compatibilidad a nuestro producto en plataformas Windows y Linux, a partir de un estudio realizado, del que se obtiene que estos dos navegadores representan los más usados por los clientes, pues los porcentajes de uso en el 2007 oscilan entre 39.8 y 42.3 % para el IE6 y entre 31.0 y 31.2 % para el Firefox (programa libre y de código abierto), el resto de los navegadores no excede el 20 % de utilización. Durante el 2006, estas estadísticas se comportaron de manera similar. [3]

[3] Consultar el Anexo No.2.

1.4.20 Sistemas Gestores de Bases de Datos (SGBD ó DBMS)

1.4.20.1 PostgreSQL

Hoy día, existen muchas empresas y sitios web que necesitan mantener de forma eficiente un gran volumen de datos. Muchos de ellos optan por soluciones comerciales, aunque muchas otras confían en el software libre optando por una solución como PostgreSQL o MySQL.

Dentro de los gestores de bases de datos sobresale PostgreSQL, un motor de base de datos que es servidor de base de datos relacional libre, liberado bajo la licencia BSD.
PostgreSQL no está controlado por una sola compañía, sino que cuenta con comunidad global de desarrolladores y compañías para su evolución.
Ofrece una potencia adicional sustancial al incorporar los siguientes cuatro conceptos adicionales básicos en una vía en la que los usuarios pueden extender fácilmente el sistema: clases, herencia, tipos, funciones.
Otras características aportan potencia y flexibilidad adicional:

- Restricciones (constraints)
- Disparadores (triggers)
- Reglas (rules)
- Integridad transaccional

Las principales mejoras en PostgreSQL incluyen:

- Los bloqueos de tabla han sido sustituidos por el control de concurrencia multi-versión, el cual permite a los accesos de sólo lectura continuar leyendo datos consistentes durante la actualización de registros, y permite copias de seguridad en caliente desde pg_dump mientras la base de datos permanece disponible para consultas. Esta técnica elimina la necesidad hacer bloqueos de lectura y asegura los principios ACID (atomicidad, consistencia, aislamiento y durabilidad) de la base de datos de una manera eficiente.

- Se han implementado importantes características del motor de datos, incluyendo subconsultas, valores por defecto, restricciones a valores en los campos (constraints) y disparadores (triggers).
- Se han añadido funcionalidades en línea con el estándar SQL92, incluyendo claves primarias, identificadores entrecomillados, forzado de tipos cadena literal, conversión de tipos y entrada de enteros binarios y hexadecimales.
- Los tipos internos han sido mejorados, incluyendo nuevos tipos de fecha/hora de rango amplio y soporte para tipos geométricos adicionales.
- Establecer condiciones o CHECKs para validar las entradas de datos
- Permitir transacciones, es decir, múltiples operaciones de tabla o registros de manera segura.
- Bloqueos de registros, útil en entornos hoy por hoy multiusuario.
- Administración de Grupos de usuarios, y soporte nativo SSL.
- Múltiples lenguajes para procedimientos almacenados (incluyendo el nativo PL/PgSQL, PL/PHP, PL/Perl y PL/Python)
- La velocidad del código del motor de datos ha sido incrementada aproximadamente en un 20-40%, y su tiempo de arranque ha bajado el 80% desde que la versión 6.0 fue lanzada.
- Incluye Slony-I: Slony-I es un sistema de replicación “maestro a múltiples esclavos”, que soporta actualizaciones en cascada y promociones de esclavos. La gran razón para incluirla como herramienta de replicación es que de esta forma contamos con un sistema que posee las capacidades necesarias para replicar grandes bases de datos a un número razonable de sistemas esclavos. Es un sistema previsto para centros de datos y sitios de respaldo, donde la situación normal es que todos los nodos se encuentres disponibles, y puedan ser asegurados.

1.4.20.2 Microsoft SQL Server

Producido por Microsoft. Su principal lenguaje de consulta es Transact SQL, una implementación del lenguaje de consulta estructurado ANSI/ISO usado tanto por Microsoft como por Sybase. SQL Server es comúnmente usado por empresas que necesitan de pequeñas a medianas bases de datos, aunque en los últimos años ha sido ampliamente adoptado por empresas que poseen grandes bases de datos

empresariales. Microsoft SQL Server constituyó la entrada de Microsoft hacia el mercado de bases de datos de nivel empresarial, compitiendo con Oracle, IBM, y luego, el propio Sybase.
Su última versión SQL Server 2005, es un potente gestor de bases de datos, que incorpora numerosas características completamente novedosas, lo que lo hace un producto extremadamente costoso.

1.4.20.3 Oracle

Larry Ellison fundó Software Development Laboratories (SDL) en 1977. En 1979 SDL cambió el nombre de su compañía a Relational Software, Inc. (RSI) e introdujo su producto Oracle V2 como un sistema de administración de bases de datos incipiente disponible comercialmente. Esa versión no soportaba transacciones pero implementaba la funcionalidad básica de SQL, consultas y joins (uniones).
Desde 1985 el Sistema de Administración de Bases de Datos Oracle DBMS comenzó a soportar el modelo cliente servidor, y consultas distribuidas. En 1992 Oracle versión 7h (h significaba datawareHouse) apareció con soporte para la integridad referencial, procedimientos almacenados y triggers. En 2001 Oracle 9i apareció en el mercado con 400 nuevas características, incluyendo la posibilidad de leer y escribir documentos XML. 9i también brindaba una opción para Oracle RAC (Real Aplication Clusters), una base de datos de clústeres de computadoras, como reemplazo a Oracle Parallel Server (OPS) que había lanzado antes.

1.4.20.4 MySQL

Es un sistema de administración de base de datos, multi - hilo, multiusuario, con más de seis millones de instalaciones por todo el planeta. MySQL es propietaria y patrocinada por una sola firma, la compañía sueca MySQLAB que mantiene el derecho de la mayor parte del código base. La compañía desarrolla y mantiene el sistema, vende soporte y contratos de servicio, así como copias de licenciadas a propietarios de MySQL y emplea a personas de todo el mundo quienes colaboran vía internet.
Existen muchas APIs que permiten que aplicaciones escritas en varios lenguajes de programación accedan a bases de datos.

MySQL es muy popular en aplicaciones web y actúa como un componente de bases de datos para las plataformas LAMP, MAMP y WAMP (Linux/MAC/Windows-Apache-MySQL-PHP/Perl/Phyton) y para

herramientas de búsqueda e identificación de errores como Bugzilla. Para administrar bases de datos en MySQL pueden usarse herramientas de línea de comandos como mysql y mysqladmin.
MySQL trabaja en numerosas plataformas como AIX, HP-UX, GNU/Linux, Mac OS X, Novell NetWare, OpenBSD, OS/2, Solaris, SunOS, y todas las versiones de Windows. Su mayor desempeño se logra cuando se combina con el lenguaje de programación PHP.

1.4.21 Rational Rose

Este paquete de Rational permite documentar un producto software brindando una serie de facilidades para la elaboración de los artefactos que propone el Proceso Unificado Racional. Está compuesto a su vez por un grupo de herramientas que se especializan en un aspecto en específico:

- Rational Rose Enterprise Edition: Esta herramienta permite modelar visualmente un grupo de artefactos que se generan como parte de la metodología. Dividido en cuatro vistas fundamentales: vista de casos de uso, vista lógica, vista de componentes y vista de despliegue; propone un grupo de artefactos predefinidos que facilitan el desarrollo de software con calidad.
- Rational Requisite Pro: Esta herramienta gestiona los requerimientos funcionales y no funcionales del sistema a elaborar.
- Rational Clear Case**:** Rational ClearCase proporciona una gestión del ciclo de vida y control de los activos de desarrollo de software.
- Rational ClearQuest: proporciona un seguimiento flexible de defectos y cambios en toda la empresa. soporte completo para consultas con generación de múltiples informes y gráficos.
- Rational Soda: Automatiza la documentación del proyecto de software a lo largo de todo el ciclo de vida. Genera documentos mediante la extracción de datos solicitados directamente de los repositorios de datos de herramientas.

Existen dentro del paquete otras herramientas con otras funcionalidades que gestionan y satisfacen todo el proceso de desarrollo de software.

1.4.22 Visual Studio .NET

Visual Studio es un conjunto completo de herramientas de desarrollo para la generación de aplicaciones web ASP.NET, servicios web XML, aplicaciones de escritorio y aplicaciones móviles. Visual Basic, Visual

C++, Visual C# y Visual J# utilizan el mismo entorno de desarrollo integrado (IDE), que les permite compartir herramientas y facilita la creación de soluciones en varios lenguajes. Asimismo, dichos lenguajes aprovechan las funciones de .NET Framework, que ofrece acceso a tecnologías clave para simplificar el desarrollo de aplicaciones web ASP y servicios web XML [39].
Incorpora nuevas y mejoradas funciones de productividad: Configuración del IDE, importar y exportar configuraciones, listas de tareas, lista de errores, teclas de método abreviado Brief y Emacs.

Las siguientes funciones están disponibles en la versión propuesta:

- f*ragmentos de código***,** ofrece fragmentos personalizados listos para insertar en los proyectos;
- e*tiquetas inteligentes* que realizan tareas comunes disponibles que se aplican al contexto de su trabajo;
- *refactorización* incluye cambiar el nombre, extraer el método, extraer la interfaz, cambiar la firma y encapsular el campo; puede mantener un c*ontrol de cambios* estricto pudiendo observar dónde ha modificado un archivo en la sesión del IDE actual;
- la *ventana marcador* permite administrar y controlar los marcadores;
- *autorrecuperación***:** esta función guarda automáticamente los archivos que contengan cambios cada cinco minutos. Si el IDE se cierra inesperadamente, los archivos con cambios podrán recuperarse;
- *ventana esquema del documento***:** Esta ventana ahora admite vistas de esquema para los formularios Windows Forms, además de páginas web y páginas HTML de ASP.NET.

Visual Studio presenta un nuevo diseñador de páginas web que incluye muchas mejoras para la creación y edición de páginas web de ASP.NET y páginas HTML. Proporciona una forma más fácil y rápida de crear páginas de formularios Web Forms que en Visual Studio .NET 2003.
La vista del diseñador HTML incluye muchas mejoras que admiten las nuevas funciones de ASP.NET o facilitan el diseño WYSIWYG de páginas web. La edición basada en tareas mediante etiquetas inteligentes le guía durante la ejecución de los procedimientos más comunes con controles, como el enlace de datos y la asignación de formato. Puede editar visualmente las nuevas páginas principales de ASP.NET. La edición de plantillas se ha mejorado para facilitar el trabajo con controles de datos, así como con nuevos

controles como el control *login*. Permite editar tablas HTML para el diseño o mostrar la información en columnas de una forma más fácil e intuitiva.
Un nuevo editor XML está disponible en esta versión de Visual Studio. Este editor aprovecha la eficacia de las clases System.Xml y System.Xml.Xsl de .NET Framework y se ajusta a los estándares de XML.

1.4.23 SharpDevelop

SharpDevelop es un entorno integrado de desarrollo libre para los lenguajes de programación C#, Visual Basic .NET y Boo (programación).
Es usado típicamente por aquellos programadores de los citados lenguajes, que no desean o no pueden usar el entorno de desarrollo de Microsoft, el Microsoft Visual Studio. Para el completado automático de código, la aplicación incorpora sus propios parsers. La versión 2.1 ya es capaz de editarlos directamente.
Características principales:

- Incorpora un diseñador de Windows Forms.
- Completado de código. Soporta el uso de la combinación de teclas Ctrl + Espacio.
- Depurador incorporado.
- Herramientas para "Ir a Definición", "Encontrar referencias" y "renombrado".
- Títulos para títulos y para depuración.
- Conversor bidireccional entre C# y Visual Basic .NET, y unidireccional hacia Boo.
- Escrito enteramente en C#.
- Compilación de código directamente dentro del entorno de desarrollo integrado.
- Complementos para ILAsm y C++.
- Integración con herramientas de pruebas unitarias NUnit y MbUnit.
- Analizador para ensamblado FxCop.
- Previsualización de documentación XML.
- Gran integración con plantillas a la hora de añadir o crear ficheros, proyectos o compiladores.
- Escritura de código C#, ASP.NET, ADO.NET, XML y HTML.
- Coloreado de sintaxis para los lenguajes C#, HTML, ASP, ASP.NET, VBScript, Visual Basic .NET, y XML.

- Llaves inteligentes en la escritura de código.
- Gestión de marcadores (favoritos).
- Soporte para plantillas de código.
- Extensible mediante herramientas externas, o complementos.

1.4.24 Macromedia Dreamweaver

Dreamweaver 8 es la herramienta de desarrollo web líder del mercado y permite a sus usuarios diseñar, desarrollar y mantener de forma eficaz sitios y aplicaciones web basadas en normas.
Con Dreamweaver 8, los desarrolladores web lo abarcan todo, desde la creación y mantenimiento de sitios web básicos hasta aplicaciones avanzadas compatibles con las mejores prácticas y las tecnologías más recientes. Utilice un editor de diseño y código de primera calidad en una sola herramienta.

Dreamweaver ayuda y guía a los usuarios conforme éstos van aumentando sus conocimientos y a medida que las tecnologías web van evolucionando, facilitando una adopción fácil y rápida de las nuevas tecnologías y metodologías. Con Dreamweaver 8 y Flash video podrá agregar rápidamente contenidos de vídeo en la WEB. Arrastre y coloque vídeos de Flash en Dreamweaver 8 para agregar de forma rápida vídeos a sitios y aplicaciones web.

El panel unificado de CSS ofrece una representación sencilla y directa de la cascada de estilos aplicados al contenido y ofrece acceso rápido para realizar cambios sin necesidad de realizar búsquedas en el código probando por ensayo y error.

Simplemente señale una página web en un archivo XML o una URL de una entrada XML y Dreamweaver, lo introspeccionará para luego poder arrastrar y colocar los campos apropiados en la página.
Con más de tres millones de usuarios, la comunidad Dreamweaver ofrece numerosas ventajas, entre las que cabe mencionar el Centro para Desarrolladores de Macromedia, los programas de certificación para desarrolladores, los cursos de formación y seminarios, los foros de usuarios y los sitios independientes que ofrecen soporte a la comunidad. Hay disponibles más de 800 extensiones que pueden descargarse de forma gratuita a través de Macromedia Dreamweaver Exchange.

1.4.25 Herramientas HL7

1.4.25.1 Hermes: Framework HL7

Es una implementación confiable del estándar HL7 en su versión 2.3.1, desarrollada por un proyecto del área temática de la facultad 7 perteneciente a la Universidad de las Ciencias Informáticas, versión de HL7 más difundida a nivel mundial. Dicho framework en la próxima versión contendrá un conjunto de herramientas que permitirán la administración, integridad y flujo de la información médica entre los distintos Sistemas de Información para la Salud (SIS). Este garantiza un fuerte tipado, con notaciones más familiares al usuario y equivalentes a los tipos encapsulados en su núcleo, el usuario no necesita conocer las especificaciones en detalle de HL7 para utilizar el mismo en sus aplicaciones.
Al contar con sus herramientas se realizará del envío y recepción de datos clínicos, información de pacientes y algunos reportes de laboratorio, entre los distintos Sistemas de Información para la Salud que lo utilicen, empleando para ello un solo formato, el especificado por HL7. Reduce los recursos invertidos en la negociación de las interfaces entre aplicaciones para la salud, Con su utilización se lograría incrementar el prestigio del sistema de salud cubano, así como la confianza internacional en el mismo.

Permite el ahorro de una suma importante de dinero al país, al evitar la compra de implementaciones del estándar a empresas extranjeras.

Garantiza en estos momentos completa compatibilidad y posibilidades de integración con las aplicaciones médicas que implementen las reglas de codificación XML para la versión 2.3.1 (Encode Rule XML 2.3.1).
Presenta un estilo arquitectónico en dos capas, la capa inferior, se encuentra compuesta por el conjunto de clases y métodos encargados de construir los mensajes HL7, la capa superior consta de un grupo de componentes variable, que se utilizan para facilitar el trabajo con dicho framework.

1.4.25.2 Chameleon

Los componentes de mensajería *Active-X* para cibersalud incorporan en objetos los mensajes de la versión 2.3 de HL7 y facilitan la conexión de varios sistemas. Estas aplicaciones sólo tienen que solicitar el objeto idóneo (por ejemplo una orden de admisión o de medicación), agregar los elementos de datos

apropiados y enviar de forma transparente el objeto de que se trate a las aplicaciones configuradas para recibirlo. Una arquitectura flexible posibilita la migración entre los motores de interfaz y los sistemas tradicionales a aplicaciones basadas en componentes y conectadas en red.

Hoy día, pueden adquirirse productos HL7. *HL7 Chameleon* es uno de los muy conocidos productos. El interés principal de *Chameleon* es que separa la interfaz de mensaje de las diferentes implementaciones de HL7, lo que permite que una aplicación escrita con una interfaz soporte un gran número de diferentes versiones de HL7. *Chameleon* permite afrontar los problemas que puede plantear el aislamiento del código de aplicación respecto a la estructura de datos HL7 sin codificar.

Chameleon utiliza el archivo de definición de mensaje (VMD) como interfaz que actúa como puente entre el sistema y el mundo HL7.[4]

Soporta las versiones HL7 2.1 a 2.6 y la versión 3 cuando se encuentre disponible.
Es importante aclarar que dicha interfaz se crea dentro del VMD usando tablas, las cuales se usan como objetos[5] con los cuales se interactúa. Los datos son mapeados desde los mensajes entrantes y viceversa.

Dicha herramienta se encuentra disponible para las plataformas Windows 95/98/2000/XP, Windows Server 2003, Windows NT 3.51 y superior, Mac OS X, Solaris, GNU-Linux, SCO Unix, AS400, Tandem o cualquier sistema operativo que soporte un compilador C++ que cumpla con la norma ANSI 92. Está equipada con un grupo de aplicaciones de gran utilidad que de ser usadas equilibradamente reducen en gran medida el esfuerzo empleado en la creación y mantenimiento de las interfaces del sistema. Las aplicaciones mencionadas con anterioridad son HL7 Listener, HL7 Simulator, Message Browser y herramientas accesibles desde la línea de comandos (msgtransform, msgxml y msgdiff).

Presenta soporte para los lenguajes más utilizados en la actualidad: Java, C++, C#, Visual Basic, Delphi, .NET más soporte ActiveX para lenguajes asociados a la plataforma Windows como Power Builder, Foxpro y Microsoft Access.

[4] Consultar el anexo no. 7
[5] Consultar el anexo no. 8

Chameleon ha estado en el mercado por más de ocho años y se encuentra desplegado en mas de 5000 sitios, además cuenta con una lista de clientes de prestigio internacional que avalan su reputación: Patient Keeper, IBM , AccuImage, Veteran Affairs, University Health Network (UHN), Blue Cross Blue Shield Association, All Meds, + NUTH, LockheedMartin, Aramark, General Electric Medical Systems y SECTRA.

1.4.25.3 HL7 application programming interface (HAPI)

La herramienta consiste en una API open source orientada a objetos en Java. Incluye un parser HL7 para la versión 2.x. El proyecto no se encuentra afiliado a la organización HL7, solo se ha dedicado a producir software de acuerdo a las especificaciones, comenzó siendo una iniciativa de la UHN. Los mensajes HL7 son parseados utilizando objetos genéricos. Existen mensajes del estándar HL7 que no se encuentran implementados dentro de la herramienta lo que conlleva a un retraso en el desarrollo de aplicaciones que la utilicen ya que la validez de los mensajes tendría que chequearse manualmente. El código ha sido mayormente generado a mano por lo que se pueden incluir errores en alguna de sus implementaciones.

1.4.25.4 Perl HL7 Toolkit

Esta herramienta provee al cliente de una API sencilla para Perl capaz de enviar y recibir mensajes HL7 a través de la implementación de un servidor y cola HL7. Todos los paquetes son usados en entornos de producción por demanda por lo que es necesario que se chequee si el toolkit funciona para la aplicación en la que se necesite, no representa una implementación de propósito general.

La API no provee una implementación exhaustiva del estándar HL7, esta se enfoca en brindar una forma sencilla de crear mensajes que puedan ser enviados a cualquier servidor HL7 y recibir mensajes HL7 de un servidor compatible, parsearlos en segmentos y procesarlos.

1.4.25.5 Mirth

Mith es un motor con interfaz HL7 de plataformas cruzadas de código abierto que permite el envío biridecional de mensajes HL7 entre sistemas y aplicaciones sobre múltiples capas de transporte. Permite el filtrado de mensajes, el transformado, y el enrutamiento de los mismos en base a reglas definidas por el usuario. Crear interfaces HL7 para los sistemas es fácil utilizando la interfaz web y el asistente para crear canales que asocian las aplicaciones con los componentes del motor.

Para integrar los servicios con los sistemas HL7 se debe implementar una capa de adaptación para transformar los mensajes entre el dominio de la aplicación y el del dominio de HL7. Hace que este paso sea fácil proporcionando el framework para la conexión de sistemas dispares con los protocolos establecidos en los adaptadores y las herramientas de transformación de mensajes.

Utiliza una arquitectura basada en canales para conectar los sistemas con otros sistemas HL7. Los canales consisten en terminales (de entrada y de salida), filtros, y transformadores. Múltiples filtros y una cadena de transformadores se pueden asociar con un canal. La interfaz web de Mirth permite la reutilización de filtros y transformadores en múltiples canales.

Presenta amplia variedad de conectores (TCP/MLLP,Bases de datos, Archivos, JMS ,FTP/SFTP,SOAP (sobre HTTP). Es dependiente de la máquina virtual de Java en su versión 1.5. Permite la creación o utilización de filtros, perfiles de validación y transformadores. Todos los mensajes y transacciones se registran en una base de datos interna.

1.4.26 Subversion

Subversion es un sistema de control de versiones libre y de código fuente abierto. Maneja ficheros y directorios a través del tiempo. Hay un árbol de ficheros en un *repositorio* central. El repositorio es como un servidor de ficheros ordinario, excepto porque recuerda todos los cambios hechos a sus ficheros y directorios. Esto le permite recuperar versiones antiguas de sus datos, o examinar el historial de cambios de los mismos. En este aspecto, mucha gente piensa en los sistemas de versiones como en una especie de “máquina del tiempo”.

Puede acceder al repositorio a través de redes, lo que le permite ser usado por personas que se encuentran en distintos ordenadores. A cierto nivel, la capacidad para que varias personas puedan modificar y administrar el mismo conjunto de datos desde sus respectivas ubicaciones fomenta la colaboración. Se puede progresar más rápidamente sin un único conducto por el cual deban pasar todas las modificaciones. Subversion es un sistema general que puede ser usado para administrar cualquier conjunto de ficheros. Para usted, esos ficheros pueden ser código fuente— para otros, cualquier cosa desde la lista de la compra de comestibles hasta combinaciones de vídeo digital y más allá [40].
Es compatible además con sistemas operativos libres, propietarios y presenta además características que lo hacen muy atractivo: Versionado de directorios y metadatos, verdadero historial de versiones, envíos atómicos, manipulación consistente de datos y hackability; esta última hace que Subversion sea extremadamente fácil de mantener y reutilizable por otras aplicaciones y lenguajes.

1.4.27 Tortoise

TortoiseSVN es gratis. Está desarrollado bajo la Licencia Pública General GNU (GPL).

Es un cliente para el sistema de control de versiones *Subversion*. Maneja ficheros y directorios a lo largo del tiempo. Los ficheros se almacenan en un *repositorio* central. Esto permite que pueda recuperar versiones antiguas de sus ficheros y examinar la historia de cuándo y cómo cambiaron sus datos, y quién hizo el cambio [41].

TortoiseSVN se integra perfectamente en el shell de Windows (por ejemplo, el explorador). Lo que permite seguir trabajando con las herramientas que ya el usuario conoce y no tiene que cambiar a una aplicación diferente cada vez que necesite las funciones del control de versiones; aunque ni siquiera está obligado a usar el Explorador de Windows pues los menús contextuales de TortoiseSVN también funcionan en otros administradores de archivos.

1.4.28 AnkhSVN

AnkhSVN es un plug-in del sistema de control de versiones SubVersion que se integra al VS2005, y permite realizar las operaciones más comunes de control de versiones, directamente desde el entorno del VS2005. Dicha herramienta en su versión actual posee algunas características donde debemos fijar la atención, ejemplo de estas tenemos: Los nuevos proyectos pueden ser añadidos automáticamente al sistema de control de versiones, emite mensaje de alerta si al abrir una solución al VS2005 alguno de sus proyectos no están siendo supervisados por el SubVersion; los cambios en los ficheros de configuración son actualizados automáticamente sin la necesidad de reiniciar el VS2005; soporte completo para todo tipo de soluciones de dicho entorno; soporta el renombrado de carpetas en los proyectos; previene al explorador de soluciones de redibujarse cuando el AnkhSvn está escaneando los ficheros de la solución incrementando de esta forma el rendimiento de lectura de soluciones, etc.

1.4.29 Generador de acceso a datos (DAG)

DAG es una herramienta que puede ser utilizada por cualquier empresa de desarrollo de software, solo necesita un buen diseño del diagrama de UML. Es una aplicación de escritorio fácil de instalar y de utilizar, para trabajar con el mismo es necesario un conocimiento básico de UML. Desarrollada por estudiantes pertenecientes al área temática GeHos.

Dicha herramienta crea las tablas y establece las relaciones entre ellas, da la posibilidad de seleccionar los lenguajes donde se desee generar el código (código de aplicación, gestor de base datos), genera el Script BD que contiene la definición de las tablas y las funciones *select, insert, update, delete, búsqueda; g*enera las Clases persistentes, la capa de acceso a datos y las tablas de seguridad en caso de que sea necesario(o sea de que el usuario lo pida).

1.4.30 Case Studio 2.2

Case Studio es una herramienta profesional que permite diseñar tus propias bases de datos, facilitando herramientas para la creación de diagramas de relación, modelado de datos y gestión de estructuras.

Tiene soporte para trabajar con una amplia variedad de formatos de base de datos (Oracle, SQL, MySQL,

PostgreSQL, Access, etc.) y permite además generar scripts SQL, aplicar procesos de ingeniería inversa a bases de datos, usar plantillas de diseño personalizables y crear detallados informes en HTML y RTF.
A través de diagramas de relación brinda una visión más clara del contenido y estructura de la base de datos, facilitando la gestión y mantenimiento de la misma.

Dada la amplia gama de lenguajes y gestores que soporta, se vislumbra como una herramienta a utilizar por los diseñadores de Bases de datos que deseen un sistema, con soporte para múltiples SGBD, pues les permite transformar modelos anteriormente diseñados para un SGBD a otros.

A lo largo de este capitulo, se han expuesto conceptos asociados al proceso de gestión hospitalaria que facilitan la comprensión de la investigación. Se analiza el estado actual de estos procesos y se fundamentan y caracterizan un conjunto de tecnologías, herramientas y tendencias asociadas también a la posible solución propuesta para el problema científico que nos atañe.

CAPÍTULO 2: PATRONES PRESENTES EN LA SOLUCIÓN

Este capítulo es el resultado de la búsqueda y el análisis de la información vinculada al objeto de estudio, procesos a automatizar, patrones existentes asociados al campo de acción, tendencias y tecnologías a emplear en la construcción del sistema. Además se presentan en él aquellos patrones que se proponen incluir en el sistema, quedando abierta la propuesta de inclusión algún otro patrón que se adecue al dominio en cuestión.

El uso de patrones bajo determinadas circunstancias, hacen que nuestras soluciones de software sean comprensibles y fáciles de mantener; ganando en calidad y favoreciendo la comunicación y la creación de manuales para ingenieros de software. Es válido mencionar que no es objetivo de los patrones reemplazar la creatividad de los desarrolladores, ni reemplazarlos por herramientas Case automáticas. Al contrario, se reconoce la importancia del factor humano a la hora de crear y utilizar estos patrones como medio para llegar a comprender sistemas de software complejos.

2.1 Patrones de Arquitectura

Se encuentran relacionados a la interacción de objetos dentro o entre niveles arquitectónicos, se utilizan frente a problemas arquitectónicos, de adaptabilidad, requerimientos cambiantes, performance, modularidad y acoplamiento; proponiendo una solución basada en patrones de llamadas entre objetos (similar a los patrones de diseño), decisiones, criterios arquitectónicos y empaquetado de funcionalidad. Se ubican en la fase de diseño inicial del software.

2.1.1 Patrón de Capas

Descompone la aplicación en un conjunto de capas independientes y ordenadas jerárquicamente, cada nivel o capa usa los servicios de la capa inmediatamente inferior y ofrece servicios a la capa inmediatamente superior.

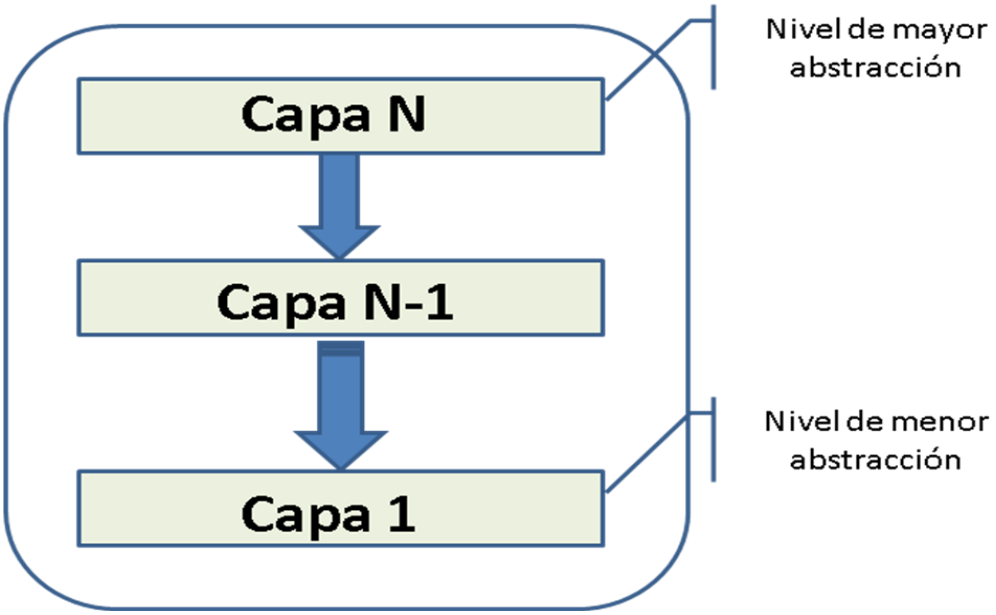


Figura No.1 Arquitectura en Capas.

El uso de este patrón brinda la posibilidad de reutilizar un mismo nivel en varias aplicaciones, permite la estandarización, el cambio de nivel no afecta el resto, ahora si no podemos desviarnos de algunos puntos interesantes a la hora de diseñar utilizando este patrón, y estos son: un número excesivo de niveles puede ser ineficiente y un nivel muy bajo hace que nuestras aplicaciones se vuelvan complejos e ineficientes. Si se hace un mal diseño, nos encontramos frente a la posibilidad de que cambios de funcionalidad se transmitan de un nivel a otro.

2.1.2 Mapeo de Datos

Este patrón propone definir una tabla (si se emplea una bases de datos relacional), para cada clase objeto persistente. Los atributos de los objetos que contienen tipos primitivos de datos (número, cadena, booleano y otros) se mapean en las columnas [42].

Si un objeto posee atributos exclusivamente de tipos primitivos de datos, el mapeo será simple. Pero la complejidad de la solución puede incrementarse un poco ya que los objetos pueden tener atributos que se

refieran a otros objetos complejos, mientras que el modelo relacional exige que los valores sean atómicos (primera forma normal) [43].

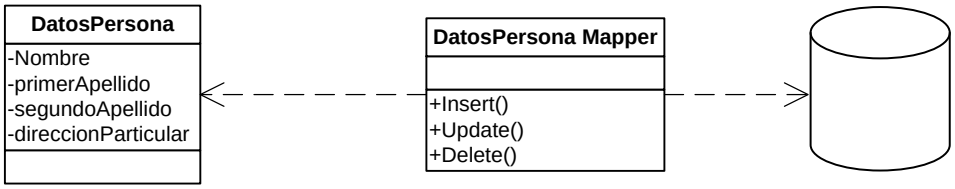


Fig. No. 2 Patrón Mapeo de Datos

2.2 Patrones de Diseño.

Se conciben como conceptos de ciencia de computación en general, independiente de aplicación. Se enfrentan a problemas con la claridad de diseño, multiplicación de clases, adaptabilidad a requerimientos cambiantes, entre otros, proponen soluciones mediante comportamientos de factoría, clase-responsabilidad-contrato (CRC), etc. Podemos ubicarlos en el refinamiento del diseño.

2.2.1 Experto.

Problema: ¿Cómo asignar responsabilidades, de la forma más eficiente?

Solución: Asignar una responsabilidad al experto en información: la clase que cuenta con la información necesaria para cumplir la responsabilidad.

Si estas asignaciones de responsabilidades se hacen en la forma adecuada, los sistemas tienden a ser más fáciles de entender, mantener y ampliar, lo que nos ofrece la garantía de poder reutilizar los componentes en futuras aplicaciones.

Explicación: Experto es un patrón que se usa más que cualquier otro al asignar responsabilidades; es un principio básico que suele utilizarse en el diseño orientado a objetos. Con él no se pretende designar una idea oscura ni extraña; expresa simplemente la “intuición” de que los objetos hacen cosas relacionadas con la información que poseen.[44]

Cuando el cumplimiento de una responsabilidad requiere de información distribuida en varias clases, aparecen “expertos parciales”, que colaboran con el cumplimiento de la responsabilidad en cuestión. En este caso de la existencia de “expertos parciales”, la interacción de estos se efectuará a través de mensajes para compartir el trabajo.

Este patrón como tantas otras cosas en la tecnología orientada a objetos, ofrece una analogía con el mundo real. Un ejemplo real sería el de una empresa, donde el director financiero para elaborar estado financiero general necesita de la información de cuentas por cobrar y cuentas por pagar que le brindan los encargados de generar dichos reportes individuales sobre créditos y deudas.

Beneficios:

- Se conserva el encapsulamiento, ya que los objetos se valen de su propia información para hacer lo que se les pide. Esto soporta un *bajo acoplamiento*, lo que favorece el hecho de tener sistemas más robustos y de fácil mantenimiento (*bajo acoplamiento* es un patrón GRASP que examinaremos más adelante) [45].

- El comportamiento se distribuye entre las clases que cuentan con la información requerida, alentando con ello definiciones de clases “sencillas” y más cohesivas que son fáciles de comprender y mantener. Así se brinda una *alta cohesión* (patrón que se explicará más adelante) [46].

2.2.2 Creador.

Problema: ¿Quién debería ser el responsable de crear una nueva instancia de alguna clase?

Solución: La responsabilidades de crear una instancia de la *clase A* se le dará a aquella *clase B*, en los siguientes casos:

- B agrega los objetos A.

- B contiene los objetos A.

- B registra las instancias de los objetos A.

- B utiliza específicamente los objetos A.
- B tiene los datos de inicialización que serán transmitidos hacia A cuando este objeto sea creado (B es experto en la creación de A).

Explicación: El patrón Creador guía la asignación de responsabilidades relacionadas con la creación de objetos, tarea muy frecuente en los sistemas orientados a objetos. El propósito fundamental de este patrón es encontrar un creador que debemos conectar con el objeto producido en cualquier evento. Al escogerlo como creador, se da soporte al bajo acoplamiento. [47]

Una forma de ampliar la explicación sería como sigue:

El agregado agrega la parte, el contenedor contiene el contenido, el registro registra. En un diagrama de clases se registran las relaciones muy frecuentes entre las clases. El patrón Creador indica que clase incluyente del contenedor o registro es idónea para asumir la responsabilidad de crear la cosa contenida o registrada. Desde luego, se trata tan sólo de una directriz [48].

Beneficios:

- Se brinda un soporte al *bajo acoplamiento*, lo cual supone menos dependencias respecto al mantenimiento y mejores oportunidades de reutilización. Es probable que el acoplamiento no aumente, pues la clase *creada* tiende a ser visible a la clase *creador*, debido a las asociaciones actuales que nos llevaron a elegirla como el parámetro adecuado.

2.2.3 Alta Cohesión

Problema: ¿Cómo mantener la complejidad dentro de los límites manejables?

La *solución* reside en asignar una responsabilidad de modo que la cohesión siga siendo alta. Debemos mencionar que la cohesión es una medida de cuan relacionadas y enfocadas están las responsabilidades de una clase, además de que una alta cohesión garantiza que clases con responsabilidades estrechamente relacionadas no realicen un trabajo enorme.

El patrón en referencia es un principio que debemos tener presente en todas las decisiones del diseño: es la meta principal que ha de buscarse en todo momento. Es un patrón evaluativo que el desarrollador aplica al valorar sus decisiones de diseño [49].

Ahora, existen formas de calificar el grado de cohesión, siendo estas: muy baja, baja, alta y moderada.

Un diseño con muy baja cohesión, es aquel en el que una clase es la responsable de operaciones correspondientes a áreas funcionales muy heterogéneas, sucediendo algo parecido con un diseño de baja cohesión, ya que en este caso la clase tiene la responsabilidad exclusiva de una tarea compleja dentro de un área funcional. En ninguno de los dos casos anteriores se delega o distribuyen las responsabilidades, dichas clases abarcan el volumen de las responsabilidades a realizar sin importar su complejidad o heterogeneidad.

Un diseño con un nivel de cohesión bajo o muy bajo, presenta problemas a la hora de comprender, reutilizar o conservar dichas clases, además de constantes afectaciones dadas por cambios en dicho diseño.

Un nivel moderado de cohesión implica que la clase posee un peso ligero pues agrupa áreas que están relacionadas lógicamente con el concepto de clases pero no entre ellas.

El alto nivel de cohesión, es presentado por las clases que tienen responsabilidades moderadas en un área funcional y colaboran con otras para llevar a cabo las tareas.

Los niveles de alta y moderada cohesión son aceptados, pero vale la pena aclarar que lo ideal es alta cohesión.

Una clase con mucha cohesión presenta *beneficios* tales como: facilidad de mantenimiento, comprensión y uso. Su alto grado de funcionalidad, combinada con una reducida cantidad de operaciones, también simplifica el mantenimiento y los mejoramientos. La ventaja que significa una gran funcionalidad también soporta un aumento de la capacidad de reutilización.

El patrón alta cohesión a menudo genera bajo acoplamiento y vale aclarar que una clase muy cohesiva puede destinarse a un propósito muy específico.

2.2.4 Bajo Acoplamiento.

Este patrón surge para contrarrestar la problemática que se enuncia como sigue: ¿Cómo dar soporte a una dependencia escasa y a un aumento de la reutilización?, la solución se enfoca a asignar una responsabilidad para mantener bajo acoplamiento.

Aquellas clases con alto (o fuerte acoplamiento), recurren a muchas otras para realizar sus responsabilidades, lo cual no es conveniente ya que esto puede acarrear consigo problemas como: cambios locales como resultado de cambios en las clases afines, difícil comprensión al aislarse, su reutilización se dificulta pues es necesaria la presencia de las otras clases.

Ampliando la caracterización de este patrón, es un principio que no podemos descartar durante las decisiones del diseño, como meta principal a lograr siempre. Se puede catalogar como un patrón evaluativo que el diseñador aplica al juzgar sus decisiones de diseño [50].

El bajo acoplamiento estimula la asignación de responsabilidades de forma tal que la inclusión de éstas no incremente el acoplamiento, creando clases más independientes y con mayor resistencia al impacto de los campos, que aumentan la productividad y la posibilidad de reutilización.

Es válido aclarar que este patrón no puede verse de forma independiente a los patrones Experto y Alta Cohesión, sino más bien incluirse como otro de los principios del diseño que influyen de forma determinante a la hora de la asignación de responsabilidades.

2.2.5 Controlador.

A la hora de diseñar nos hacemos la siguiente pregunta: ¿Quién debería encargarse de tender un evento del sistema?

Esto lo podemos resolver tomando como lineamientos el de asignar la responsabilidad del manejo de un mensaje de los eventos del sistema a una clase que represente una de las siguientes opciones: Controlador de fachada, controlador de tareas o controlador de casos de uso.

La mayor parte de los sistemas reciben eventos de entrada externa, los cuales generalmente incluyen una interfaz gráfica para el usuario (IGU) operado por una persona. Otros medios de entrada son los mensajes

externos entre ellos un conmutador de telecomunicaciones para procesar llamadas o las señales procedentes de sensores como sucede en los sistemas de control de procesos.

En todos los casos, si se recurre a un diseño orientado a objetos, hay que elegir los controladores que manejen esos eventos de entrada. Este patrón ofrece una guía para tomar decisiones apropiadas que generalmente se aceptan.

La misma clase controlador debería utilizarse con todos los eventos sistémicos de un caso de uso, de modo que podarnos conservar la información referente al estado del caso**.** Esta información será útil por ejemplo para identificar los eventos del sistema fuera de secuencia, Puede emplearse varios controladores en los casos de uso.

2.2.6 Fabricación Pura.

Patrón que asigna un conjunto altamente cohesivo de responsabilidades a una clase artificial que no representa nada en el dominio del problema: una clase creada para dar soporte a una alta cohesión, un bajo acoplamiento y reutilización.

Para diseñar utilizando este patrón debe buscarse ante todo un gran potencial de reutilización, asegurándose que las responsabilidades de las clases a agrupar en la nueva clase, sean pequeñas y cohesivas, sería lo mismo decir que, estas clases deben tener responsabilidades de *granularidad fina*.

El uso concreto de este patrón se puede evidenciar cuando producto del negocio de nuestros sistemas, necesitamos realizar un proceso, que está compuesto, como proceso, a la vez por procesos independientes, se crean clases que se convierten es una especie de objeto de función central.

Utilizando este patrón se gana en soporte a una Alta Cohesión pues se distribuyen las responsabilidades en clases de granularidad fina, centradas en un conjunto muy específico de tareas afines.

2.2.7 Fábrica Abstracta

Este patrón proporciona una interfaz para crear familias de objetos relacionados o que dependen entre sí, sin especificar sus clases concretas.

Se puede usar Fábrica Abstracta cuando: Un sistema debe ser independiente de cómo se crean, componen y representan sus productos, una familia de objetos productos relacionados está diseñada para ser usada conjuntamente, y es necesario hacer cumplir esta restricción o quiere proporcionar una biblioteca de clases de productos y solo quiere revelar sus interfaces, no sus implementaciones.

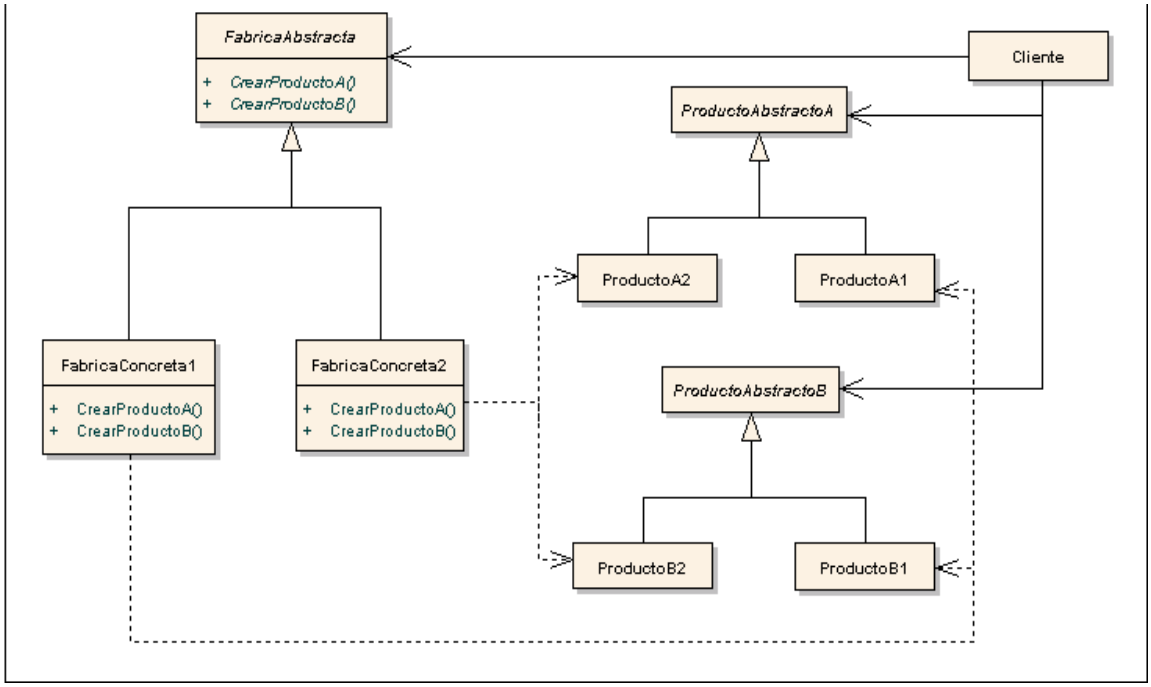


Fig. No. 3 Patrón Fábrica Abstracta

Dentro de su estructura encontramos diferentes clases participantes, las cuales citamos a continuación

1) FabricaAbstracta**:** Declara una interfaz para operaciones que crean objetos producto, abstractos.

2) FabricaConcreta: Implementa las operaciones para crear objetos producto concreto.

3) ProductoAbstracto: Declara una interfaz para un tipo de objeto producto.

4) ProductoConcreto: Define un objeto producto para que sea creado por la fábrica correspondiente. Implementa la interfaz ProductoAbstracto.

5) Cliente: Sólo usa interfaces declaradas por las clases FabricaAbstracta y ProductoAbstracto.

Usualmente se crea una única instancia de una clase FabricaConcreta en tiempo de ejecución, quien crea objetos producto que tienen una determinada implementación. Para crear diferentes objetos producto, los clientes deben usar una fábrica concreta diferente.
FabricaAbstracta delega la creación de objetos producto en su subclase FabricaConcreta.

Este patrón:
1) Separa las clases concretas: Ayuda a controlar las clases de objetos que crea una aplicación, encapsula la responsabilidad y el proceso de creación de objetos producto, aísla a los clientes de las clases de implementación, manipulan las instancias a través de sus interfaces abstractas. Los nombres de las clases producto quedan aisladas en la implementación de la fábrica concreta; no aparecen en el código cliente.
2) Facilita el intercambio de familias de productos: La clase de una fábrica concreta solo aparece una vez en una aplicación, cuando se crea. Esto facilita cambiarla (como crea una familia completa de productos, cambia toda de una vez).
3) Promueve la consistencia entre productos: Se diseñan objetos producto en una familia para trabajar juntos.
4) Es difícil dar cabida a nuevos tipos de productos: Ampliar las fábricas abstractas para producir nuevos tipos de productos no es fácil ya que la interfaz FabricaAbstracta fija el conjunto de productos que se pueden crear (implica cambiar la clase FabricaAbstracta y todas sus subclases).

2.2.8 Patrón Singleton.

El uso de este patrón garantiza que una sola clase sólo tenga una instancia y proporciona un punto de acceso global a ella.

Al mencionarlo, surge la duda de si sería importante, que algunas clases tuviesen exactamente una instancia. ¿Cómo lo podemos asegurar y que ésta sea fácilmente accesible? Una variable global no previene de crear múltiples instancias de objetos. Una solución mejor es hacer que sea la propia clase la responsable de su única instancia, quien debe garantizar que no se pueda crear ninguna otra (interceptando las peticiones para crear nuevos objetos) y proporcione un modo de acceder a ella.

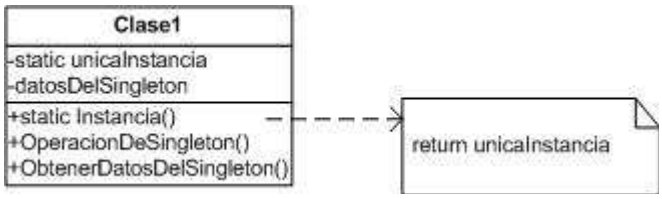


Fig. No. 4 Patrón Singleton

Este patrón, propuesto para usarse en conjunto con la Fábrica Abstracta en el middleware entre acceso a datos y lógica del negocio, permite el acceso controlado a una única instancia, *e*ncapsulando esta y garantizando tener un control estricto sobre como y cuando acceden a ella los clientes. Reduce el espacio de nombres mejorando el uso de las variables globales y mantiene controlada la creación de objetos, disminuyendo el uso de memoria al tener una sola instancia que se usa en todo el contexto de la aplicación

2.3 Patrones de Idiomas

Se utilizan en los flujos de implementación, mantenimiento y despliegue, comúnmente reconocidos como estándares de codificación y proyecto, describen como codificar y representar operaciones comunes bien conocidas en un nuevo ambiente, o a través de un grupo, brindan legibilidad y predictibilidad.

A continuación, se describen instrucciones a seguir en el desarrollo y codificación del sistema.

2.3.1 Instrucciones de nomenclatura

La existencia de un modelo de nomenclatura coherente, es uno de los elementos más importantes en cuanto a previsibilidad y capacidad de descubrimiento en una biblioteca de clases o sistema. El uso y el conocimiento generalizados de estas instrucciones de nomenclatura deberían eliminar la mayoría de las preguntas más frecuentes de los usuarios y desarrolladores. En este acápite se proporcionan instrucciones de nomenclatura, algunas de las reglas generales con relación a los estilos de mayúsculas, distinción entre mayúsculas y minúsculas y elección de palabras.

En cuanto a estilos de mayúsculas, es necesario que se utilicen las tres convenciones siguientes para poner en mayúsculas los identificadores.

2.3.1.1 Mayúsculas y Minúsculas Pascal

La primera letra del identificador y la primera letra de las siguientes palabras concatenadas están en mayúsculas. El estilo de mayúsculas y minúsculas Pascal se puede utilizar en identificadores de tres o más caracteres. Por ejemplo:

```
ColorFondo
```

2.3.1.2 Mayúsculas y Minúsculas Camel

La primera letra del identificador está en minúscula y la primera letra de las siguientes palabras concatenadas en mayúscula. Por ejemplo:

```
colorFondo
```

2.3.1.3 Mayúsculas

Todas las letras del identificador van en mayúsculas. Utilice esta convención sólo para identificadores que estén formados por dos o menos letras. Por ejemplo:

```
System.IO

System.Web.UI
```

Además, puede que sea necesario utilizar mayúsculas en los identificadores para mantener la compatibilidad con esquemas existentes de símbolos no administrados, donde los caracteres en

mayúsculas se utilizan con frecuencia en valores de constantes y enumeraciones. En general, estos símbolos no deben ser visibles fuera del ensamblado en el que se utilizan.[6]

A continuación se presentan como se propone la utilización de cada una de las convenciones anteriormente descritas.

2.3.1.4 Distinción de mayúsculas y minúsculas

Para evitar confusiones y garantizar la interoperación entre lenguajes, se siguen estas reglas con respecto a la distinción entre mayúsculas y minúsculas:

No se utilizarán nombres que requieran distinción entre mayúsculas y minúsculas. Los componentes se deben poder utilizar en los lenguajes que distinguen, y en los que no distinguen, entre mayúsculas y minúsculas. Los lenguajes que no hacen esta distinción no pueden diferenciar, dentro del mismo contexto, dos nombres que difieren sólo en el uso de mayúsculas y minúsculas. Por consiguiente, se debe evitar esta situación en los componentes o clases creados.

No se deben crear dos espacios de nombres con nombres que difieran sólo en las mayúsculas y minúsculas. Por ejemplo, un lenguaje que no haga distinción entre mayúsculas y minúsculas no distingue entre las dos declaraciones siguientes de espacio de nombres.

```
namespace gehos.data;

namespace Gehos.Data;
```

No se declararán funciones con nombres de parámetros que difieran sólo en las mayúsculas y minúsculas. El siguiente ejemplo es incorrecto.

[6] Consultar el Anexo no. 3 para profundizar en las las reglas de uso de mayúsculas y los diferentes tipos de identificadores.

```
void MiFuncion(string a, string A)
```

No se crearán un espacio de nombres con nombres de tipos que difieran sólo en las mayúsculas y minúsculas. En el siguiente ejemplo, Persona p y PERSONA p son nombres de tipo incorrectos ya que difieren sólo en el uso de las mayúsculas y minúsculas.

```
Gehos.Data.Entidades.Persona p

Gehos.Data.Entidades.PERSONA p
```

No debe crear un tipo con nombres de propiedades que difieran sólo en las mayúsculas y minúsculas. En el siguiente ejemplo, int Color y int COLOR son nombres de propiedades incorrectos ya que difieren sólo en el uso de las mayúsculas y minúsculas.

```
int Color {get, set}

int COLOR {get, set}
```

No se deben crear un tipo con nombres de métodos que difieran sólo en las mayúsculas y minúsculas. En el siguiente ejemplo, calcular y Calcular son nombres de métodos incorrectos ya que difieren sólo en el uso de las mayúsculas y minúsculas.

```
void calcular()

void Calcular()
```

2.3.1.5 Abreviaturas

Para evitar confusiones y garantizar la interoperación entre lenguajes, se disponen las siguientes reglas con respecto a la utilización de abreviaturas:

No utilice abreviaturas ni contracciones como parte de nombres de identificadores. Por ejemplo, utilice GetWindow en vez de GetWin.

No utilice acrónimos que no estén aceptados en el campo de la informática.

Si es necesario, utilice acrónimos conocidos para reemplazar nombres en frases largas. Por ejemplo, utilice UI para interfaz de usuario y OLAP para procesamiento analítico en línea.

Cuando utilice acrónimos, utilice el estilo de mayúsculas y minúsculas Pascal o Camel en acrónimos de más de dos caracteres. Por ejemplo, use HtmlButton o htmlButton. Sin embargo, deberá utilizar mayúsculas en acrónimos que tengan sólo dos caracteres, por ejemplo System.IO en vez de System.Io.

No utilice abreviaturas en nombres de identificadores o parámetros. Si tiene que utilizar abreviaturas, utilice las Mayúsculas y minúsculas Camel en abreviaturas de dos o más caracteres, aunque esto contradiga la abreviatura estándar de la palabra.

2.3.1.6 Elección de Palabra

Evite utilizar nombres de clases que dupliquen los espacios de nombres utilizados habitualmente en .NET Framework. Por ejemplo, no utilice ninguno de los nombres siguientes como nombres de clases: System, Collections, Forms o UI.

Además, evite utilizar identificadores que entren en conflicto con las palabras claves del lenguaje. [7]

[7] Consultar anexo no. 4.

2.3.1.7 Evitar confusión de Nombres de Tipos

Los distintos lenguajes de programación utilizan términos diferentes para identificar los tipos administrados básicos. Los diseñadores de bibliotecas de clases deben evitar utilizar terminología específica del lenguaje. Los desarrolladores deben seguir las reglas que se describen en esta sección para evitar la confusión de nombres de tipos.

Utilice nombres que describan el significado del tipo, en vez de nombres que describan el tipo. En el caso poco frecuente de que un parámetro no tenga ningún otro significado semántico aparte del significado del tipo, utilice un nombre genérico. Por ejemplo, una clase que admite la escritura de diversidad de tipos de datos en una secuencia puede tener los métodos siguientes.

```
[C#]
void Write(double value);
void Write(float value);
void Write(long value);
void Write(int value);
void Write(short value);
```

No se deben crear nombres de métodos con terminología específica del lenguaje, como se muestra en el siguiente ejemplo.

```
[C#]
void Write(double doubleValue);
void Write(float floatValue);
void Write(long longValue);
void Write(int intValue);
void Write(short shortValue);
```

En el caso excepcional de que sea necesario crear un método con un nombre único para cada tipo básico de datos, utilice nombres de tipos universales.[8]

Por ejemplo, una clase que admita la lectura de diversidad de tipos de datos en una secuencia puede tener los siguientes métodos.

```
[C#]
double ReadDouble();
float ReadSingle();
long ReadInt64();
int ReadInt32();
short ReadInt16();
```

[8] Consulte el anexo no. 5 para profundizar en los nombres de tipos básicos y las sustituciones universales.

Es preferible utilizar el ejemplo anterior, en vez de la alternativa específica de lenguaje que se muestra a continuación.

```
[C#]
double ReadDouble();
float ReadFloat();
long ReadLong();
int ReadInt();
short ReadShort();
```

2.3.1.8 Instrucciones de Nomenclatura de Espacios de Nombres

Por regla general, en los espacios de nombres se utiliza el nombre de la compañía seguido del nombre de la tecnología y, opcionalmente, la característica y el diseño como se muestra a continuación.

```
CompanyName.TechnologyName[.Feature][.Design]
Por ejemplo:
GeHos.Seguridad
GeHos.Seguridad.Roles
```

Al incluir un prefijo en los nombres de espacios de nombres que contengan el nombre de una compañía o marca de reconocido prestigio, se evita la posibilidad de publicar dos espacios de nombres que tengan el mismo nombre. Por ejemplo, Microsoft.Office es un prefijo adecuado para las clases de automatización de Office que proporciona Microsoft.

Debe utilizarse un nombre de tecnología reconocida y estable en el segundo nivel de un nombre jerárquico. Utilice una jerarquía organizativa como base para la jerarquía de espacios de nombres. Asigne un nombre a un espacio de nombres que contenga los tipos que proporcionan funcionalidades en tiempo de diseño para un espacio de nombres base con el sufijo Roles. Por ejemplo, el Espacio de nombres GeHos.Seguridad.Roles contiene las clases de Roles y clases relacionadas para diseñar aplicaciones basadas en GeHos.Seguridad.

Un espacio de nombres anidado debe tener una dependencia en los tipos del espacio de nombres contenedor. Por ejemplo, las clases de GeHos.Seguridad.Roles dependen de las clases de GeHos.Seguridad. No obstante, las clases de GeHos.Seguridad no dependen de las clases en GeHos.Seguridad.Roles.

En los espacios de nombres, se debe utilizar el estilo de Mayúsculas y minúsculas Pascal y separar los componentes lógicos con puntos, como en Microsoft.Office.PowerPoint. Si la marca utilizada no emplea la regla de mayúsculas y minúsculas tradicional, siga el sistema que utiliza la marca, aunque no siga la regla Pascal. Por ejemplo, los espacios de nombres NeXT.WebObjects y ee.cummings muestran las variaciones correctas de utilización de mayúsculas y minúsculas con respecto a la regla Pascal.

Los nombres de espacios de nombres deben utilizarse en plural, siempre que sea correcto semánticamente. Por ejemplo, utilice System.Collections en vez de System.Collection. Las excepciones a esta regla son los nombres y abreviaturas de marcas. Por ejemplo, utilice System.IO en vez de System.IOs.

No utilice el mismo nombre para un espacio de nombres y para una clase. Por ejemplo, no proporcione un espacio de nombres Debug y una clase Debug.

Por último, tenga en cuenta que el nombre de un espacio de nombres no tiene que ser análogo al nombre del ensamblado. Por ejemplo, si asigna el nombre MyCompany.MyTechnology.dll a un ensamblado, no es necesario que contenga un espacio de nombres MyCompany.MyTechnology.

2.3.1.9 Instrucciones de Nomenclatura de Clases

En las reglas siguientes se describen las instrucciones de nomenclatura de clases:

Utilizar un sustantivo o un sintagma nominal para asignar un nombre a una clase, el estilo de Mayúsculas y minúsculas Pascal y las abreviaturas con moderación.

No utilizar un prefijo de tipo, como C para clase, en un nombre de clase. Utilice, por ejemplo, el nombre de clase FileStream en vez de CFileStream.

De vez en cuando, es necesario proporcionar un nombre de clase que comience con la letra I, aunque la clase no sea una interfaz. Esto es correcto siempre que I sea la primera letra de una palabra que forme parte del nombre de la clase. Por ejemplo, IdentityStore es un nombre de clase correcto.

Cuando sea apropiado, utilice una palabra compuesta en el nombre de una clase derivada. La segunda parte del nombre de la clase derivada debe ser el nombre de la clase base. Por ejemplo, ApplicationException es un nombre correcto para una clase derivada de la clase Exception, pues ApplicationException es una clase de Exception. En esta regla se debe utilizar la lógica. Por ejemplo, Button es un nombre adecuado para una clase derivada de Control. Aunque un botón es una clase de control, si incluye Control como parte del nombre de una clase alargaría el nombre innecesariamente.

A continuación, se incluyen algunos ejemplos de clases con nombres correctos:

```
[C#]

public class FileStream

public class Button

public class String
```

2.3.1.10 Instrucciones de Nomenclatura de Interfaces

En las reglas siguientes se describen las pautas de nomenclatura de interfaces:

Asignar nombres a interfaces utilizando sustantivos, sintagmas nominales o adjetivos que describan su comportamiento. Utilizar el estilo de Mayúsculas y minúsculas Pascal y las abreviaturas con moderación. Incluir un prefijo con la letra I en los nombres de interfaces para indicar que el tipo es una interfaz.

Utilizar nombres similares cuando defina un par clase/interfaz, donde la clase es una implementación estándar de la interfaz. Los nombres deben ser distintos sólo en el prefijo "I" del nombre de la interfaz. No utilice el carácter de subrayado (_).

A continuación, se incluyen algunos ejemplos de interfaces con nombres correctos:

```
public interface IServiceProvider

public interface IFormatable
```

2.3.1.11 Instrucciones de nomenclatura de tipos de enumeración

El tipo de valor de enumeración (*Enum*) se hereda de la Clase Enum. En las reglas siguientes se describen las instrucciones de nomenclatura de enumeraciones:

Utilizar el estilo de *Mayúsculas y minúsculas Pascal* en los nombres de valores y tipos *Enum*, las abreviaturas con moderación, no utilice el sufijo *Enum* en nombres de tipo *Enum***,** Utilice un nombre en singular para la mayoría de los tipos *Enum*, pero utilice un nombre en plural para los tipos *Enum* que son campos de bits, agregue siempre *FlagsAttribute* a un tipo *Enum* de campo de bits.

2.3.1.12 Instrucciones de nomenclatura de campos estáticos

En las reglas siguientes se describen las instrucciones de nomenclatura de campos estáticos:

Utilizar sustantivos, sintagmas nominales o abreviaturas de nombres al asignar nombres a campos estáticos, el estilo de *Mayúsculas y minúsculas Pascal*, no debe utilizarse un prefijo de notación húngara

en nombres de campos estáticos. Se recomienda utilizar propiedades estáticas en lugar de campos estáticos públicos cada vez que sea posible.

2.3.1.13 Instrucciones de nomenclatura de parámetros

Es importante seguir estas instrucciones de nomenclatura de parámetros, ya que las herramientas de diseño visual que proporcionan ayuda contextual y funcionalidad de exploración de clases muestran en el diseñador los nombres de los parámetros de métodos a los usuarios. En las reglas siguientes se describen las instrucciones de nomenclatura de parámetros:

Utilizar el estilo de *Mayúsculas y minúsculas Camel* para los nombres de parámetros.

Utilizar nombres de parámetros descriptivos. Los nombres de parámetros deben ser lo suficientemente descriptivos como para que el nombre y el tipo del parámetro se puedan utilizar para determinar su significado en la mayoría de los escenarios. Por ejemplo, las herramientas de diseño visual que proporcionan ayuda contextual muestran los parámetros de los métodos a los programadores mientras escriben. Por tanto, los nombres de los parámetros deberían ser lo suficientemente descriptivos en este escenario como para permitir a los programadores suministrar los parámetros adecuados.

El significado del parámetro deber estar presente en el nombre de éste, nunca utilice nombres que describan el tipo de parámetro. Las herramientas de desarrollo deben proporcionar información descriptiva sobre el tipo de parámetro. Por tanto, el nombre del parámetro también sirve para describir su significado. Utilice los nombres de parámetros basados en tipos con moderación y sólo cuando sea correcto.

No incluya un prefijo en los nombres de parámetros con notación húngara de tipo.

A continuación, se incluye un ejemplo de parámetros con nombres correctos:

```
[C#]

Persona SeleccionarPersona(string personaNombre)
```

2.3.1.14 Instrucciones de nomenclatura de métodos

En las reglas siguientes se describen las instrucciones de nomenclatura de métodos:

Utilizar verbos o sintagmas verbales al asignar nombres a los métodos, y el estilo de *Mayúsculas y minúsculas Pascal*.

A continuación, se incluyen algunos ejemplos de métodos con nombres correctos:

```
EliminarTodos()

Ejecutar()
```

2.3.1.15 Instrucciones de nomenclatura de propiedades

Cuando se utilicen propiedades, se seguirán instrucciones de nomenclatura como: utilice un sustantivo o un sintagma nominal al asignar nombres a las propiedades, el estilo de *Mayúsculas y minúsculas Pascal*, no utilice la notación húngara, es conveniente crear una propiedad con el mismo nombre que el tipo subyacente correspondiente. Por ejemplo, si declara una propiedad con el nombre Color, el tipo de propiedad también deberá llamarse *Color*. Vea el ejemplo que se muestra más adelante en este tema.

En el siguiente ejemplo de código se muestra cómo asignar nombres correctos a propiedades.

```
[C#]
public class ClaseEjemplo
{  public Color ColorFondo
   {  // El código para las operaciones de Get y Set va aquí. }}
```

En el siguiente ejemplo de código se muestra cómo proporcionar una propiedad con el mismo nombre que el tipo.

```
 [C#]
public enum Color
{   // aquí va el código del enum.}
public class Control
{   public Color Color    {  get {// código.}  set {// código.}   }}
```

La reutilización es una de las tendencias actuales en la producción de software, por lo que se requiere de software bien diseñados e implementados, con vistas a evitar dificultades durante el desarrollo, y consiguiendo agilizar el proceso.

Partiendo de la serie de patrones descritos, que se utilizarán para diseñar el sistema en cuestión, se brinda solución a uno de los problemas más importantes del desarrollo de sistemas de información: implementar código con calidad, que cumpla con las propiedades de claridad y sencillez, en la estructura y diseño de la solución, que sea fácil de leer por otros desarrolladores y por tanto, fácil de mantener.

CAPÍTULO 3: DESCRIPCION DE LA ARQUITECTURA

En este capitulo, se hace una propuesta de solución al problema planteado en el diseño teórico de la investigación, tomando como base las descripciones de tecnologías y herramientas del Capítulo 1 de la presente investigación.

La solución se divide en dos subtópicos correspondiendo cada uno de ellos a los artefactos fundamentales que define y construye el arquitecto de software.

A través de estos artefactos (Línea Base y Documento de Descripción de la Arquitectura) se describe la solución arquitectónica del sistema, incluyendo además otros artefactos definidos por la metodología RUP.

3.1 Línea Base de la Arquitectura.

3.1.1 Introducción.

La línea base contiene los elementos imprescindibles para lograr la mayor abstracción en el diseño arquitectónico de la aplicación.

En ella se exponen los estilos arquitectónicos seleccionados para la aplicación, así como los componentes, conectores y sus configuraciones. Se enumeran los patrones, las restricciones de software y hardware, las tecnologías y herramientas utilizadas en el diseño de la arquitectura.

Propósito

El propósito de la línea base de la arquitectura es proporcionar la información necesaria para estructurar el sistema desde el mayor nivel de abstracción.

Los usuarios son:

- El equipo de arquitectos: La utiliza como guía para la toma de decisiones arquitectónicas, y son los encargados de su refinamiento.
- El equipo de desarrolladores: La utiliza como guía para la implementación.

- Clientes: Pueden encontrar en ella la garantía de la calidad y el conocimiento de la tecnología y herramientas seleccionadas.

Alcance

La línea base de la arquitectura describe detalladamente el organigrama de la arquitectura centrada en los estilos arquitectónicos, los frameworks, las tecnologías y herramientas que soportan los estilos y patrones especificados que se utilizan en el sistema y que cumplen con los objetivos y restricciones de este.

3.1.2 Concepciones Generales.

El sistema se desarrollará utilizando RUP como metodología. Esta metodología define tres puntos fundamentales que guían el diseño de la arquitectura, estos son:

- Guiado por los casos de uso.
- Centrado en la arquitectura.
- Iterativo e incremental.

Estos puntos garantizan que la solución satisfaga las necesidades del cliente, ya que al centrar el desarrollo en los casos de uso permite concebir el sistema en módulos teniendo en cuenta para cada uno de ellos el conjunto de operaciones que dada su prioridad y complejidad son escogidas para el desarrollo de la primera iteración del sistema. Es válido que con la primera iteración no culmina el desarrollo, en las iteraciones restantes y definidas en el plan de desarrollo del proyecto, se definen y estructuran las restantes necesidades del sistema y se refinan las desarrolladas en iteraciones anteriores.

3.1.3 Organigrama de la Arquitectura.

El sistema propuesto para desarrollar, Sistema de Gestión Hospitalaria, entra dentro de la categoría de software para la gestión, este debe poseer una base de datos que le permita el almacenamiento de la información que luego se utilizará para una toma de decisiones, que en este tipo de sistema debe poseer el grado máximo de confiabilidad y garantía, pues los principales beneficiados o afectados son seres humanos.

En el sistema en cuestión las principales operaciones están enfocadas al movimiento, tratamiento y diagnóstico del paciente dentro de la institución hospitalaria; estas operaciones se catalogan y organizan de acuerdo, principalmente, a los departamentos identificados.

- Inscripción – Admisión (IA): Encargado del control y gestión de la inscripción, admisión y movimientos del paciente, durante su paso por el hospital.
- Cuerpo de Guardia (CG): Encargado de la consulta de los pacientes, sean de urgencia o no, recibidas por el cuerpo de guardia de la institución.
- Laboratorio (LIS): Gestión de los exámenes y pruebas practicadas al paciente.
- Banco de Sangre del Hospital (BS): Gestión de pedidos y donaciones de sangre. Maneja la historia de donaciones y transfusiones del paciente o donante.
- Farmacia (FAR): Gestiona medicamentos y drogas del hospital.
- Archivo HC (AHC): Mantiene el control de la HC física, brinda información exacta de su movimiento dentro de la institución.
- Bloque Quirúrgico (BQ): Gestiona las planificaciones e información concerniente a las intervenciones quirúrgicas del paciente sin importar la naturaleza de estas.
- Configuración (CFG): Gestiona todos los nomencladores, los recursos humanos del hospital, así como la seguridad del sistema.

Además, estos sistemas necesitan desarrollar las operaciones convencionales de procesamiento de datos, lo que genera una lógica de negocio densa. Es extremadamente importante para estos sistemas que la información se gestione con una seguridad y calidad extrema. Las interrupciones en el sistema deben ser evitadas, deben ser capaces de asimilar cambios en la lógica de los procesos, minimizando el impacto de estos cambios. También, deben poder comunicarse con el resto de los sistemas que cubren y mejoran la atención al paciente.

A partir de los elementos y necesidades mencionadas anteriormente la arquitectura del sistema se ha organizado a partir de la combinación de los estilos en capas, orientados a objetos y servicios.

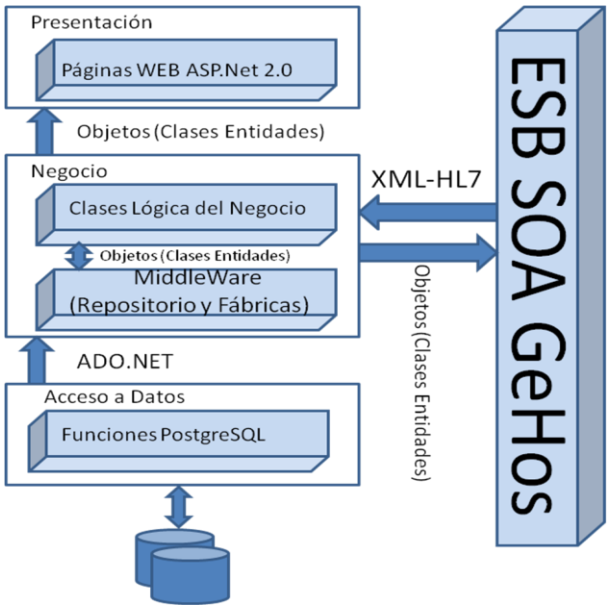


Fig. No. 5 Estructura Modular del Sistema de Gestión Hospitalaria.

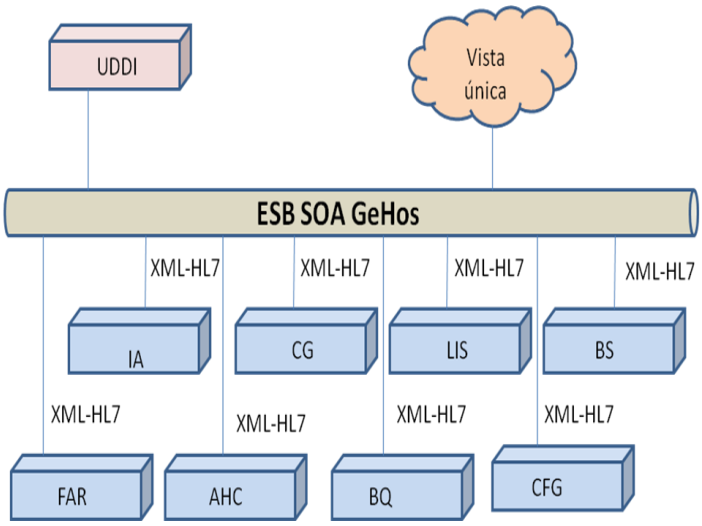


Fig. No. 6 Estructura Global del Sistema de Gestión Hospitalaria.

Componentes o elementos

Los principales componentes que distinguen la arquitectura son:

- Presentación:

  Esta capa contiene las interfaces necesarias para que el usuario y el sistema intercambien toda la información necesaria para el proceso de gestión, compuesta por páginas WEB ASP.NET. Interactúa con la capa inferior, mediante invocación de los métodos que conformar la lógica del negocio, produciendo un intercambio de objetos.

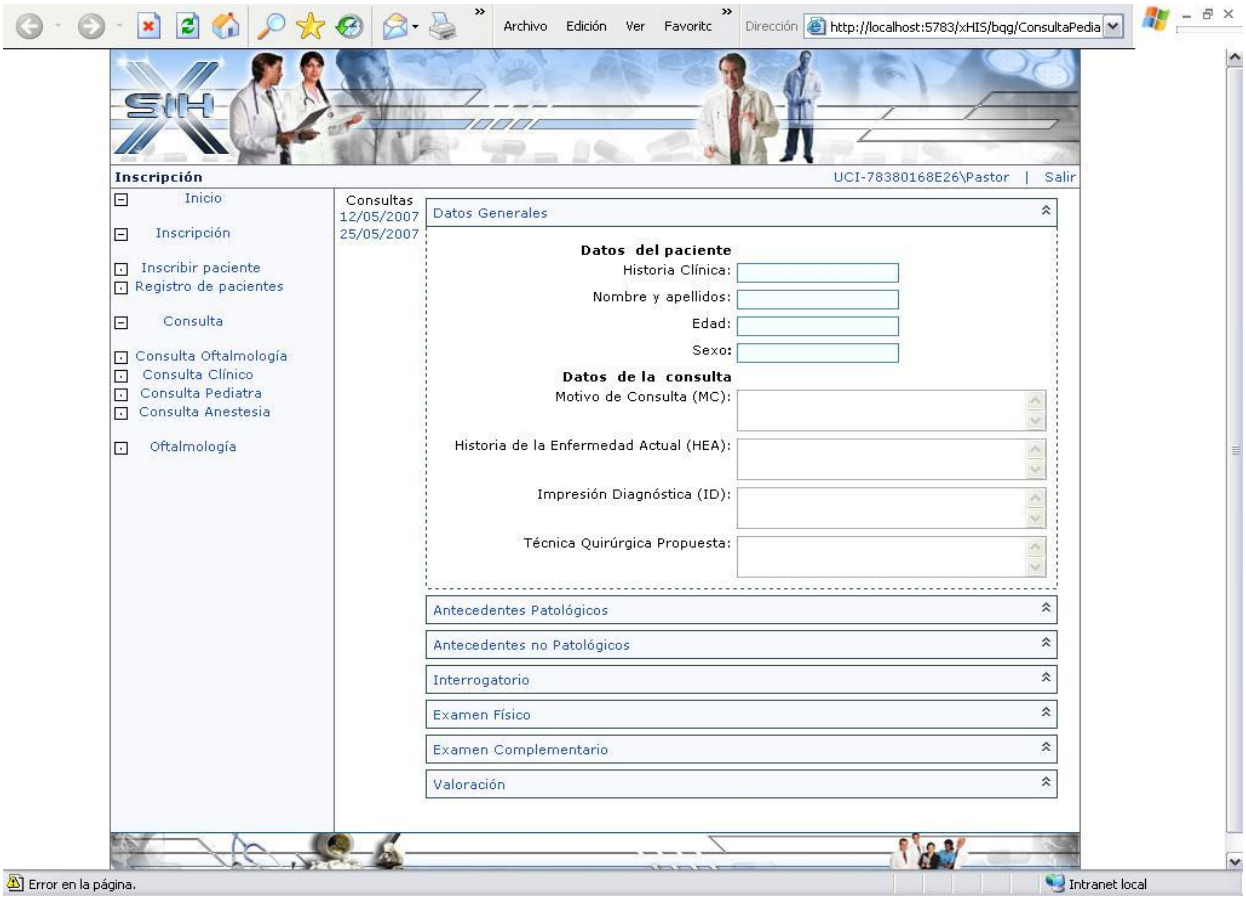


Fig. No. 7 Interfaz de Usuario (Ejemplo)

- Negocio:

  Esta capa se divide en dos subcapas encargadas de resolver la lógica del negocio.

  Clases del Negocio: Almacena todas las clases encargadas de representar la lógica del negocio, y se controla la seguridad en cada invocación de los métodos. Interactúa con la subcapa Middleware invocando los métodos definidos en las clases Repositorios, el intercambio consiste al igual que en el caso anterior en colecciones de objetos.

  Middleware: Es una subcapa con la función de acceder al repositorio de datos, es encargada de ejecutar la lógica de acceso a datos, contiene dos paquetes de clases:

  Repositorios: Los repositorios son los encargados de manejar las colecciones de Objetos Comunes (Entidades de la BD representados como objetos) y realizar operaciones sobre ellas.

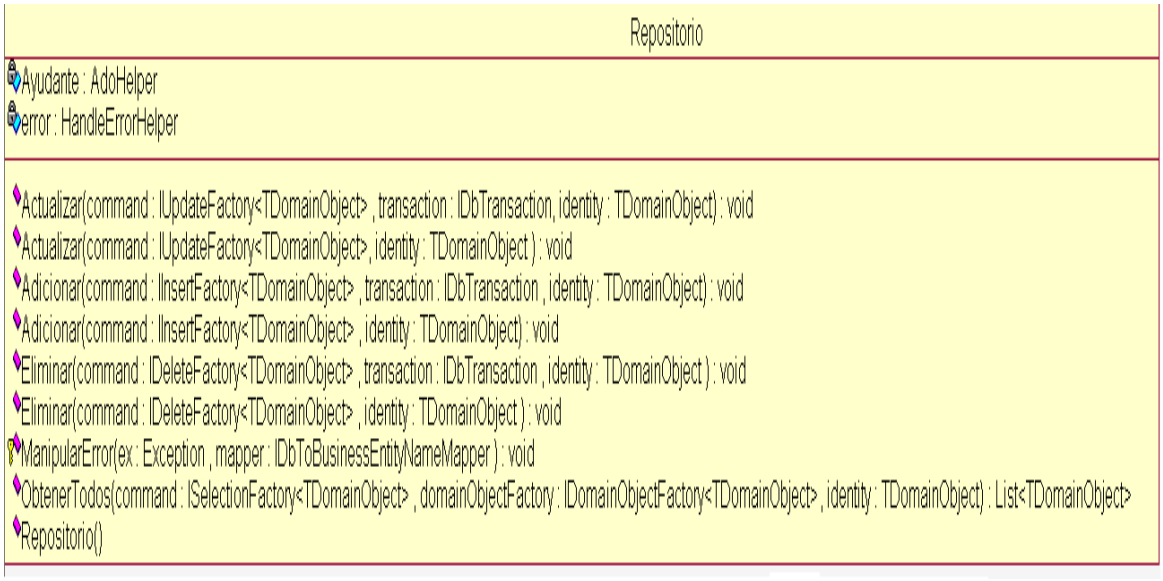


Fig. No. 8 Clase Repositorio

Fábricas de Objetos: Paquetes de clases que contiene la funcionalidad necesaria para acceder a la base de datos y realizar las operaciones que se explican a continuación. Por cada entidad mapeada desde la base de datos, existen cuatro clases que heredan de las interfaces ISelectionFactory: encargada de llevar a cabo el proceso de selección de una entidad determinada en la base de datos, IInsertFactory: encargada del proceso de inserción, IDeleteFactory: Encargada de suprimir una entidad determinada, IUpdateFactory: encargada de actualizar atributos de los objetos en la BD, existe además por cada entidad otra clase que hereda de la clase interfaz IDomainObjectFactory y es la encargada de realizar el proceso de mapeo de las entidades (convertir tupla a tupla el resultado de un proceso de selección en la entidad a la que corresponde).

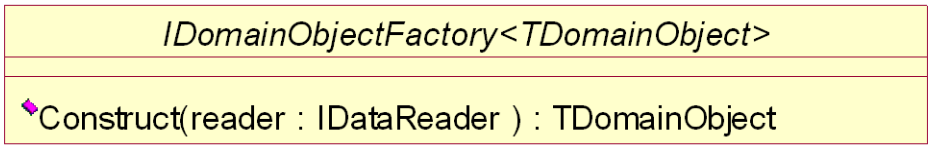


Fig. No. 9 Clase Interfaz IDomainObjectFactory

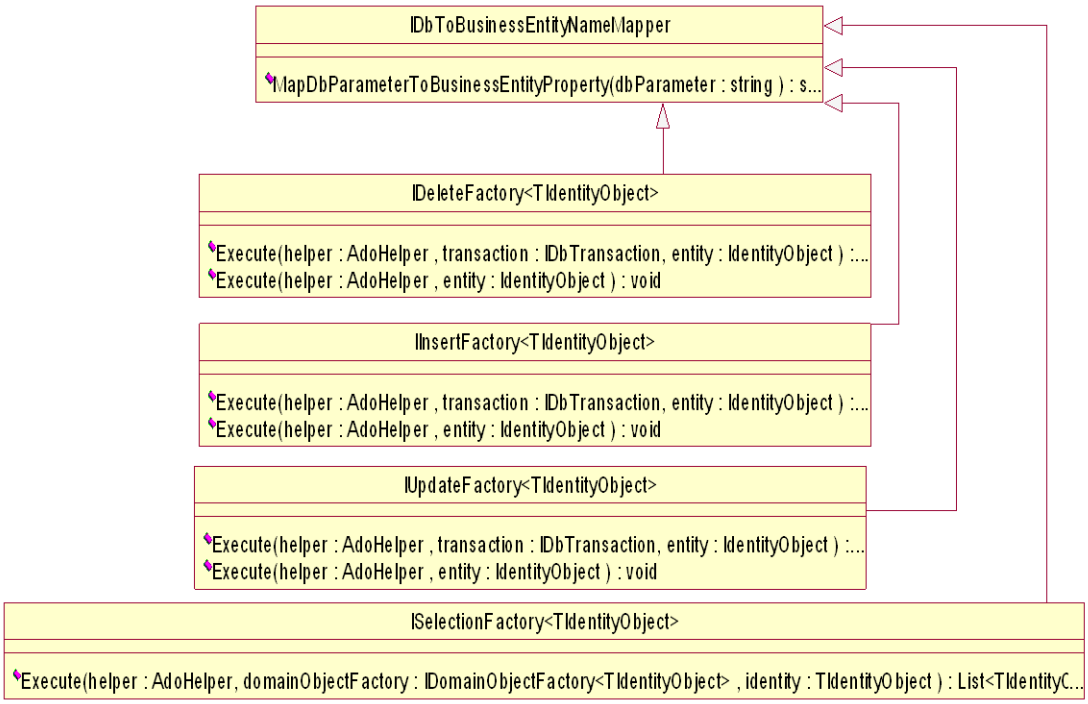


Fig. No. 10 Relación entre las diferentes Fábricas y la clase DbToBusinessEntityNameMapper

- Acceso a datos: Ubicada en el servidor de Bases de Datos, está compuesta por el conjunto de funciones programadas en lenguaje NpgSql que se enlazan con el middleware para ejecutar las solicitudes sobre el servidor de Bases de Datos. Engloban toda la lógica de acceso a datos.

- ESB SOA GeHos: Conjunto de Servicios WEB que cubren las funcionalidades exigidas por la lógica de negocio; utilizados para facilitar la integración con sistemas externos ó módulos del sistema, que no se puedan conectar directamente con el negocio de los restantes módulos por problemas de configuración de la red o localizaciones de estos. La información facilitada por estos consisten en mensajes HL7 sobre XML construidos y revisados utilizando Chameleon.

Conectores / Configuraciones

Los conectores, son las formas de comunicación entre los componentes o elementos definidos, es la forma en que está representada la información que fluye entre estos.

- Presentación – Negocio.
  Las funcionalidades que brinda el negocio son accedidas mediante invocación de métodos y las respuestas de estos constituyen colecciones de objetos.
- Negocio – Middleware
  Las funcionalidades del Middleware son accedidas mediante invocación de métodos pertenecientes a las clases Repositorios y las respuestas de estos constituyen colecciones de objetos.
- Negocio – ESB SOA GeHos
  La interacción con los servicios WEB ubicados en el Bus de Servicios se hará utilizando mensajes HL7 sobre XML.[9]
- Middleware – Acceso a datos.
  Se realiza, utilizando las clases agrupadas dentro del Paquete Fábrica de Objetos, estas a su vez hacen referencia a las operaciones de un ayudante (helper) perteneciente a ADO.NET, que se encarga de manejar las conexiones y transacciones a la Base de Datos.
- ESB SOA GeHos – Negocio.
  Las funcionalidades que brinda el negocio, serán reutilizadas por los servicios WEB, mediante la invocación de métodos y las respuestas de estos constituyen colecciones de objetos.
- Acceso a datos – Base de Datos.
  La comunicación se establece a nivel de gestor de bases de datos a través de sus funciones internas.

[9] Consultar el Anexo no. 9 donde se brinda un ejemplo de mensaje HL7 utilizando XML.

Restricciones

La principal restricción que se les impone a los componentes, es que los ubicados en capas superiores, solo pueden hacer uso de los que están ubicados en capas inferiores o verticales, que se extiendan a su nivel.

3.1.4 Frameworks de desarrollo

Otro nivel de abstracción de la arquitectura, son los frameworks o marcos de trabajo, que no son más que arquitecturas definidas para un determinado dominio de la aplicación y contienen un número determinado de componentes implementados, con sus interfaces bien definidas que pueden ser reutilizados o redefinidos.

A partir de la organización de la arquitectura a través de los estilos definidos, se propone la utilización del Framework .NET 2.0. Se utilizará a todos los niveles y capas de la aplicación, reutilizando y redefiniendo de éste interfaces y componentes. Sus características se explican detalladamente en el acápite 1.4.10.

3.1.5 Patrones de Arquitectura.

Los patrones de arquitectura escogidos para el desarrollo del sistema son: Arquitectura en Capas y Mapeo de datos. Sus características, ventajas, desventajas y fundamentación se encuentran en los acápites 2.1.1 y 2.1.2 respectivamente.

3.1.6 Lenguaje, Tecnologías y Herramientas de apoyo al desarrollo.

Para lograr minimizar los tiempos de desarrollo y gastos en licencias, garantizando el desarrollo de un producto de software que se adapte a las necesidades del sistema de salud cubano y permita a los clientes finales un entorno de trabajo amigable y flexible. Se proponen las siguientes herramientas y tecnologías a utilizar, utilizando como base lo expuesto en el acápite 1.4:

- Gestor de base datos: PostgreSQL 8.2
- Lenguajes de programación: ASP.net, C# y JavaScript.
- Metodología AJAX.

- Metodología de Desarrollo de SW: Metodología RUP con notación UML
- Herramienta CASE de Modelado UML: Rational Suite 2003
- Servidor Web: Apache 2.0 (Mod_Mono)
- Navegador : IE6 y FireFox (compatibilidad absoluta)
- Servicios WEB, con formato HL7.
- Chameleon V4.1.
- Framework .Net v2.0, Mono 1.2.4
- IDE's de Desarrollo: VS2005.
- Herramienta Diseño WEB: Macromedia DreamWeaver 8.0
- Herramienta de Control de Versiones: SubVersion v1.3.1, Tortoise 1.4.2, AnkSVN 1.0
- Herramienta de Generación de Capa de Acceso a Datos: DAG.
- Herramienta de Diseño de Bases de Datos: Case Studio 2.2
- Herramienta de Análisis de Migración: MOMA V 1.2.4.

C# y ASP.NET, se seleccionan por ser lenguajes de programación optimizados para el trabajo con la plataforma, permiten crear código muy eficiente, en aquellos puntos de la aplicación que son críticos y acceder a las interfaces de programación de aplicaciones (API) existentes. Además, permite aprovechar la experiencia creciente por parte del equipo de desarrollo; el alto rendimiento y escalabilidad, la seguridad mejorada sobre otras tecnologías web existentes. Así como, la posibilidad de desplegar los sistemas desarrollados con esta tecnología, en ambientes tanto Windows como Linux. JavaScript se selecciona por su universal aceptación (96 %) entre los desarrolladores y por la limitante de portabilidad de Visual Basic Script, al ser compatible solamente con Internet Explorer.

PostgreSQL, se selecciona por presentar características que lo hacen especialmente adecuado al tipo de sistema de gestión que se desea construir. Además de otras que se describen a continuación: Write - Ahead Logging (WAL) que permite asegurar la atomicidad y durabilidad de la información. Dos de las características ACID de las bases de datos. Usando WAL las modificaciones son escritas primero a un log o bitácora antes de ser aplicadas a la base de datos. Esto garantiza que las operaciones de deshacer o rehacer se puedan ejecutar sin problemas y garantizar la integridad de los datos.

Teniendo en cuenta que se trabajará en el área del software para la salud, y que la información que se manipulará será sensible y relevante, por ser información relativa a las personas; así como a los estudios y diagnósticos asociados a éstas, conviene entonces, escoger un DBMS que cumpla con las funciones de respaldo, y PostgreSQL deviene candidato ideal, además de que no se ha querido encarecer el sistema final usando un gestor como Oracle o SqlServer.

Dentro de todas herramientas HL7, se propone Chameleon 4.1, tomando como base las características expuestas en el subacápite 1.4.24.2, dentro de las cuales resaltan la amplia gama de versiones del estándar que soporta, compatibilidad con plataformas libres y propietarias y con los lenguajes de programación más utilizados en la actualidad, ha estado en el mercado por más de 8 años y cuenta con un respaldo empresarial de prestigio. Es la única de las herramientas que se encuentra certificada por HL7 Organization.

3.2 Descripción de la Arquitectura

3.2.1 Introducción

Propósito

Este documento tiene como propósito lograr una mejor comprensión del sistema, organizar el desarrollo, fomentar la reutilización, contribuyendo de esta forma a una evolución del sistema más rápida y eficaz.
Para esto se proporciona una comprensión arquitectónica global del sistema, utilizando las vistas arquitectónicas definidas por la metodología RUP, que muestran las diferentes características del sistema.
Además, debe capacitar a los desarrolladores, directivos, clientes y otros usuarios en la compresión al detalle del sistema propuesto, facilitando su participación en el mismo.

Alcance

Este documento influye en la toma de decisiones arquitectónicas correspondientes a los sistemas de gestión de la información hospitalaria que se desarrollen en el proyecto GEHOS de la facultad 7 de la Universidad de Ciencias Informáticas. Abarca todos los módulos del sistema de gestión.

3.2.2 Representación Arquitectónica

Para la representación de la arquitectura propuesta, el documento presenta las vistas que a continuación se mencionan. Serán modeladas utilizando Rational Rose 2003 Enterprise Edition.

- Vista de casos de uso.
- Vista lógica.
- Vista de implementación.
- Vista de despliegue.

3.2.3 Objetivos y restricciones de la arquitectura.

Las metas y restricciones que definen características de importancia para la arquitectura serán descritas a continuación.

3.2.3.1 Requerimientos de Hardware:

Para la implantación del sistema se propone la siguiente configuración de hardware en dependencia del número de clientes. La comunicación entre los servidores de negociación y bases de datos debe permitir manejar un gran volumen de Información por lo que se requiere que tenga una tarjeta de red que cumpla con el estándar 100Base-T y pueda soportar además múltiples conexiones a estos, debe estar protegida contra fallos de corriente y conectividad y programar los tiempos de realización de copias de seguridad.

3.2.3.1.1 Estaciones de trabajo.

- Mouse/Teclado PS 2 o USB
- CPU: Pentium II a 800 Mhz o superior
- RAM: 128 Mb, recomendado 256 Mb
- Almacenamiento: 10 Gb de memoria o superior.
- Impresora

3.2.3.1.2 Servidor de aplicaciones.

- Sistema operativo: Linux
- Servidor web: Apache 2.x
- Arquitectura del hardware: Intel

Clientes	CPU	Memoria RAM	Capacidad de almacenamiento y Tipo
Menos de 150	1 GHz o superior (Recomendado Dual-Core)	512 Mb	20 Gb (IDE)
150 a 500	2 GHz o superior **Dual CPU** (Recomendado **Quad-Core**)	1 Gb	30 Gb (IDE)
500 a 1000	2 GHz o superior **Dual CPU** (Recomendado **Quad-Core**)	2 Gb	30 Gb (Recomendado SCSI RAID)
c/ 1000	2 GHz c/ 1000 o superior	1Gb c/ 1000 o superior	10 Gb c/1000 (Recomendado SCSI RAID)

Tabla No. 1 Configuración de hardware para servidor de aplicaciones dependiendo de cantidad de clientes.

3.2.3.1.3 Servidor de Bases de Datos.

- Sistema operativo: Linux
- Arquitectura del hardware: Intel
- Gestor de bases de datos: PostgreSQL

Clientes	CPU	Memoria RAM	Capacidad de almacenamiento y Tipo
Menos de 150	1 GHz o superior (Recomendado Dual-Core)	512 Mb	20 Gb (IDE)
150 a 500	2 GHz o superior **Dual CPU** (Recomendado **Quad-Core**)	1 Gb	30 Gb (IDE)
500 a 1000	2 GHz o superior **Dual CPU** (Recomendado **Quad-Core**)	2 Gb	30 Gb (Recomendado SCSI RAID)
c/ 1000	2 GHz c/ 1000 o superior	1Gb c/ 1000 o superior	10 Gb c/1000 (Recomendado SCSI RAID)

Tabla No. 2 Configuración de hardware para servidor de bases de datos dependiendo de cantidad de clientes.

3.2.3.2 Requerimientos de Software:

3.2.3.2.1 Estaciones de trabajo

1. Sistema Operativo: Windows 98 o superior, GNU-Linux
2. Navegador Web: IE6, Firefox.

3.2.3.2.2 Servidor WEB

1. Sistema Operativo: Windows 98 o superior, GNU-Linux
2. Apache 2.0 o superior (mod_mono instalado)

3.2.3.2.3 Servidor de Bases de Datos.

1. Sistema Operativo: Windows 98 o superior, GNU-Linux
2. PostgreSQL 8.2

3.2.3.3 Redes

1. La red existente en las instalaciones debe ser capaz de soportar transacciones de paquetes de información de al menos 30 máquinas simultáneamente.

3.2.3.4 Seguridad

1. La seguridad se tratará desde la fase de diseño del sistema.
2. Se garantizará un fuerte tratamiento de excepciones.
3. Parte de la seguridad corre por parte de los Frameworks propuestos (.Net, Mono).
4. Los usuarios solamente tendrán acceso a las tablas y Bases de Datos a las cuales se les designe, según el rol que desempeñan, no se cuentas de usuarios con privilegios administrativos para realizar las conexiones entre servidores.
5. Se registrarán y auditarán las trazas dejadas por los usuarios al realizar las diferentes operaciones en el sistema. La programación de las auditorías se establecerá según corresponda.

3.2.3.5 Portabilidad, escalabilidad, reusabilidad

1. Cada módulo del sistema se desarrollará de forma tal, que pueda ser reutilizado o ser capaz de funcionar por sí solo y no como parte del paquete general.
2. La aplicación se construirá utilizando estándares internacionales y patrones, para facilitar su integración futura, con componentes desarrollados por cualquier empresa y garantizar posibilidades de un mantenimiento ágil.
3. La documentación de la arquitectura, debe ser reutilizable para poder documentarla como una familia de productos.
4. El sistema deberá poder ser usado desde cualquier sistema operativo.

5. El sistema deberá hacer un uso racional de los recursos de hardware, sobre todo en estaciones de trabajo clientes.
6. El sistema se desarrollará utilizando .Net Framework 2.0, pero el despliegue se realizará sobre la plataforma Mono versión 1.2.4.
7. De acuerdo a las condiciones económicas del país, el sistema deberá minimizar la cantidad de gastos de despliegue, así como contar con la capacidad de acomodarse a sus necesidades.

3.2.3.6 Restricciones de acuerdo a la estrategia de diseño.

1. El diseño de la aplicación, se hará aplicando programación orientada a objetos.
2. Se utilizarán las tecnologías que brindan los framework definidos:
    a. Programación orientada a objetos.
    b. En la subcapa de Middleware, se utilizará ADO.NET y la Interfaz de programación de Aplicaciones (API) de PostgreSQL (NpgSql.dll).

3.2.3.7 Integración de los componentes y su comunicación

El código dentro un mismo componente, utiliza llamadas a métodos o eventos de forma directa. La comunicación entre diferentes componentes se realiza mediante llamadas a servicios web o de forma directa a nivel de negocio. En caso de utilizarse servicios web, la información que es transmitida debe cumplir con los estándares internacionales establecidos para facilitar la integración entre nuevos componentes y otros sistemas hospitalarios. La base de datos debe ser accedida, de forma directa mediante controladoras y los componentes reusados son integrados mediantes interfaces sencillas.

3.2.3.8 Mecanismos de la Arquitectura para futuras modificaciones y extensiones.

El modelo de la vista de implementación describe la distribución de los diferentes componentes y capas de abstracción.
La alteración parcial o completa de algunos de ellos no debe afectar el correcto funcionamiento de los demás. La implementación de los diferentes patrones de diseño propuestos garantiza lo anterior.

Para el desarrollo de la capa de acceso a datos de los componentes se utiliza DAG, un sistema desarrollado para la generación total o parcial de esta capa, reduciendo los tiempos de desarrollo sin afectar el negocio del sistema. Además, permite acoplar al sistema cualquier sistema gestor de bases de datos relacionales o simplemente utilizar una unidad física como archivo para el almacenamiento de la información.

3.2.3.9 Herramientas de Desarrollo

1. Para el modelado de todo el proceso ingenieril, se utilizará el Rational Rose 2003 Enterprise Edition que demanda 256 MB de memoria RAM.
2. IDE de desarrollo, Visual Studio 2005 que demanda para su funcionamiento recomendado 512 MB de RAM, integrado con los plugins siguientes:
    a. AnkhSVN 1.0 para el control de versiones.
3. Servidor de bases de datos: PostgreSQL 8.2 (512 MB de RAM)
4. Servidor de Negociaciones: Apache 2.X con mod_aspnet (512 MB de RAM)
5. Servidor de versiones: SubVersion.
6. Chameleon Interfaceware.V4.1.
7. Case Studio 2.2 que demanda 256 MB de memoria RAM para su correcto funcionamiento.

3.2.3.10 Estructura del equipo de Desarrollo

El equipo de trabajo está distribuido de la siguiente forma:[10]

- Jefe de proyecto
- Líder de desarrollo
- Arquitecto
- Analista principal
- Implementador Integrador
- Planificador

[10] Consultar Anexo no. 10 para profundizar sobre la organización del equipo de desarrollo.

- Diseñador de bases de datos
- Contador
- Gestión de la configuración
- 8 Módulos de desarrollo
- 1 Grupo de Investigación
- 1 Grupo para el control de la calidad y prueba del sistema.

Por cada módulo la distribución es la siguiente:

- Jefe de proyecto
- Analista
- Implementador
- Diseñador de bases de datos

Los módulos de desarrollo son los siguientes:

- Inscripción y Admisión
- Bloque Quirúrgico
- Consulta externa
- Laboratorio
- Farmacia
- Banco de sangre
- Configuración y Seguridad
- Archivo HC

3.2.3.10.1 Configuración de los puestos de trabajo por roles

- Analista
    - PC con mouse y teclado.
    - Instalación Rational Rose Suite 2003.
    - Paquete de Office, para manejar la documentación.
- Implementador
    - PC con mouse y teclado, 512 MB de RAM.
    - VS2005
    - .NET Framework V.2.0

- Documentador
    - PC con mouse y teclado.
    - Instalación Rational Rose Suite 2003, para modelar casos de prueba.
    - Paquete de Office, para manejar la documentación.
- Diseñador BD.
    - PC con mouse y teclado.
    - Case Studio 2.22, para modelar bases de datos.

3.2.4 Vista de Casos de Uso

La vista de casos de uso nos brinda información de escenarios y casos de usos de interés para cada iteración del ciclo de desarrollo. Se describen en esta los casos de uso de interés arquitectónico que encapsulan la funcionalidad del sistema. Los casos de uso arquitectónicamente significativos son aquellos que sirven para validar la arquitectura propuesta y describen las funcionalidades imprescindibles para el sistema.

Módulo de Configuración.

- Autenticar Usuario
- Gestionar Usuarios
- Gestionar Roles de Usuarios
- Gestionar Nomencladores.
- Consultar Nomencladores

Estos casos de uso son prioritarios en el sistema ya que brindan funcionalidades necesarias para la puesta en marcha de la aplicación, además de que constituyen la base de seguridad de la aplicación y a partir de ellos se puede gestionar la traza de las operaciones realizadas en el sistema, los nomencladores que se utilizan en varios o todos los módulos del sistema, así como los datos del personal, en este caso de salud, de los cuales dependen varias operaciones.

Los casos de usos que se presentan y comentan a continuación son la base para el posterior funcionamiento del sistema, no se puede realizar ninguna de las operaciones propuestas si no se cuenta con la base conceptual para realizarlas.
A partir de estas funcionalidades, se puede comprobar que la arquitectura funciona para los elementos fundamentales, además de su comportamiento estable.

Módulo de Inscripción-Admisión

- **Gestionar inscripción:** Este Caso de uso tiene como principal objetivo inscribir a los pacientes en el hospital, luego de estar inscritos ya podrán atenderse en cualquiera de las secciones del mismo, así como recibir todos los servicios que brinda la institución.
- **Gestionar ingreso:** Este caso de uso se inicia cuando se quiere realizar un ingreso, modificar o reubicar a un paciente o en caso de que se necesite anular el ingreso.
- **Localizar camas disponibles:** El caso de uso se invoca cuando se va a efectuar un ingreso y hace falta ubicar a un paciente en un servicio determinado. Los servicios tienen varias salas y estas a su vez tienen camas que pueden estar disponibles u ocupadas, en este último caso el paciente tendría que ingresar en otra sala que tenga camas disponibles o en otro servicio, siendo el ingreso catalogado como fuera de servicio.
- **Buscar paciente:** El caso de uso es iniciado cuando el auxiliar de admisión necesita ingresar, trasladar o egresar a un paciente y le hace falta localizarlo para poder realizar los movimientos pertinentes.
- **Gestionar traslado:** El caso de uso se inicia cuando el auxiliar de admisión va a realizar un traslado de un paciente de un servicio o sala a otro. Este caso de uso permite trasladar a un paciente, permutar pacientes y anular el traslado.
- **Gestionar egreso:** El caso de uso se inicia cuando el auxiliar de admisión va a realizar un egreso de un paciente, ya sea por alta médica, por fallecimiento o por traslado. Permite además realizar egresos masivos y anular egresos.

Módulo Farmacia

- **Ver Disponibilidad:** Busca información de los productos disponibles. Se muestran las disponibilidades de los productos que presenta la farmacia para su distribución.
- **Registrar Prescripción Paciente:** Registra la prescripción medica orienta a cada paciente para la posterior solicitud de los productos.
- **Registrar Solicitud:** Registra en el sistema las solicitudes hechas por los servicios atendiendo a la disponibilidad.
- **Registrar Consumo:** Registra los consumos por servicios y pacientes que se producen en los centros solicitantes.
- **Registrar Devoluciones:** Registra las devoluciones de productos sin uso, rotos o vencidos solicitados por el servicio.
- **Dispensar Productos:** Dispensa los medicamentos solicitados por los servicios a farmacia.
- **Recepcionar Devoluciones:** Da entrada al sistema a las devoluciones echas por los servicios solicitantes.
- **Realizar inventario:** Comprueba la existencia en farmacia de los productos y compararlo con lo registrado en el sistema.
- **Controlar Producto:** Registra los datos de entrada o salida de algún producto existente en farmacia.
- **Recibir Pedidos y Distribuciones:** Inserta en el sistema algún pedido de farmacia que ha sido entregado o alguna distribución de productos de los proveedores.
- **Gestionar Pedidos:** Elabora un pedido según el déficit de productos en farmacia para su posterior solicitud a los proveedores.
- **Imprimir Pedidos:** Imprime la solicitud de pedido de medicamentos.
- **Gestionar Datos de Stock Farmacia:** Calcular, introduce o actualiza algún parámetro de stock por producto o propio de la farmacia.
- **Gestionar Datos de Stock Centro:** Calcula, introduce o actualiza algún parámetro de stock por producto o propio del servicio.
- **Gestionar Productos:** Insertar, actualiza, elimina o muestra características de los productos.

- **Consultar Informes Farmacia:** Visualiza informes de entradas y salidas de productos hechos por la farmacia.
- **Consultar Informes Centro** Visualiza informes de entradas y salidas de productos hechos por los servicios, así como sus consumos.

Módulo Cuerpo de Guardia.

- **Gestionar Recepción:** Gestiona todas las consultas entrantes al cuerpo de guardia clasificándolas para un mejor manejo de las prioridades de atención médica y solicitando en algunos casos servicios de enfermería para la agilización de la consulta.
- **Gestionar Consulta:** Controla los registros de las consultas de los pacientes que lleguen al cuerpo de guardia ya sea por urgencia o por la consulta del cuerpo de guardia. Ingresa los datos a los pacientes, los remite a enfermería, a servicios complementarios, sala de observación o a la sala UCIE.

Módulo de Historia Clínica (HC)

- **Gestionar HC:** El caso de uso clasifica y elimina las HC, registra su E/S, brinda una ubicación dentro del archivo, permite unirlas en caso de que el paciente posea varias HC o realiza alguna búsqueda.
- **Buscar HC:** El caso de uso permite buscar una HC por cualquiera de sus campos.
- **Crear Reporte:** El caso de uso permite hacer reportes concernientes a los procesos descritos en el caso de uso Gestionar HC filtrados por un rango de fechas que define el usuario.

Módulo de Laboratorio

- **Emitir Solicitud:** Emitir una solicitud de análisis.
- **Recepcionar Solicitud de análisis:** Recepcionar una solicitud de análisis, o sea cuando el paciente se presenta en el laboratorio para realizarse su análisis, la recepcionista verifica que se haya emitido su solicitud de análisis.
- **Tomar Datos de Muestra:** Tomar los datos de la muestra como el nombre, el estado, si necesita ser almacenada.
- **Registrar datos de almacenaje:** Poner los datos de almacenaje a una muestra que lo necesite.

- **Emitir Resultado:** Emitir el resultado de un examen después de analizarlo.
- **Buscar Solicitud:** Buscar una solicitud de análisis de un paciente por cualquier motivo.
- **Buscar Análisis:** Buscar un análisis para emitir un resultado una vez analizada la muestra.
- **Liberar Resultado:** Liberar un resultado después de verificar que no tenga errores.

Módulo Banco de Sangre Hospital

- **Gestionar Ficha de Donación:** Registra, elimina o modifica las características físicas y el estado de salud de los donantes en cada una de sus donaciones.
- **Gestionar Solicitudes:** Registra, cancela y actualiza una solicitud de sangre o sus componentes al banco de sangre, para un paciente que requiera una posible transfusión.
- **Registrar Grupo Sanguíneo:** Registra el grupo sanguíneo y el factor de un paciente, datos que serán útiles cuando se vayan a transfundir.
- **Atender Solicitudes:** Atiende las solicitudes realizadas por otras áreas del hospital al banco de sangre, teniendo en cuenta la disponibilidad del stock, y las prioridades de asignación de unidades a pacientes receptores.
- **Gestionar Stock:** Registra las entradas de unidades al sistema ya sea por captura inicial de existencias o entradas externas, también se registran las salidas de unidades del sistema, especificando las causas y se establecen los límites de almacenamiento del stock.

Módulo Bloque Quirúrgico

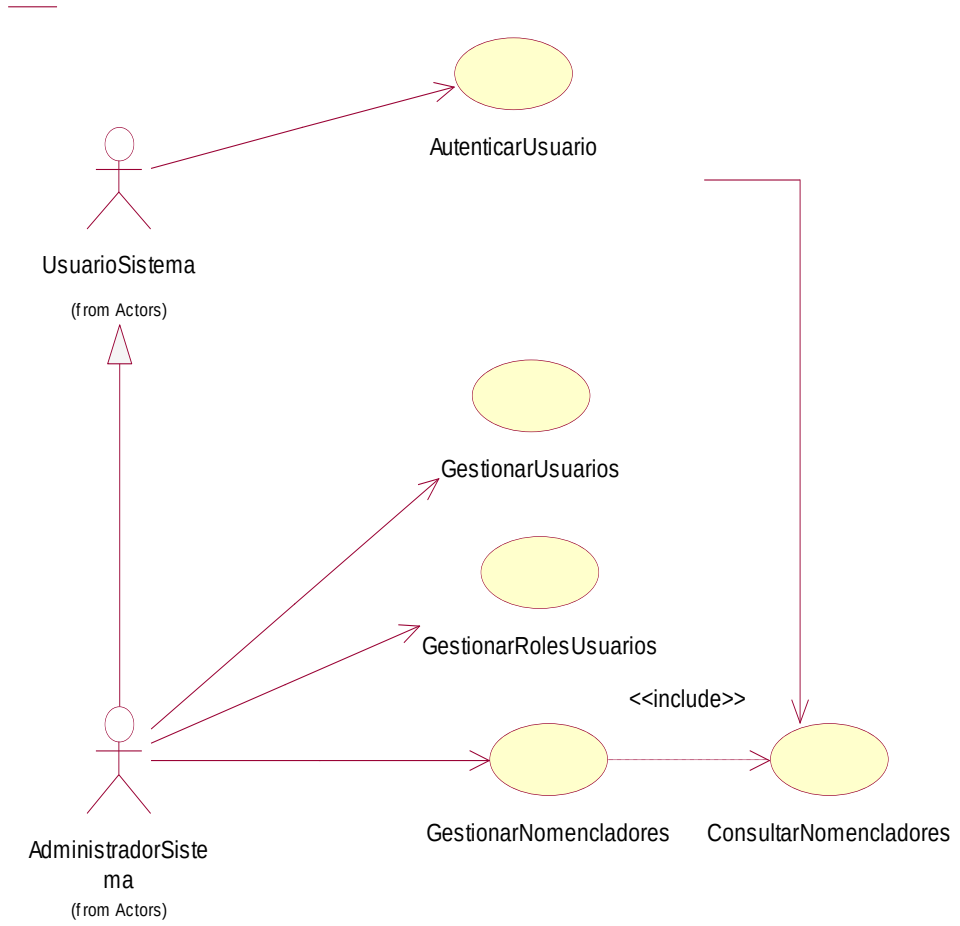
- **Gestionar Anuncio Operatorio:** Su principal objetivo es crear o modificar el anuncio operatorio de un paciente con los datos requeridos.
- **Gestionar Anuncio Operatorio Oftalmológico:** Su principal objetivo es crear o modificar el anuncio operatorio oftalmológico de un paciente con los datos requeridos específicos de la especialidad.
- **Cambiar estado del Anuncio Operatorio:** Permite posponer o suspender un anuncio operatorio especificando la causa.

- **Gestionar Hoja del Especialista:** Su objetivo principal es permitir al especialista crear y modificar la hoja del especialista en la que se recogen los datos de una consulta especializada general. Estos datos son los que se recogen en cualquier tipo de consulta de este tipo es decir los comunes.
- **Gestionar Informe Operatorio:** Permite crear y modificar después de concluida la intervención quirúrgica el informe operatorio.
- **Gestionar Hoja de Pediatría:** Su objetivo principal es permitir al usuario crear y modificar la hoja que contiene los datos de la consulta para menores de aprobación de cirugía en la cual se analizan ciertos parámetros para pasar a cirugía.
- **Gestionar Hoja Clínica:** Su objetivo principal es permitir al usuario crear y modificar la hoja que contiene los datos de la consulta para adultos de aprobación de cirugía en la cual se analizan ciertos parámetros para pasar a cirugía.
- **Gestionar Hoja de Consulta Anestesia:** Su objetivo principal es permitir al usuario crear y modificar la hoja de anestesia correspondiente a la consulta del anestesista.
- **Gestionar Hoja A .Acto Quirúrgico:** Este caso de uso permite que se recojan todos los datos durante el transcurso de la intervención quirúrgica.
- **Gestionar Planificación:** Permite al vicedirector quirúrgico crear, modificar o eliminar la planificación de cualquier cantidad de operaciones en un tiempo determinado por el criterio que escoja.
- **Gestionar Hoja del Especialista Oftalmológico:** Su objetivo principal es permitir al especialista crear y modificar la hoja del especialista oftalmológico, que presenta datos específicos de esta especialidad.
- **Buscar Anuncio Operatorio:** Permite visualizar los anuncios operatorios, haciendo una búsqueda por criterios específicos.
- **Buscar Consulta:** Permite visualizar las consultas, haciendo una búsqueda por criterios específicos.
- **Buscar Planificación:** Permite visualizar la planificación quirúrgica.
- **Gestionar Informe Operatorio Oftalmológico:** Permite crear y modificar el informe operatorio oftalmológico.

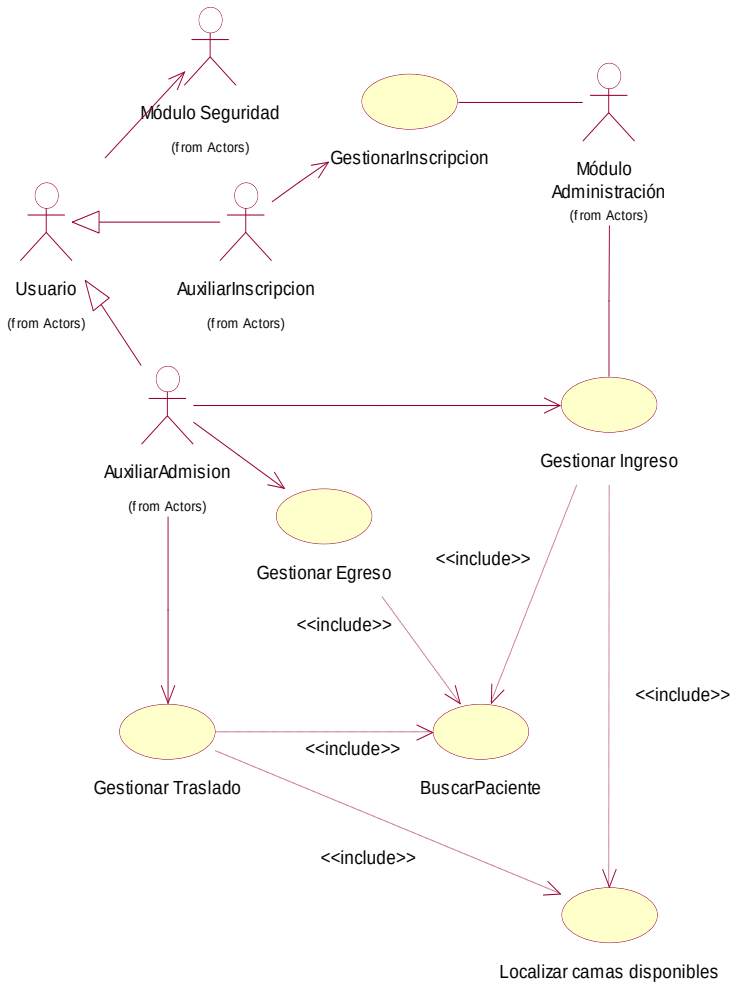
- **Gestionar Planificación Personal:** Permite visualizar la planificación quirúrgica de los especialistas.
- **Buscar Informe Operatorio:** Permite buscar el informe operatorio de un paciente.

A continuación, se exponen los diagramas de casos de uso arquitectónicamente significativos, de cada uno de los módulos enunciados anteriormente.

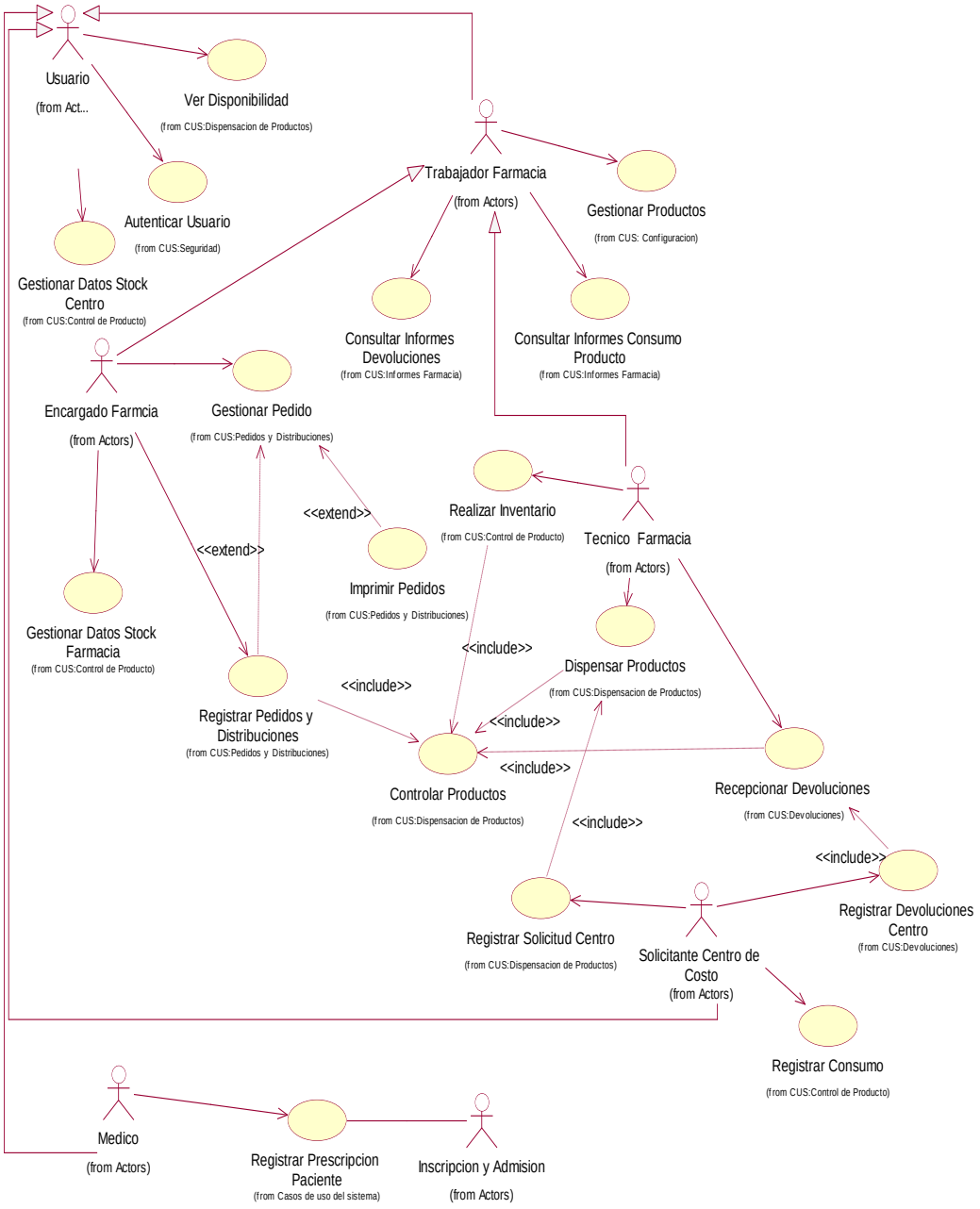
Paquete Configuración



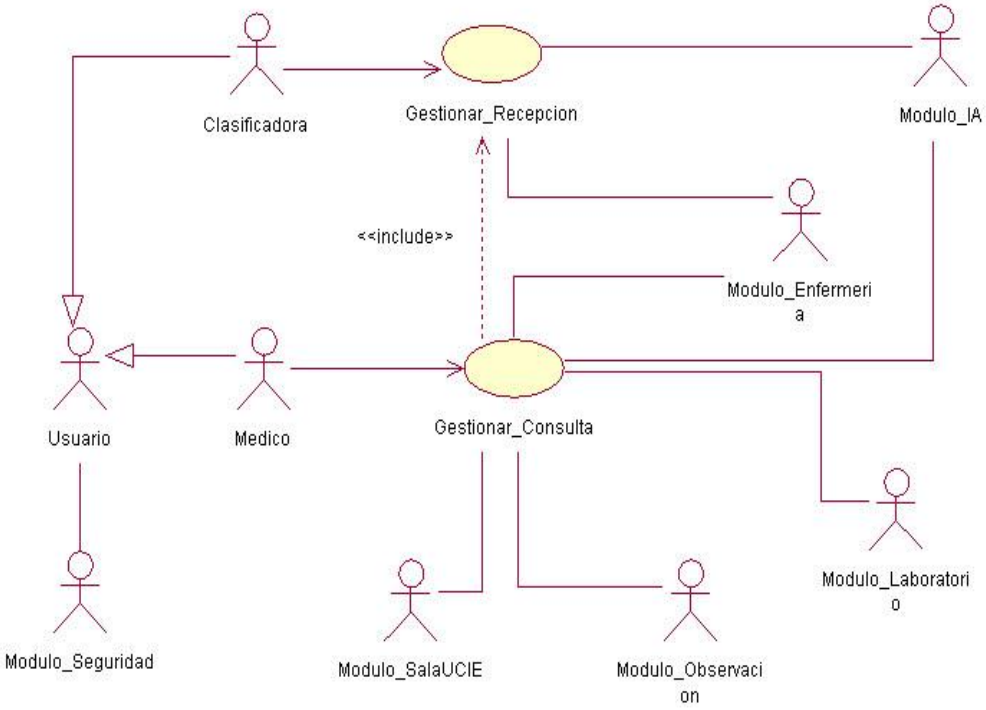
Paquete Inscripción Admisión



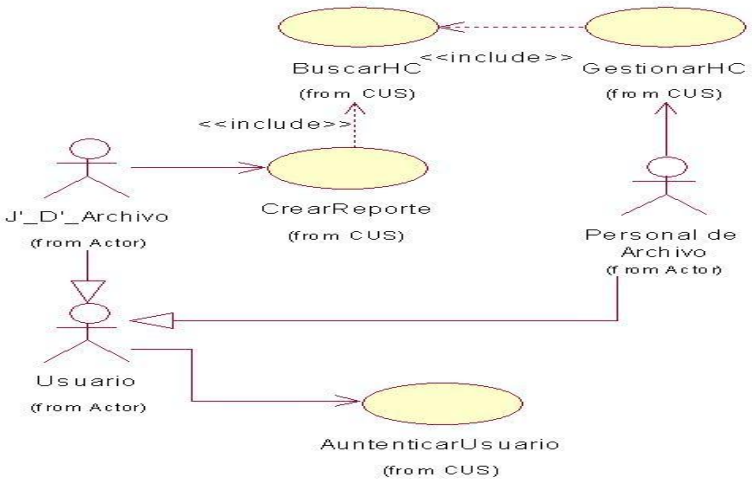
Paquete Farmacia



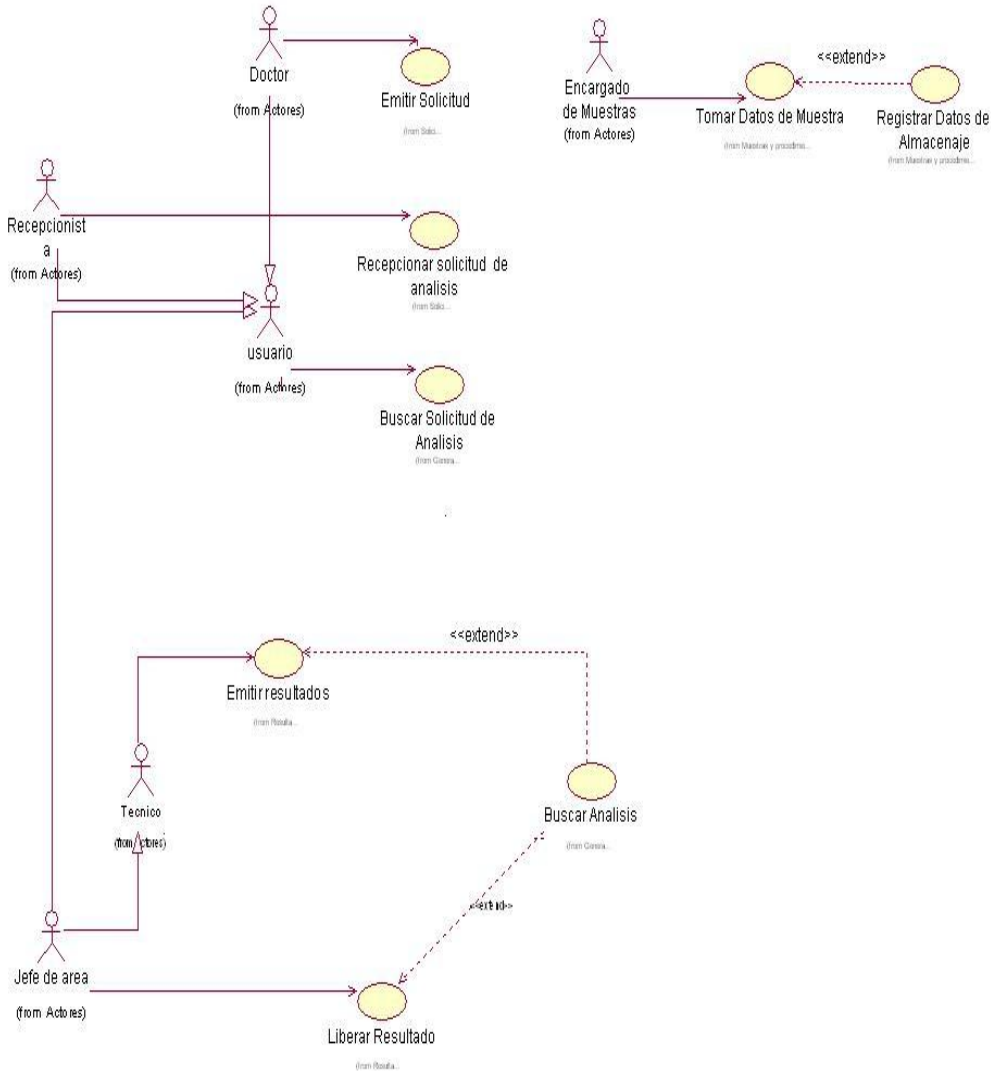
Paquete Cuerpo de Guardia



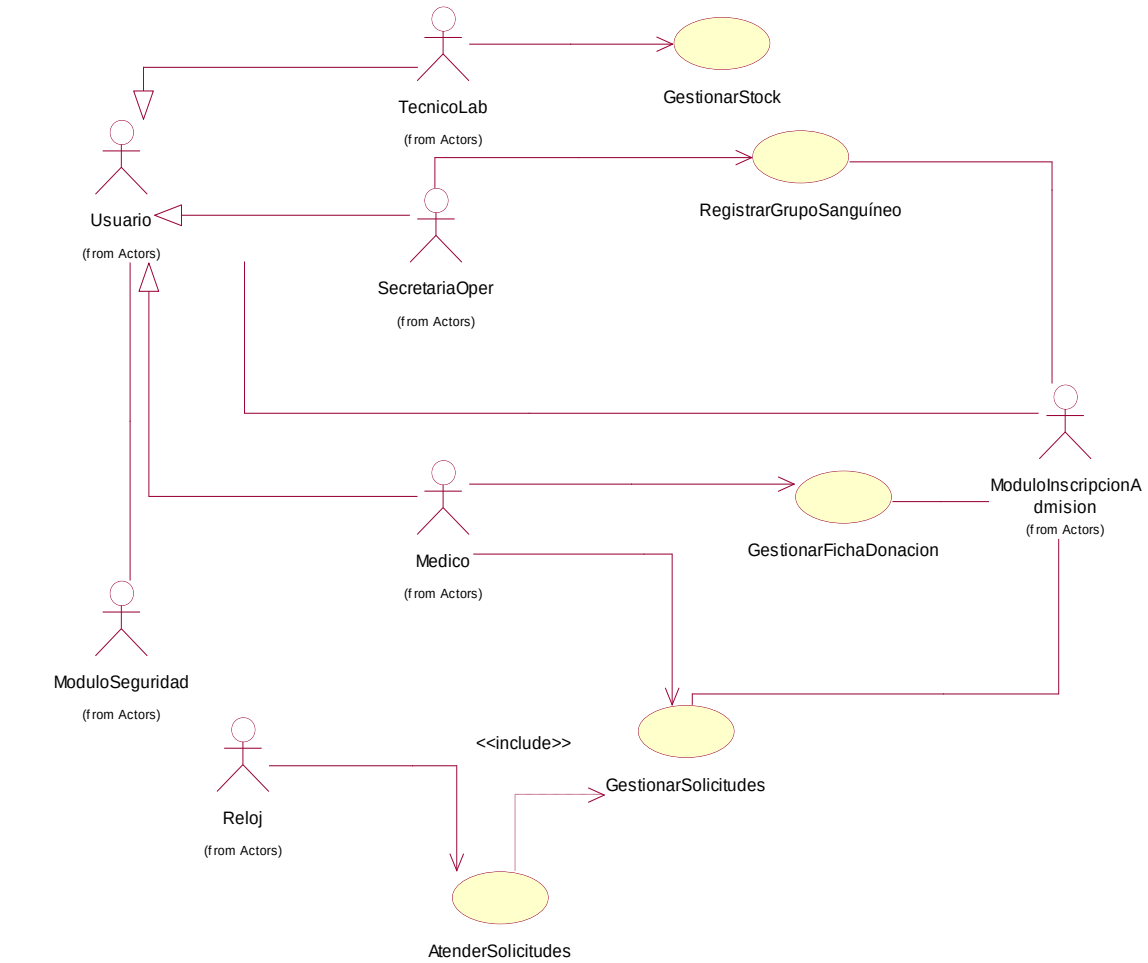
Paquete ArchivoHC



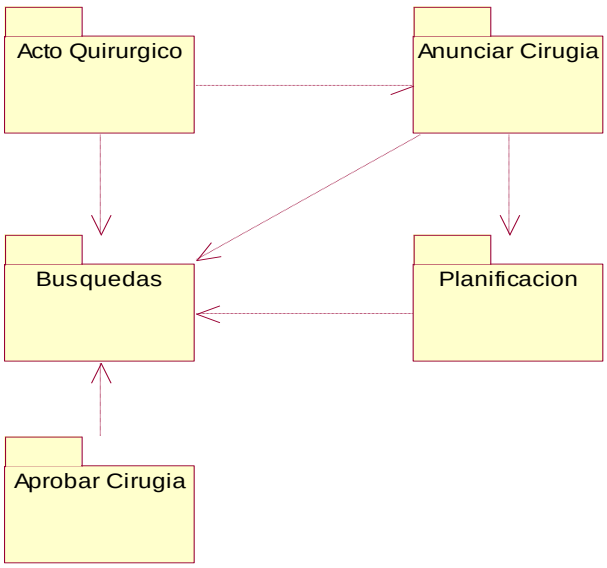
Paquete Laboratorio



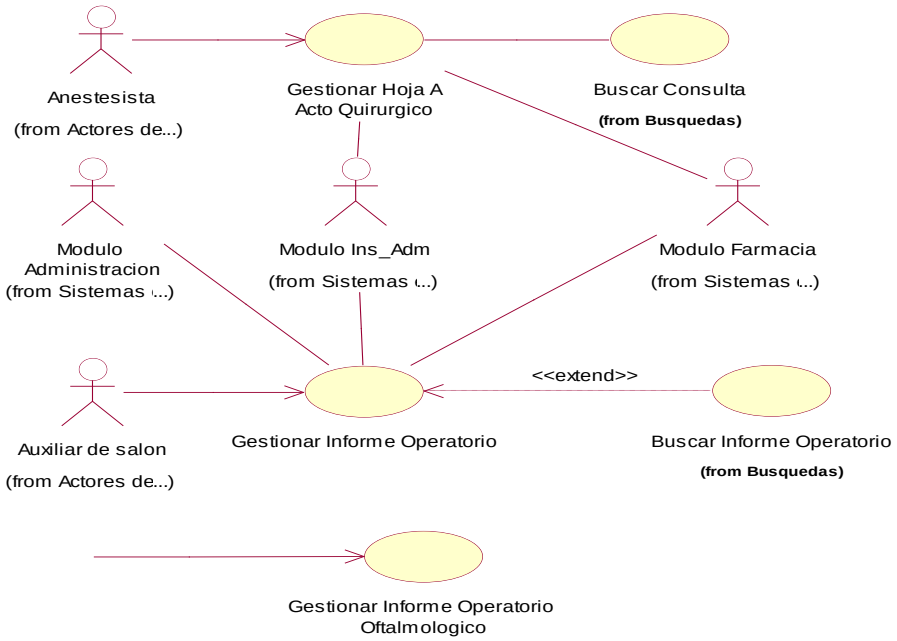
Paquete Banco de Sangre Hospital



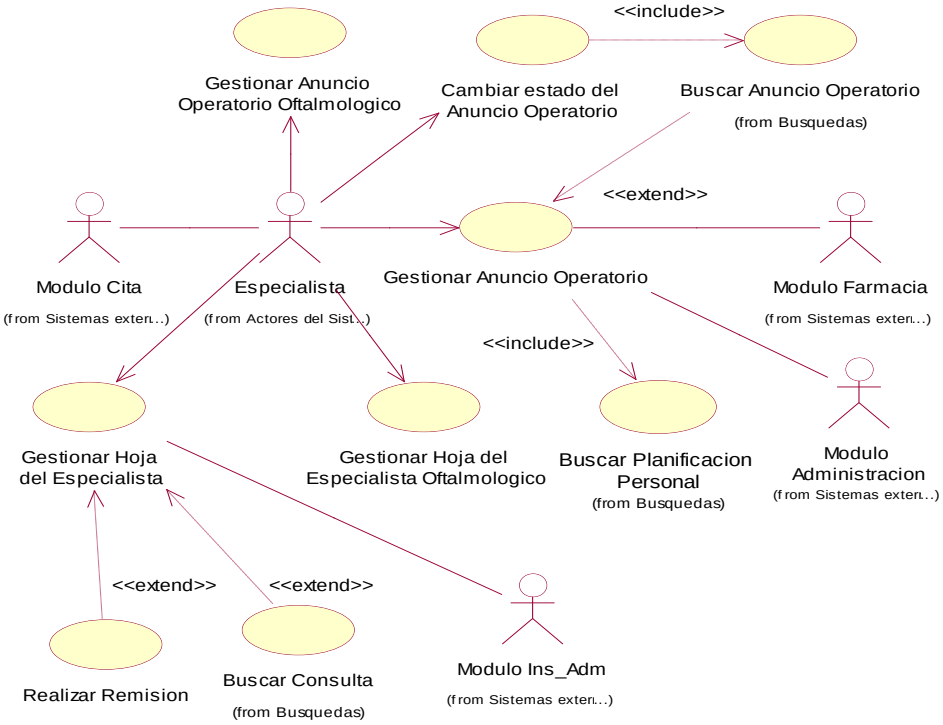
Paquete Bloque Quirúrgico



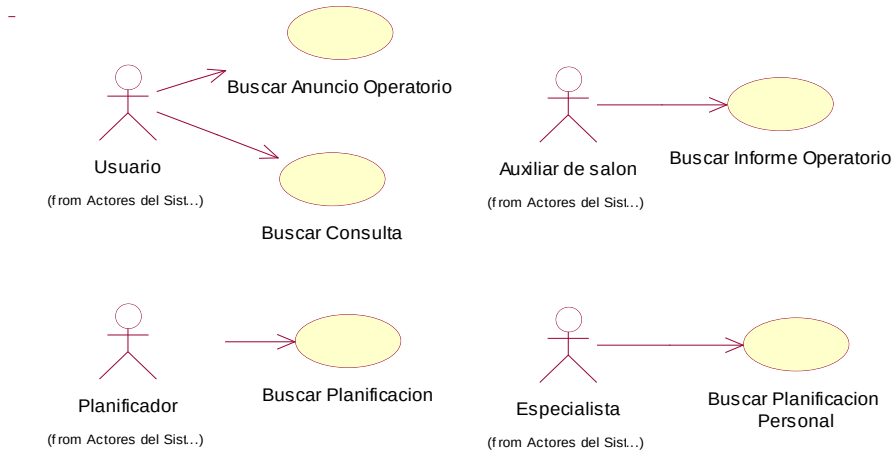
Paquete Acto Quirúrgico



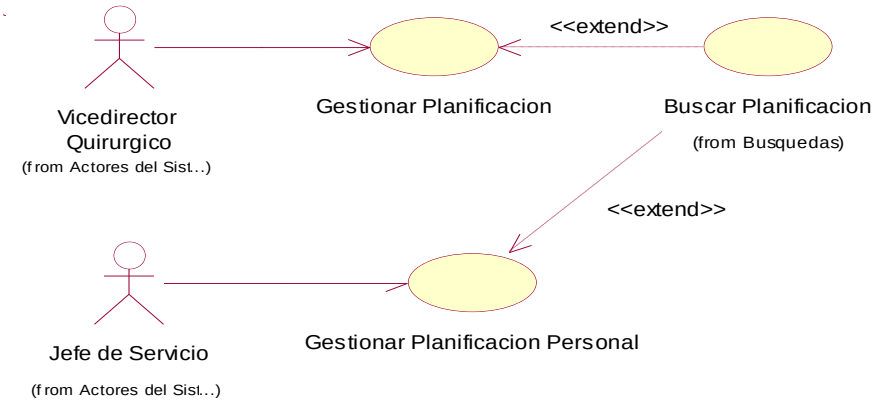
Paquete Anunciar Cirugía



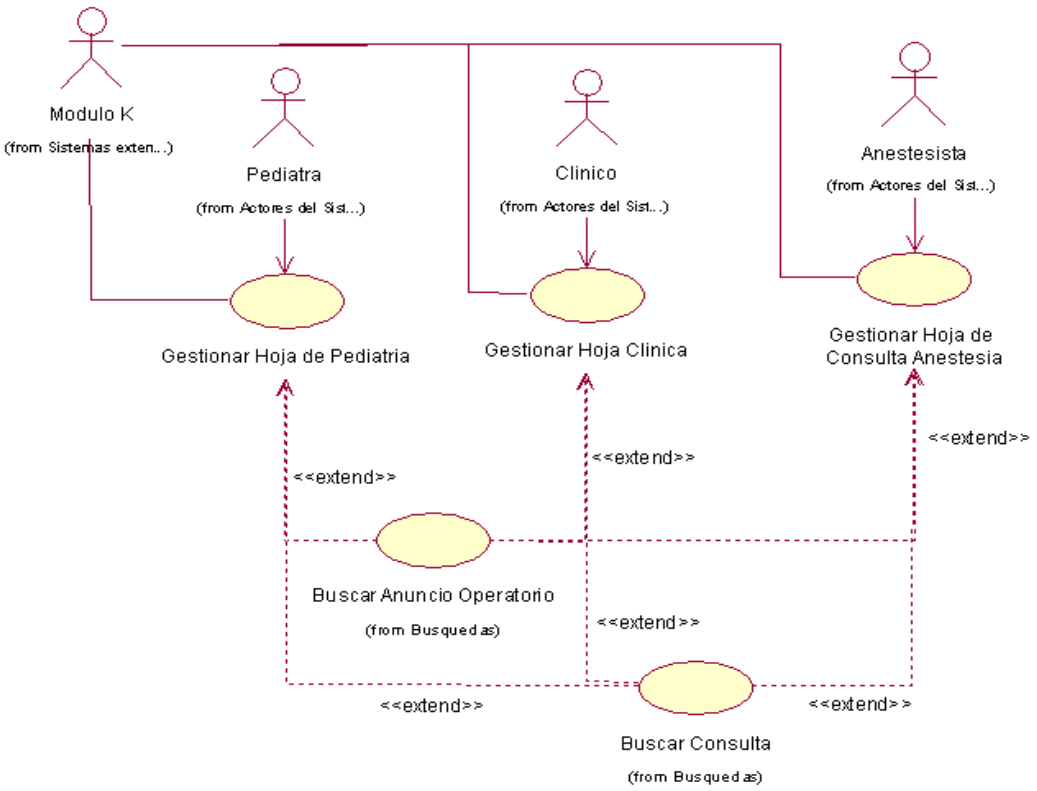
Paquete Búsquedas



Paquete Planificación



Paquete Aprobar Cirugía

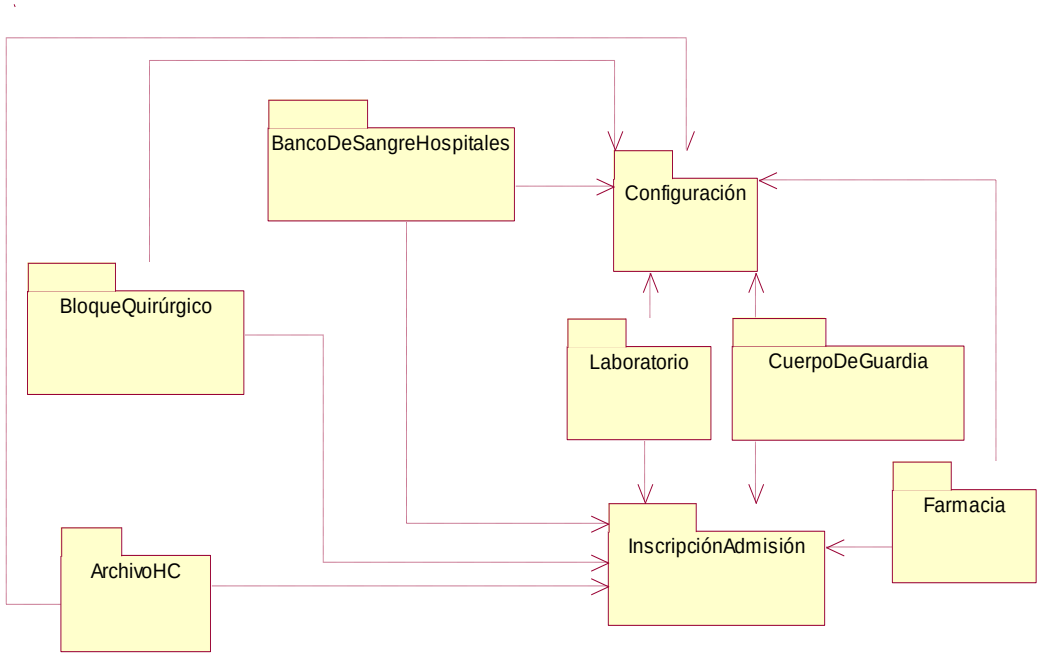


3.2.5 Vista Lógica

La vista lógica describe las clases más importantes que formarán parte del ciclo de desarrollo. Se describen los paquetes del sistema de una forma abstracta y las relaciones que existen entre ellos.

La siguiente figura muestra la división del sistema en módulos. Esto facilita el mantenimiento y la reusabilidad del sistema, pues cualquier cambio en el negocio, solo afectaría al módulo al cual pertenece la responsabilidad. Facilita y organiza el trabajo en equipo, deja abierta a consideración la opción de desarrollar en paralelo y garantizar una terminación ágil y eficaz.

Paquetes de Casos de Uso arquitectónicamente significativos del Sistema

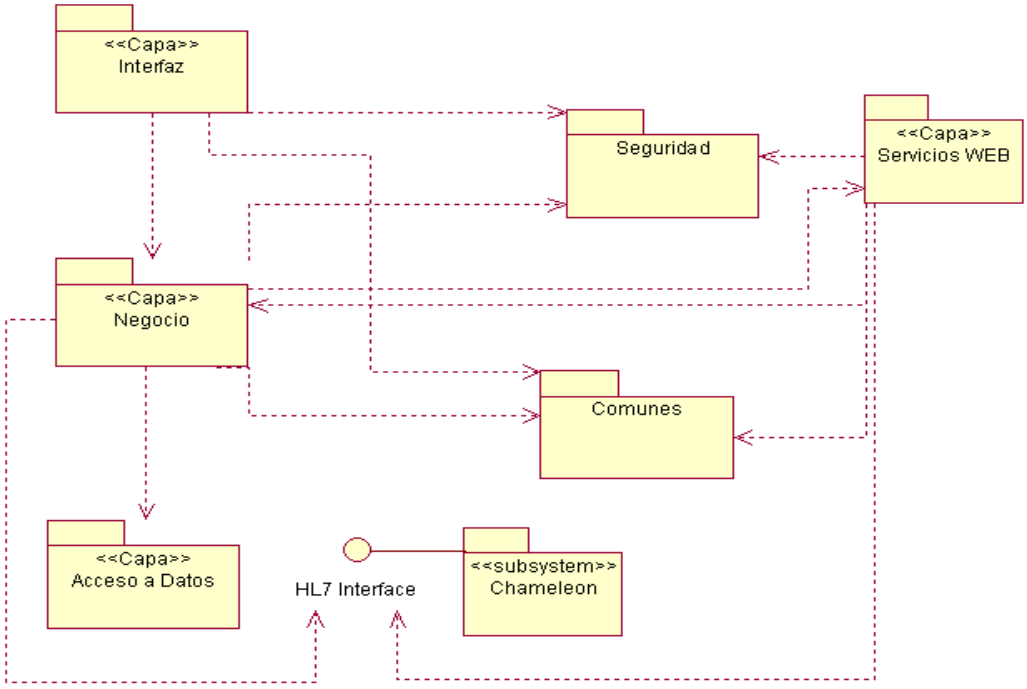


Los principales módulos identificados son:

- Configuración: Contiene la realización de casos de uso correspondientes a la gestión de roles, usuarios y nomencladores, y la autenticación de usuarios.
- Inscripción – Admisión: Contiene la realización de casos de uso encargados de la inscripción, ingresos, egresos, traslado, búsqueda de pacientes y camas disponibles dentro del sistema.

- Bloque Quirúrgico: Contiene la realización de casos de uso correspondientes a la gestión de Hoja de Anestesia para el acto quirúrgico, informes operatorios, hoja de especialistas, planificación personal y de operaciones, hojas de pediatría, hojas clínicas y de anestesia; búsquedas de anuncios e informes operatorios, consultas y planificaciones; modificar y crear anuncios operatorios.
- Banco de Sangre Hospitales: Contiene la realización de caso de uso encargados de la gestión del stock del banco, fichas de donación y solicitudes, así como el registro del grupo sanguíneo y la atención a solicitudes de sangre.
- Archivo HC: Contiene la realización de casos de uso de gestión de la Historia Clínica del paciente, la búsqueda de esta y la creación de reportes asociados a los movimientos y otros parámetros de las historias.
- Laboratorio: Contiene la realización de casos de uso correspondientes a la reservaciones y recepción de solicitudes de exámenes, almacenamiento y toma de muestras, búsquedas relacionadas con solicitudes de exámenes y Resultados de estos; emitir, modificar y liberar resultados de análisis.
- Cuerpo de Guardia: Contiene la realización de casos de usos encargados de representar la recepción y consulta de pacientes en el cuerpo de guardia del hospital.
- Farmacia: Contiene la realización de casos de uso correspondiente al control los productos de baja, los consumos, productos y el stock; además de aquellos como recepción de devoluciones, entregas de productos y elaboración de pedidos.

La distribución de la aplicación para cada uno de los módulos se hará de la siguiente forma:



Interfaz: Conjunto de clases que componen las interfaces web del sistema.

Negocio: Conjunto de clases agrupadas en dos subgrupos Negocio: clases que manejan toda la lógica del negocio, contienen un fuerte tratamiento de errores y Middleware: Conjunto de clases completamente generados con la herramienta DAG, que controlan la lógica transaccional y la interacción con la base de datos, esta capa brinda abstracción del gestor de bases de datos utilizado. Interactúa con el subsistema Chameleon con el objetivo de transformar los mensajes HL7.

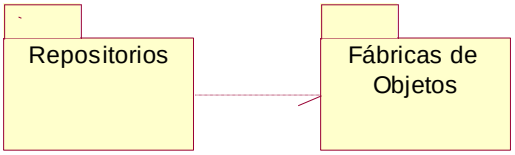
Acceso a Datos: Ubicado en el servidor de bases de datos, consistente en el conjunto de funciones que garantizan la ejecución de los pedidos a la base de datos.

Seguridad: Conjunto de clases que manejan la seguridad del sistema (usuarios, roles y permisos).

Comunes: Conjunto de clases que contiene la información sobre todas las entidades necesarias para el trabajo en el sistema.

Servicios Web: Conjunto de servicios web, que contiene la lógica del negocio del sistema y permite la interacción entre sistemas. Interactúa con el subsistema Chameleon.

La estructura de la subcapa de negocio Middleware es la siguiente:



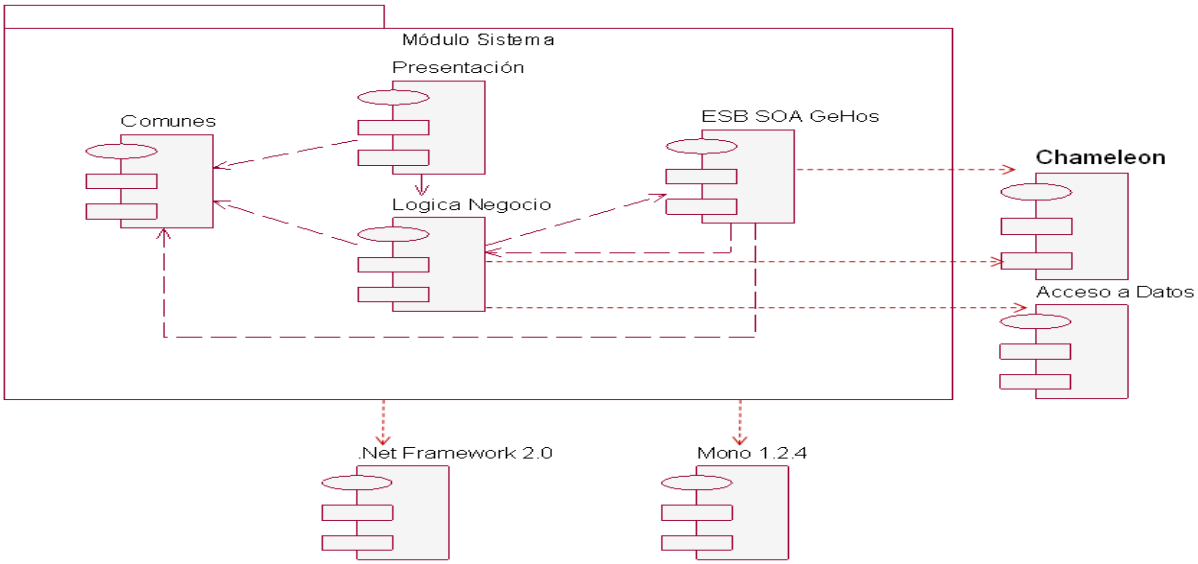
Fábricas de Objetos: Conjunto de cinco clases por cada entidad de la base de datos generada por el DAG, cada una de ellas es un tipo diferente de fábrica concreta, encargadas de realizar las operaciones de inserción, modificación, selección, eliminación u construcción de una entidad en específico en el sistema de almacenamiento que se haya designado, sea un SGBD o fichero.

Repositorios: Conjunto de clases con las cuales se comunica la capa de negocio, encargadas de manipular colecciones de objetos de un tipo de entidad en específico.

Chameleon: Subsistema encargado de brindar una interfaz HL7 con la cual interactúan los paquetes de negocio y servicios WEB para lograr el intercambio de información utilizando el estándar HL7.

3.2.6 Vista de Implementación.

La vista de implementación proporciona una descripción de las principales capas y subsistemas de componentes de la aplicación. Los paquetes definidos para cada módulo del sistema son:



Cada uno de los paquetes representados en el diagrama anterior encapsulan varios componentes que se interrelacionan para concebir el correcto funcionamiento del sistema, estableciendo además las dependencias representadas.

Los componentes se encuentran distribuidos como sigue:

- Comunes:

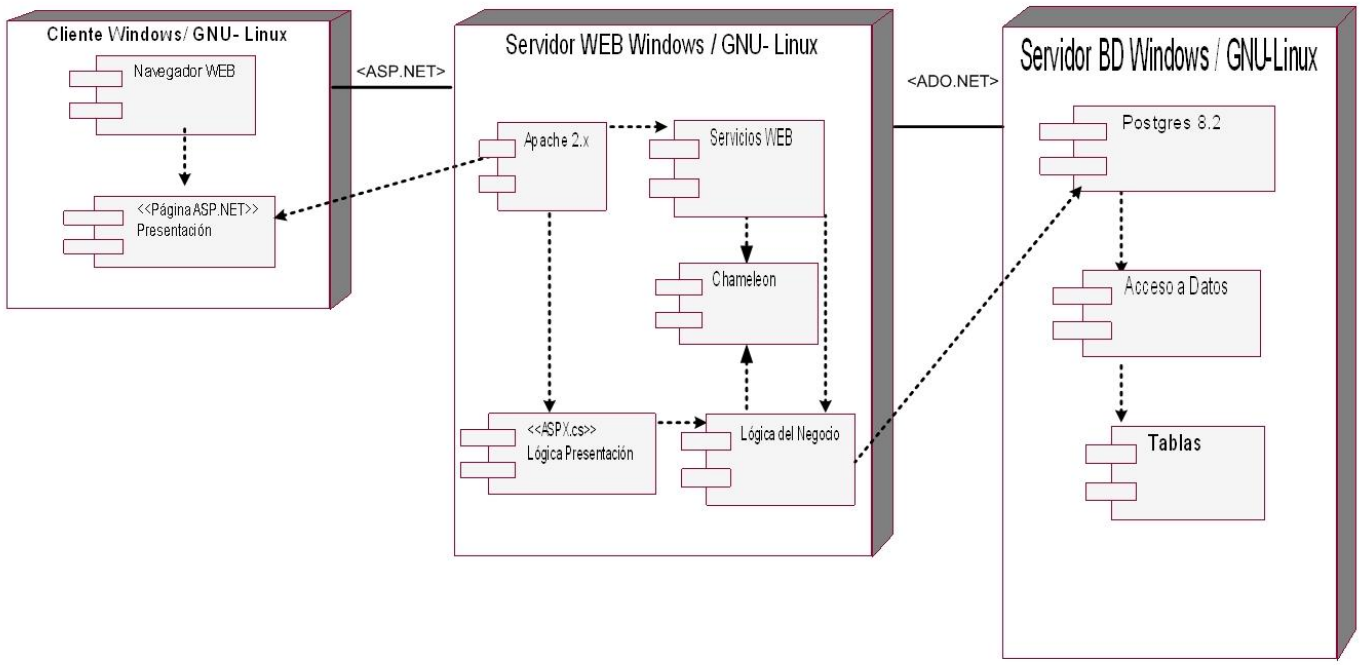
  Ensamblado de seguridad.

  Ensamblado de clases entidades, usadas en todas las capas de la aplicación.
- Presentación:

  Ensamblados y Páginas WEB ASP.NET 2.0 que engloban la lógica de presentación.
- Lógica del Negocio: Conjunto de ensamblados que representan las funcionalidades requeridas por la lógica de negocio y ejecutar las transacciones necesarias hacia y desde el soporte de almacenamiento.
- Acceso a datos: Funciones o procedimientos almacenados implementados en el servidor de bases de datos que proveen de datos y ejecutan las operaciones solicitadas desde los ensamblados que conforman la capa de negocio.
- ESB SOA GeHos: Ensamblados y conjunto de servicios WEB que mediante el uso de estándares internaciones de comunicación, permiten la integración y comunicación con sistemas externos y módulos de nuestro sistema en caso requerido.
- Chameleon: Componente que soporta y garantiza la comunicación mediante el estándar HL7.

3.2.7 Vista de Despliegue.

La vista de despliegue brinda información sobre la distribución física del sistema y cada uno de sus componentes, además de dar solución a parte de los requisitos no funcionales (hardware y software).



- Nodo Cliente: Se precisa en este nodo la presencia de un navegador WEB, preferentemente los propuestos en la línea base de la arquitectura; encargado de interactuar con el servidor WEB y visualizar las interfaces WEB al cliente.
- Nodo Servidor WEB: Contiene un servidor WEB Apache 2.0 o superior con el módulo mod_mono incluido, este será el encargado de atender las peticiones con el nodo cliente, sean estas peticiones de lógica de presentación ó servicios WEB; los que interactúan con el componente que satisface la lógica del negocio y este a su vez con el de acceso a datos para conformar así la respuesta a la petición recibida. Los servicios WEB y la lógica de negocio dependen de las funcionalidades que brinda el subsistema Chameleon ubicado en el nodo en cuestión.
- Nodo Servidor BD: Contiene el Servidor de Bases de datos relacional PostgreSQL 8.2, que atiende las peticiones del componente de acceso a datos localizado en el Nodo Servidor WEB.

En este capítulo, se presentan dos documentos de obligada consulta que conforman la descripción arquitectónica del sistema de gestión hospitalaria. Se exponen y fundamentan los patrones utilizados, así como las tecnologías y herramientas. Se describe la arquitectura mediante las vistas propuestas por la metodología RUP. Se justifica un desarrollo ágil y de calidad mediante el uso de los frameworks propuestos.

Se considera que la propuesta cumple con los requisitos que se necesitan como solución al problema planteado, es completamente desarrollable por parte de los implementadores, puede ser aprobada y administrada por los arquitectos y jefe de proyecto.

CONCLUSIONES

Se realizó el diseño de una arquitectura de software que posibilita la integración entre los sistemas de gestión de la información en los hospitales y demás componentes del sistema de salud cubano. Para ello, se definieron:

- estrategias de desarrollo,
- mejor combinación de herramientas atendiendo la razón calidad del producto/tiempo de desarrollo,
- vías comunicación e integración con componentes mediante la utilización de estándares internacionales,

Se fundamentaron patrones de diseño, arquitectura e idioma añadiéndole al sistema calidad, claridad y sencillez en la estructura y diseño de la solución, facilidades de lectura y mantenimiento de código.

Se diseño la arquitectura de seguridad, acceso a datos y negocio.

RECOMENDACIONES

Se recomienda a la facultad y en especial a los líderes y arquitectos del área temática de Gestión Hospitalaria:

- Efectuar un continuo refinamiento de la arquitectura durante cada ciclo de desarrollo.
- Realizar evaluación de costo de la tecnología y requerimientos de hardware e incentivar la búsqueda de la reducción de estos hacia las necesidades de los clientes.
- Aplicar métodos de evaluación de la arquitectura, con el objetivo de identificar posibles debilidades.
- Tener en cuenta la propuesta de arquitectura, para futuros desarrollos dentro del área temática.
- Valorar la posibilidad de hacer extensiva esta propuesta a otras áreas temáticas relacionadas con la salud.

REFERENCIAS BIBLIOGRÁFICAS

[1]. ASSETA, D. A.; ROMERO, D. D. F., *et al. SISTEMAS DE INFORMACION HOSPITALARIA (SIH), Su importancia para el desarrollo de los servicios de salud y el control de la gestión* publicado en el año 2004, última actualización: 22/03/2007. 11 p. Disponible en: http://www.medicos-municipales.org.ar/bc1104.htm#1.

[2]. Idem [1].

[3]. REYNOSO, C. B. *Introducción a la Arquitectura de Software.* publicado en el año 2004, última actualización: 03/04/2007. 42 p. Disponible en: http://www.microsoft.com/spanish/msdn/arquitectura/roadmap_arq/intro.asp.

[4]. BASS; CLEMENTS, *et al. Published Software Architecture Definitions, Modern Definitions.* publicado en el año 2003, última actualización: 16/03/2007. Disponible en: http://www.sei.cmu.edu/architecture/published_definitions.html#Modern.

[5]. Idem [4].

[6]. Idem [3].

[7]. REYNOSO, C. y KICILLOF, N. *De Lenguajes de descripción arquitectónica de Sofware (ADL).* publicado en el año 2004, última actualización: 03/04/2007. 58 p. Disponible en: http://www.willydev.net/descargas/prev/ADL.pdf.

[8]. Idem [3].

[9]. Idem [7].

[10]. WESLEY, A.; RUMBAUGH, E. J., *et al. Lenguaje Unificado de Modelamiento.* publicado en el año 2000, última actualización: 06/04/2007. Disponible en: http://www.creangel.com/uml/intro.php.

[11]. Idem [10].

[12]. WEBOPEDIA. *Web services.* publicado en el año 2006, última actualización: 15/03/2007. 1 p. Disponible en: http://www.webopedia.com/TERM/W/Web_services.html.

[13]. MADRID, U. C. D. *Concepto y características de los SGBD* publicado en el año 1997, última actualización: 29/03/2007. Disponible en: http://www.eubd.ucm.es/html/personales/enred/mantonia/docauto/tema5/tema5.htm.

[14]. Idem [13].

[15]. Idem [13]

[16]. INSTITUTO POLITÉCNICO NACIONAL NO. 2508 COL. SAN PEDRO ZACATENCO, M. *Sistemas de Bases de Datos Distribuidas.* publicado en el año 2001, última actualización: 28/03/2007. 1 p. Disponible en: http://www.cs.cinvestav.mx/SC/prof_personal/adiaz/Disdb/temario.html.

[17]. SANCHEZ, M. A. M. *Metodologías De Desarrollo De Software.* publicado en el año 2002, última actualización: 25/02/2007. 5 p. Disponible en: http://www.informatizate.net/articulos/pdfs/metodologias_de_desarrollo_de_software_07062004.pdf.

[18]. Idem [17].

[19]. REYNOSO, C. y KICILLOF, N. *Estilos y Patrones en la Estrategia de Arquitectura de Microsoft.* publicado en el año 2004, última actualización: 05/04/2007. Disponible en: http://www.microsoft.com/spanish/msdn/arquitectura/roadmap_arq/style.asp#24.

[20]. CERRITOS, A.; PUERTO, F. J. F.*, et al. Sistema de Información Hospitalaria.* publicado en el año 2003, última actualización: 25/03/2007. 24 p. Disponible en: www.facmed.unam.mx/emc/computo/ssa/HIS/his.pdf.

[21]. LATORILLA, E. CARE2X - El Proyecto. publicado en el año 2004,, ultima actualización: 08/04/2007, Disponible en: http://www.care2x.org/index.php?c2x_lang=es&chglang=1.

[22]. Idem [19].

[23]. Idem [19].

[24]. Idem [19].

[25]. Idem [19].

[26]. Idem [19].

[27]. Idem [19].

[28]. Idem [19].

[29]. SPAIN, H. L. *¿Qué es HL7?* publicado en el año 2007, última actualización: 07/04/2007. Disponible en: http://www.hl7spain.org/VerPagina.asp?IDPage=0.

[30]. CMS-SPAIN. *Software AG promueve el uso de XML/HL7 para crear una Historia Clínica Única del paciente para todas las CC.AA.* publicado en el año 2005, última actualización: 05/04/2007. Disponible en: http://www.ecm-spain.com/noticia.asp?IdItem=5715.

[31]. WIKIMEDIAPROJECT. *XSLT.* publicado en el año 2007, última actualización: 06/05/2007. Disponible en: http://es.wikipedia.org/wiki/XSLT.

[32]. MERELO, J. J. *Transformando documentos XML usando XSLT.* publicado en el año 2001, última actualización: 25/03/2007. Disponible en: http://geneura.ugr.es/~jmerelo/XSLT/XSLT-2001-1ed.htm.

[33]. CORPORATION, M. *Usar WSDL.* publicado en el año 2007, última actualización: 05/04/2007. Disponible en: http://msdn2.microsoft.com/es-es/library/ms175476(SQL.90).aspx.

[34]. GONZÁLEZ, C. B. *WSDL para la documentación de Servicios Web.* publicado en el año 2007, última actualización: 29/03/2007. Disponible en: http://www.desarrolloweb.com/articulos/1581.php.

[35]. GONZÁLEZ, C. B. *UDDI (Universal Description Discovery and Integration).* publicado en el año 2007, última actualización: 29/03/2007. Disponible en: http://www.desarrolloweb.com/articulos/1589.php.

[36]. GARRETT, J. J. *Ajax: Un nuevo acercamiento a las aplicaciones web.* publicado en el año 2005, última actualización: 10/04/2007. Disponible en: http://www.maestrosdelweb.com/editorial/ajax/.

[37]. MURIEL, I. M. *Lector de Contadores de Agua en Pocket PC bajo .NET Compact Framework.* [I+D]. MÁLAGA: UNIVERSIDAD DE MÁLAGA, 2005, [Consultado el: 30/03/2007]. 113 p. Disponible en: http://www.lcc.uma.es/pfc/385.pdf.

[38]. ADR INFOR S.L., L. *ASP.NET.* publicado en el año 2005, última actualización: 05/04/2007. Disponible en: http://www.adrformacion.com/cursos/aspnet2/leccion3/tutorial1.html.

[39]. CORPORATION, M. *Introducción a Visual Studio.* publicado en el año 2007, última actualización: 08/04/2007. Disponible en: http://msdn2.microsoft.com/es-es/library/fx6bk1f4(VS.80).aspx.

[40]. COLLINS-SUSSMAN, B.; FITZPATRICK, B. W.*, et al. Control de versiones con Subversion.* publicado en el año 2004, última actualización: 06/04/2005. Disponible en: http://svnbook.red-bean.com/nightly/es/svn-book.pdf.

[41]. KÜNG, S.; ONKEN, L.*, et al. TortoiseSVN: Un cliente de Subversion para Windows: Version 1.4.3.* publicado en el año 2006, última actualización: 08/04/2007. Disponible en: http://ufpr.dl.sourceforge.net/sourceforge/tortoisesvn/TortoiseSVN-1.4.3-es.pdf.

[42]. LARMAN, C. *UML y patrones.* Editado por: Varela, E. F. La Habana: 2004. vol. II, 460 p.

[43]. Idem [35].

[44]. LARMAN, C. *UML y patrones.* Editado por: Varela, E. F. La Habana: 2004. vol. I, 193-196 p.

[45]. Idem [37].

[46]. Idem [37].

[47]. LARMAN, C. *UML y patrones.* Editado por: Varela, E. F. La Habana: 2004. vol. I, 197-199 p.

[48]. Idem [40].

[49]. LARMAN, C. *UML y patrones.* Editado por: Varela, E. F. La Habana: 2004. vol. I, 203-205 p.

[50]. LARMAN, C. *UML y patrones.* Editado por: Varela, E. F. La Habana: 2004. vol. I, 200-202 p.

BIBLIOGRAFÍA

1. ADR INFOR S.L., L. ASP.NET, publicado en el año 2005, última actualización: 05/04/2007. Disponible en: http://www.adrformacion.com/cursos/aspnet2/leccion3/tutorial1.html.
2. ASSETA, D. A.; ROMERO, D. D. F., et al. SISTEMAS DE INFORMACION HOSPITALARIA (SIH), Su importancia para el desarrollo de los servicios de salud y el control de la gestión publicado en el año 2004, última actualización: 22/03/2007. 11 p. Disponible en: http://www.medicos-municipales.org.ar/bc1104.htm#1.
3. BASS; CLEMENTS, et al. Published Software Architecture Definitions, Modern Definitions, publicado en el año 2003, última actualización: 16/03/2007. Disponible en: http://www.sei.cmu.edu/architecture/published_definitions.html#Modern.
4. CARTER, J. Community Software Architecture Definitions, publicado en el año 2007, última actualización: 23/03/2007. Disponible en: http://www.sei.cmu.edu/architecture/community_definitions.html.
5. CERRITOS, A.; PUERTO, F. J. F., et al. Sistema de Información Hospitalaria, publicado en el año 2003, última actualización: 25/03/2007. 24 p. Disponible en: www.facmed.unam.mx/emc/computo/ssa/HIS/his.pdf.
6. CMS-SPAIN. Software AG promueve el uso de XML/HL7 para crear una Historia Clínica Única del paciente para todas las CC.AA, publicado en el año 2005, última actualización: 05/04/2007. Disponible en: http://www.ecm-spain.com/noticia.asp?IdItem=5715.
7. COLLINS-SUSSMAN, B.; FITZPATRICK, B. W., et al. Control de versiones con Subversion, publicado en el año 2004, última actualización: 06/04/2005. Disponible en: http://svnbook.red-bean.com/nightly/es/svn-book.pdf.
8. CORPORATION, M. Introducción a Visual Studio, publicado en el año 2007, última actualización: 08/04/2007. Disponible en: http://msdn2.microsoft.com/es-es/library/fx6bk1f4(VS.80).aspx.
9. CORPORATION, M. Usar WSDL, publicado en el año 2007, última actualización: 05/04/2007. Disponible en: http://msdn2.microsoft.com/es-es/library/ms175476(SQL.90).aspx.
10. INSTITUTO POLITÉCNICO NACIONAL NO. 2508 COL. SAN PEDRO ZACATENCO, M. Sistemas de Bases de Datos Distribuidas, publicado en el año 2001, última actualización: 28/03/2007. 1 p. Disponible en: http://www.cs.cinvestav.mx/SC/prof_personal/adiaz/Disdb/temario.html.
11. JOSE ALBERTO MALDONADO, M. R., PERE CRESPO. Utilización de arquetipos para la integración de sistemas de información hospitalarios, publicado en el año 2005 última

actualización: 24/03/2007. 4 p. Disponible en: http://gim.upv.es/sih/articulos/caseib2002.pdf.

12. KÜNG, S.; ONKEN, L., et al. TortoiseSVN: Un cliente de Subversion para Windows: Version 1.4.3, publicado en el año 2006, última actualización: 08/04/2007. Disponible en: http://ufpr.dl.sourceforge.net/sourceforge/tortoisesvn/TortoiseSVN-1.4.3-es.pdf.
13. LANE; RECHTIN, et al. Published Software Architecture Definitions, Bibliographic Definitions, publicado en el año 2003, última actualización: 22/03/2007. Disponible en: http://www.sei.cmu.edu/architecture/published_definitions.html#Bibliographic.
14. LARMAN, C. UML y patrones. Editado por: Varela, E. F. La Habana: 2004. vol. I y II,
15. LATORILLA, E. CARE2X - El Proyecto, publicado en el año 2004, Disponible en: http://www.care2x.org/index.php?c2x_lang=es&chglang=1.
16. MADRID, U. C. D. Concepto y características de los SGBD publicado en el año 1997, última actualización: 29/03/2007. Disponible en: http://www.eubd.ucm.es/html/personales/enred/mantonia/docauto/tema5/tema5.htm.
17. MASADELANTE.COM. ¿Qué es un servidor? - Definición de servidor, publicado en el año 2007, última actualización: 05/03/2007. 1 p. Disponible en: http://www.masadelante.com/faq-servidor.htm.
18. MELLON, U. C. How Do You Define Software Architecture? publicado en el año 2007, última actualización: 22/03/2007. 1 p. Disponible en: http://www.sei.cmu.edu/architecture/definitions.html.
19. MERELO, J. J. Transformando documentos XML usando XSLT, publicado en el año 2001, última actualización: 25/03/2007. Disponible en: http://geneura.ugr.es/~jmerelo/XSLT/XSLT-2001-1ed.htm.
20. MURIEL, I. M. Lector de Contadores de Agua en Pocket PC bajo .NET Compact Framework. [I+D]. MÁLAGA: UNIVERSIDAD DE MÁLAGA, 2005, [Consultado el: 30/03/2007]. 113 p. Disponible en: http://www.lcc.uma.es/pfc/385.pdf.
21. PRESSMAN, R. Ingeniería del Software: Un enfoque práctico. Madrid: McGraw Hill Prentice-Hall, 2001.
22. PROCESS, R. U. Published Software Architecture Definitions, Classic Definitions, publicado en el año 1999, última actualización: 23/03/2007. Disponible en: http://www.sei.cmu.edu/architecture/published_definitions.html#Classic.
23. PROJECT, W. Historia de la Informática en Cuba, publicado en el año 2007, última actualización: 15/03/2007. 4 p. Disponible en: http://es.wikipedia.org/wiki/Historia_de_la_Inform%C3%A1tica_en_Cuba.

24. REYNOSO, C. y KICILLOF, N. De Lenguajes de descripción arquitectónica de Sofware (ADL), publicado en el año 2004, última actualización: 03/04/2007. 58 p. Disponible en: http://www.willydev.net/descargas/prev/ADL.pdf.
25. REYNOSO, C. y KICILLOF, N . Estilos y Patrones en la Estrategia de Arquitectura de Microsoft, publicado en el año 2004, última actualización: 05/04/2007. Disponible en: http://www.microsoft.com/spanish/msdn/arquitectura/roadmap_arq/style.asp#24.
26. REYNOSO, C. B. Introducción a la Arquitectura de Software, publicado en el año 2004, última actualización: 03/04/2007. 42 p. Disponible en: http://www.microsoft.com/spanish/msdn/arquitectura/roadmap_arq/intro.asp.
27. SANCHEZ, M. A. M. Metodologías De Desarrollo De Software, publicado en el año 2002, última actualización: 25/02/2007. 5 p. Disponible en: http://www.informatizate.net/articulos/pdfs/metodologias_de_desarrollo_de_software_07062004.pdf
28. SPAIN, H. L. ¿Qué es HL7? publicado en el año 2007, última actualización: 07/04/2007.Disponible en: http://www.hl7spain.org/VerPagina.asp?IDPage=0.
29. W3C. Guía Breve de Servicios Web, publicado en el año 2005, última actualización: 15/03/2007. 5 p. Disponible en: http://www.w3c.es/Divulgacion/Guiasbreves/ServiciosWeb.
30. WEBOPEDIA. Browser, publicado en el año 2004, última actualización: 22/03/2007. Disponible en: http://www.webopedia.com/TERM/b/browser.html.
31. WEBOPEDIA.. Web services, publicado en el año 2006, última actualización: 15/03/2007. 1 p. Disponible en: http://www.webopedia.com/TERM/W/Web_services.html.
32. WESLEY, A.; RUMBAUGH, E. J., et al. Lenguaje Unificado de Modelamiento, publicado en el año 2000, última actualización: 06/04/2007. Disponible en: http://www.creangel.com/uml/intro.php.
33. WIKIMEDIAPROJECT. XSLT, publicado en el año 2007, última actualización: 06/05/2007. Disponible en: http://es.wikipedia.org/wiki/XSLT.

ANEXOS

Anexo No. 1 Propiedades de aplicaciones que utilizan Arquitectura Orientada a Servicios (SOA).

	Programación Estructurada	Objetos	Componentes	Servicios
Granularidad	Muy fina	Fina	Intermedia	Gruesa
Contrato	Definido	Privado/Público	Público	Publicado
Reusabilidad	Baja	Baja	Intermedia	Alta
Acoplamiento	Fuerte	Fuerte	Débil	Muy débil
Dependencias	Tiempo de Compilación	Tiempo de Compilación	Tiempo de Compilación	Ejecución
Ámbito de Comunicación	Intra-Aplicación	Intra-Aplicación	Intra-Aplicación	Inter-Empresas

Anexo No. 2 Estadísticas del uso de Navegadores desde Enero del 2002 hasta febrero del 2007.

2007	**IE7**	**IE6**	**IE5**	**Fx**	**Moz**	**S**	**O**
February	16.4%	39.8%	2.5%	31.2%	1.4%	1.7%	1.5%
January	13.3%	42.3%	3.0%	31.0%	1.5%	1.7%	1.5%
2006	**IE7**	**IE6**	**IE5**	**Fx**	**Moz**	**N7/8**	**O**
December	10.7%	45.3%	3.4%	30.3%	2.6%	0.2%	1.5%
November	7.1%	49.9%	3.6%	29.9%	2.5%	0.2%	1.5%
October	3.1%	54.5%	3.8%	28.8%	2.4%	0.3%	1.4%
September	2.5%	55.6%	4.0%	27.3%	2.3%	0.4%	1.6%
August	2.0%	56.2%	4.1%	27.1%	2.3%	0.3%	1.6%
July	1.9%	56.3%	4.2%	25.5%	2.3%	0.4%	1.4%

June	1.6%	58.2%	4.3%	24.9%	2.2%	0.3%	1.4%
May	1.1%	57.4%	4.5%	25.7%	2.3%	0.3%	1.5%
April	0.7%	58.0%	5.0%	25.2%	2.5%	0.4%	1.5%
March	0.6%	58.8%	5.3%	24.5%	2.4%	0.5%	1.5%
February	0.5%	59.5%	5.7%	25.1%	2.9%	0.4%	1.5%
January	0.2%	60.3%	5.5%	25.0%	3.1%	0.5%	1.6%
2005	**IE6**	**IE5**	**Fx**	**Moz**	**N7**	**O8**	**O7**
December	61.5%	6.5%	24.0%	2.7%	0.4%	1.3%	0.2%
November	62.7%	6.2%	23.6%	2.8%	0.4%	1.3%	0.2%
October	67.5%	6.0%	19.6%	2.6%	0.4%	1.2%	0.2%
September	69.8%	5.7%	18.0%	2.5%	0.4%	1.0%	0.2%
August	68.4%	6.3%	18.9%	2.4%	0.4%	0.8%	0.3%
July	67.9%	5.9%	19.8%	2.6%	0.5%	0.8%	0.4%
June	65.0%	6.8%	20.7%	2.9%	0.6%	0.7%	0.5%
May	64.8%	6.8%	21.0%	3.1%	0.7%	0.7%	0.6%
April	63.5%	7.9%	20.9%	3.1%	0.9%	0.4%	1.0%
March	63.6%	8.9%	18.9%	3.3%	1.0%	0.3%	1.6%
February	63.9%	9.5%	17.9%	3.3%	1.0%		1.7%
January	64.8%	9.7%	16.6%	3.4%	1.1%		1.9%
2004	**IE6**	**IE5**	**Moz**	**N3**	**N7**	**N4**	**O7**
December	65.5%	9.9%	17.0%	0.2%	1.2%	0.2%	1.8%
November	66.0%	10.2%	16.5%	0.2%	1.2%	0.3%	1.6%
October	67.3%	10.8%	14.7%	0.3%	1.3%	0.3%	1.6%
September	67.8%	11.2%	13.7%	0.3%	1.4%	0.3%	1.7%

August	67.0%	13.0%	12.7%	0.4%	1.4%	0.4%	1.6%
July	67.2%	13.2%	12.6%	0.4%	1.4%	0.4%	1.6%
June	67.6%	13.2%	12.2%	0.5%	1.4%	0.4%	1.6%
May	68.1%	13.8%	9.5%	0.6%	1.4%	0.4%	1.6%
April	68.2%	14.0%	8.5%	0.8%	1.4%	0.6%	1.4%
March	68.2%	14.6%	7.9%	0.8%	1.4%	0.6%	1.4%
February	68.3%	15.2%	7.3%	0.6%	1.5%	0.4%	1.5%
January	68.9%	15.8%	5.5%	0.4%	1.5%	0.5%	1.5%
2003	**IE6**	**IE5**	**Moz**	**N3**	**N7**	**N4**	**O7**
November	71.2%	13.7%	7.2%	0.5%	1.6%	0.5%	1.9%
September	69.7%	16.9%	6.2%	0.6%	1.5%	0.6%	1.8%
July	66.9%	20.3%	5.7%	0.6%	1.5%	0.6%	1.7%
May	65.0%	22.7%	4.6%	1.0%	1.4%	0.9%	1.4%
March	63.4%	24.6%	4.2%	0.9%	1.4%	1.1%	1.2%
January	55.3%	29.3%	4.0%	1.2%	1.1%	1.7%	
2002	**IE6**	**IE5**	**AOL**	**N3**	**N5**	**N4**	**IE4**
November	53.5%	29.9%	5.2%	1.1%	4.9%	2.0%	
September	49.1%	34.4%	4.5%	1.3%	4.5%	2.2%	
July	44.4%	40.1%	3.5%	1.2%	3.5%	2.6%	0.5%
May	40.7%	46.0%	2.8%	1.2%	2.7%	3.4%	0.7%
March	36.7%	49.4%	3.0%	1.2%	2.4%	4.1%	0.7%
January	30.1%	55.7%	2.8%	1.3%	2.2%	4.4%	1.0%
Leyenda							
IE	Internet Explorer						

Fx	Firefox (identified as Mozilla before 2005)
Moz	The Mozilla Suite (Safari, Konqueror, Gecko, Netscape)
S	Safari (identified as Mozilla before 2007)
O	Opera
N	Netscape (identified as Mozilla after 2006)
AOL	America Online (based on both Internet Explorer and Mozilla)

Anexo No. 3 Reglas de uso de mayúsculas y ejemplos de los diferentes tipos de identificadores.

Identificador	Uso de mayúsculas o minúsculas	Ejemplo
Class	Pascal	**AppDomain**
Tipo Enum	Pascal	**ErrorLevel**
Valores enum	Pascal	**FatalError**
Evento	Pascal	**ValueChange**
Clase de excepciones	Pascal	**WebException** **Nota** Termina siempre con el sufijo **Exception**.
Campo estático de sólo lectura	Pascal	**RedValue**
Interfaz	Pascal	**IDisposable** **Nota** Comienza siempre con el prefijo I.
Método	Pascal	**ToString**
Espacio de nombres	Pascal	**System.Drawing**

Parámetro	Camel	**typeName**
Propiedad	Pascal	**BackColor**
Campo de instancia protegido	Camel	**redValue** **Nota** Se utiliza en contadas ocasiones. Es preferible utilizar una propiedad, en vez de un campo de instancia protegido.
Campo de instancia público	Pascal	**RedValue** **Nota** Se utiliza en contadas ocasiones. Es preferible utilizar una propiedad, en vez de un campo de instancia público.

Anexo No. 4 Palabras claves del lenguaje.

Addhandler	AddressOf	Alias	And	Ansi
As	Assembly	Auto	Base	Boolean
ByRef	Byte	ByVal	Call	Uso de mayúsculas o minúsculas
Catch	CBool	CByte	CChar	CDate
CDec	CDbl	Char	CInt	Class
CLng	CObj	Const	CShort	CSng
CStr	CType	Date	Decimal	Declare
Default	Delegate	Dim	Do	Double
Each	Else	ElseIf	End	Enumeración
Erase	Error	Evento	Exit	ExternalSource
False	Finalize	Finally	Float	For
Friend	Función	Get	GetType	GoTo
Handles	If	Implements	Imports	In

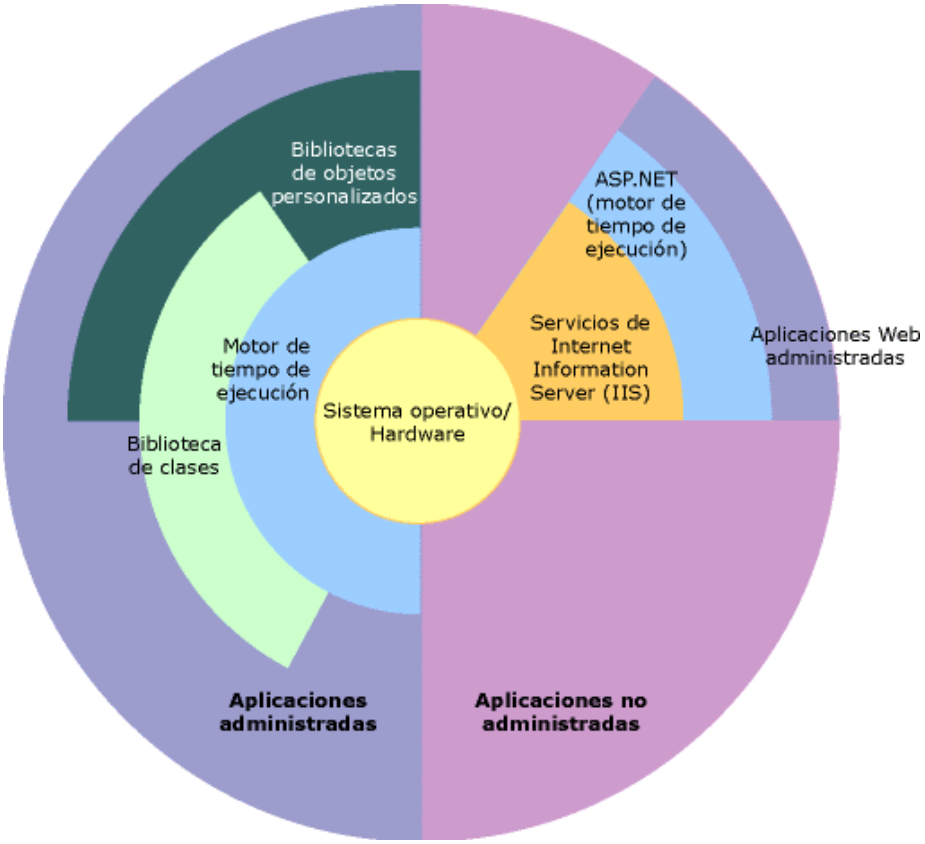
Inherits	Integer	Interfaz	Is	Let
Lib	Like	Long	Loop	Me
Mod	Module	MustInherit	MustOverride	MyBase
MyClass	Namespace	Nueva	Next	Not
Nothing	NotInheritable	NotOverridable	Object	On
Opción	Opcional	Or	Overloads	Overridable
Overrides	ParamArray	Preserve	Private	Property
Protected	Public	RaiseEvent	ReadOnly	ReDim
Region	REM	RemoveHandler	Resume	Return
Select	Set	Shadows	Shared	Short
Single	Static	Paso	Stop	String
Structure	Sub	SyncLock	Then	Throw
A	True	Try	TypeOf	Unicode
Until	volatile	When	While	With
WithEvents	WriteOnly	Xor	eval	extends
Instanceof	package	var		

Anexo No. 5 Nombres de tipos básicos y sustituciones universales

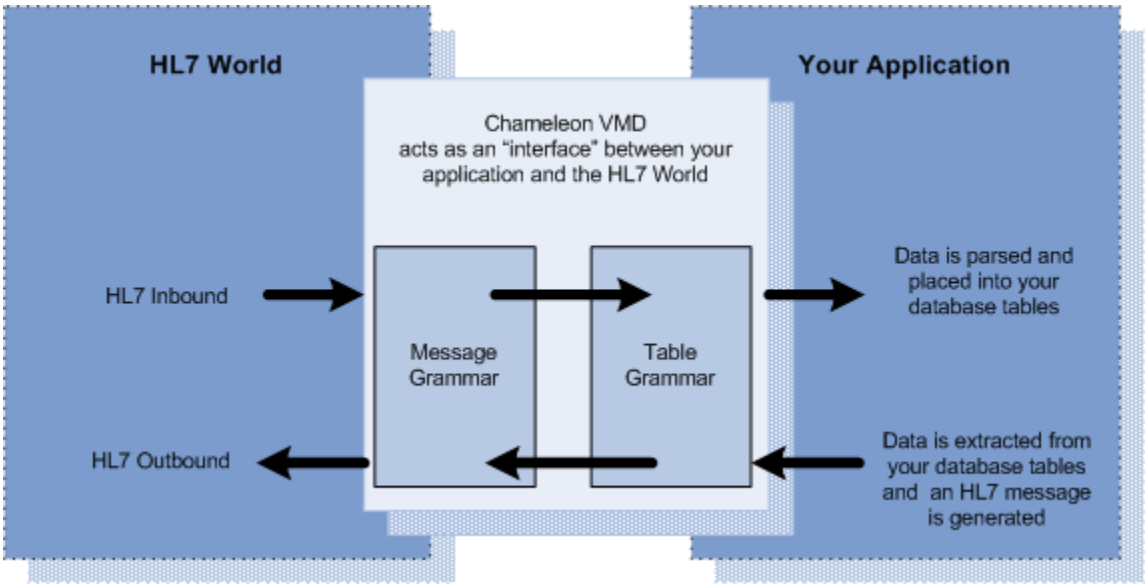
Nombre de tipo C#	Nombre de tipo Visual Basic	Nombre de tipo JScript	Nombre de tipo Visual C++	Representación Ilasm.exe	Nombre de tipo universal
sbyte	SByte	sByte	char	int8	SByte
byte	Byte	byte	unsigned char	unsigned int8	Byte
short	Short	short	short	int16	Int16
ushort	UInt16	ushort	unsigned short	unsigned int16	UInt16
int	Integer	int	int	int32	Int32
uint	UInt32	uint	unsigned int	unsigned int32	UInt32
long	Long	long	__int64	int64	Int64
ulong	UInt64	ulong	unsigned __int64	unsigned int64	UInt64
float	Single	float	float	float32	Single

double	Double	double	double	float64	Double
bool	Boolean	boolean	bool	bool	Boolean
char	Char	char	wchar_t	char	Char
string	String	string	String	string	String
object	Object	object	Object	object	Object

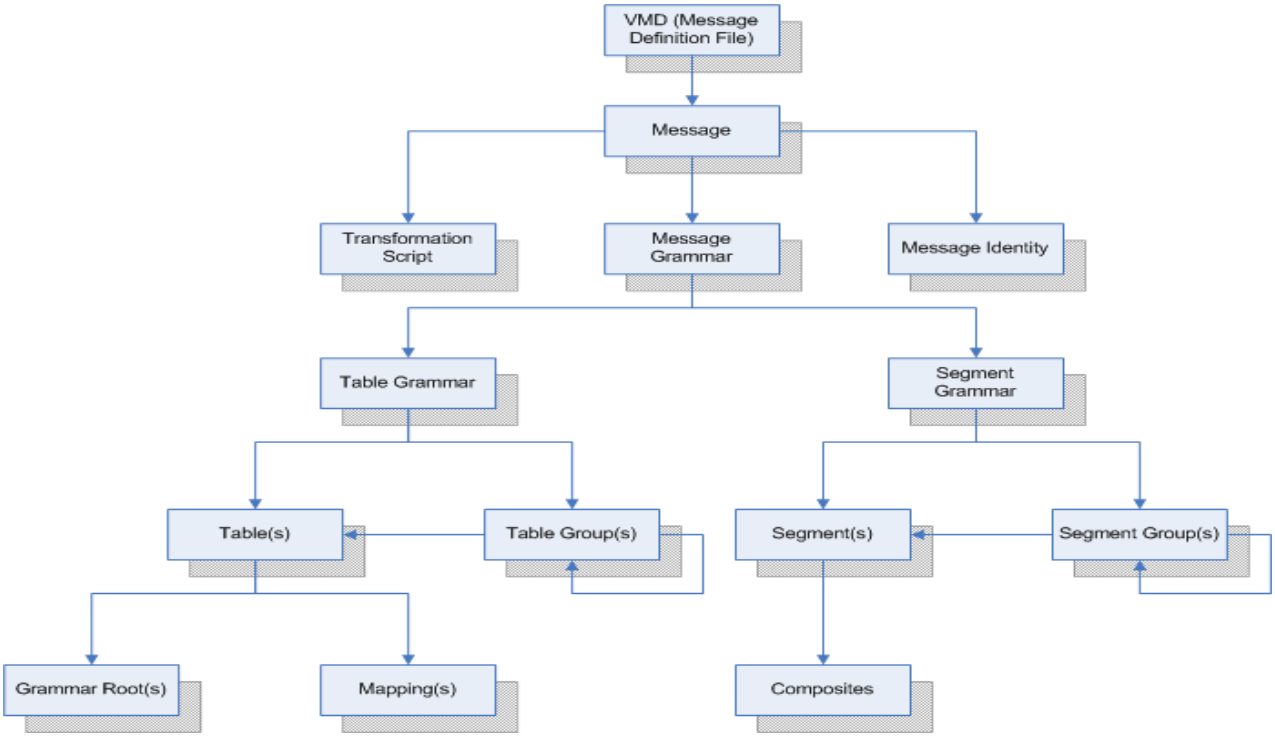
Anexo No. 6 Estructura de sistemas con .NET



Anexo No.7 Esquema de interacción entre el sistema diseñado y el mundo HL7 utilizando Chameleon



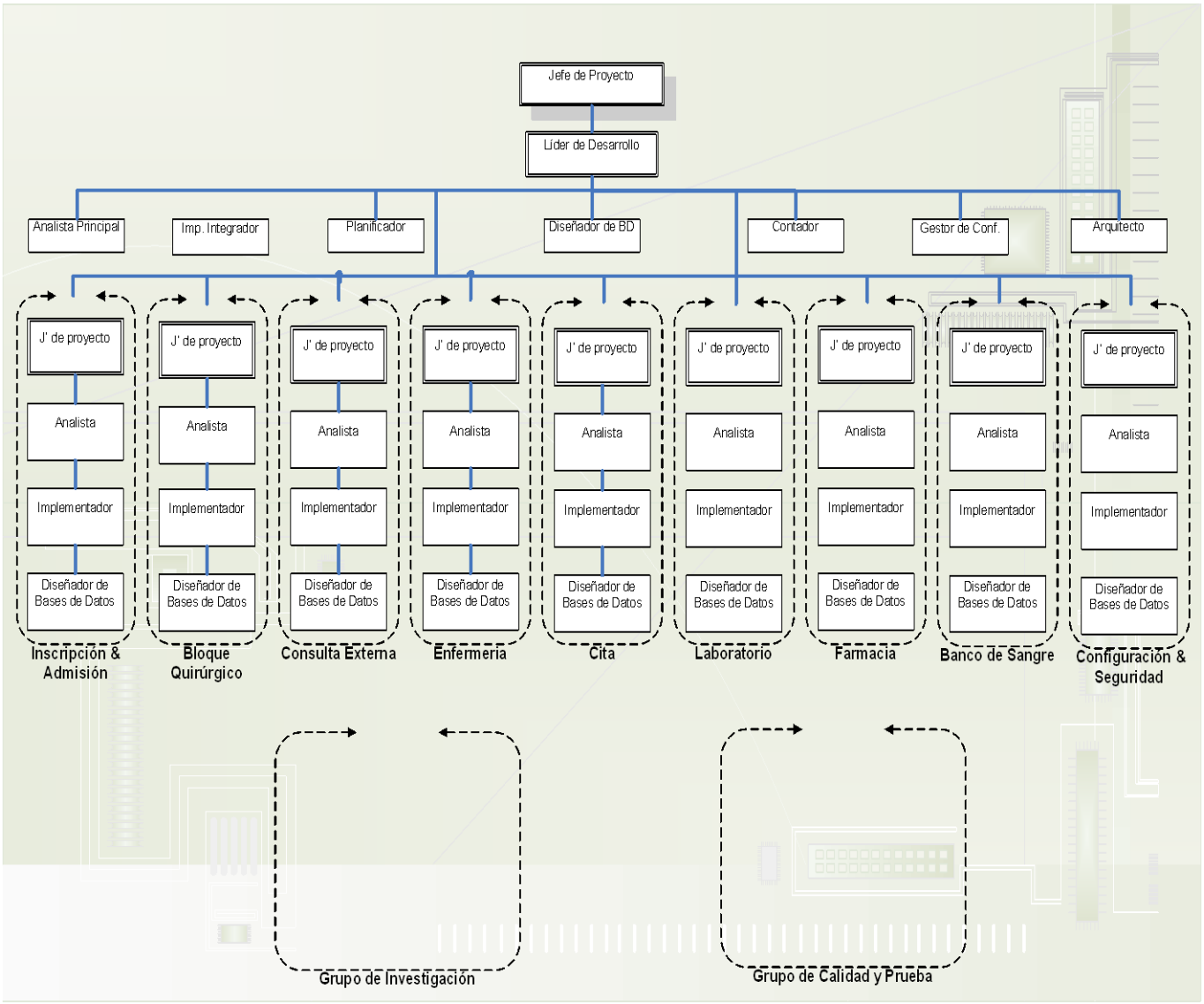
Anexo No.8 Relación entre objetos dentro de Chameleon.



Anexo No.9 Ejemplo de Mensaje HL7 sobre XML

```
<?xml version="1.0" encoding="UTF-8"?>
<OML_O21.PATIENT>
<PID>
<PID.1>1</PID.1>
<PID.3>
<CX.1>A999999999</CX.1>
<CX.5>MRN</CX.5>
</PID.3>
<PID.5>
<XPN.1>
<FN.1>Surname</FN.1>
</XPN.1>
<XPN.2>Forename</XPN.2>
<XPN.7>L</XPN.7>
</PID.5>
<PID.7>
<TS.1>19710424</TS.1>
</PID.7>
<PID.8>F</PID.8>
<PID.11>
<PID.12>PC1 1CP</PID.12>
</PID>
<OML_O21.PATIENT_VISIT>
</OML_O21.PATIENT>
<OML_O21.ORDER>
<OML_O21.ORDER_PRIOR>
<ORC>
<OML_O21.OBRSACOBXTCDNTEDG1OBXTCDNTEPIDPD1PV1PV2AL1ORCOBRNTEOBXNTE>
</OML_O21.ORDER_PRIOR>
</OML_O21.ORDER>
</OML_O21>
```

Anexo No. 10: Estructura jerárgica del proyecto, incluye los módulos que se proponen para la segunda iteración del sistema.



GLOSARIO DE TERMINOS

CGI: Consiste en una especificación para transferir información.

ECMA: Asociación de fabricantes europeos de computadoras dedicada a la estandarización de información y sistemas de comunicación.

FIREWALL: Programa designado para prevenir el acceso no autorizado desde y hacia una red privada.

FTP: Protocolo para la transferencia de archives. Este protocolo es el designado para intercambiar archivos en Internet.

HTML: Lenguaje de marcado de hipertexto, es el lenguaje autoritario para crear documentos en la World Wide Web. Define la estructura de un documento web usando etiquetas y atributos.

HTTP: Protocolo de transferencia de hipertexto, usado en la World Wide Web. Este protocolo define como los mensajes son formateados y transmitidos, además de cuales acciones deben tomar los servidores web y navegadores en respuesta a varios comandos.

ISAPI: Interfaz de programación de Aplicaciones de servidores de Internet. Permiten a los desarrolladores desarrollar aplicaciones web más rápido de lo convencional.

ISO: Organización internacional de estandarización. Ha definido un número importante de estándares computacionales

MSF: El Marco de Soluciones Microsoft abarca temas tales como la estructura del equipo de trabajo que llevará adelante el proyecto (Modelo de Equipos) tanto con el personal de la Consultora como del cliente, y sus roles en el mismo. Cubre el análisis y la mitigación de los riesgos inherentes al proyecto (Modelo de Riesgos), y la forma en que se analiza y establece el alcance del propio proyecto (Modelo de Procesos).

RPC: Protocolo que permite a una computadora ejecutar programas en un servidor.

RUP: Metodología de desarrollo de software basada en UML. Organiza el desarrollo de software en 4 fases.

SMTP: Protocolo de transferencia de correo simple. Diseñado para enviar mensajes de correo electrónico entre servidores.

SQL: Lenguaje estandarizado de consultas utilizado para consultar bases de datos.

SSL: Protocolo desarrollado por Netscape para transmitir datos privados vía internet. Usa un sistema criptográfico compuesto por dos llaves para encriptar los datos.

UDP: Protocolo de conexión. Provee pocos servicios de recuperación de errores aunque ofrece una forma directa para enviar datagramas en una red IP.

XP: Es una disciplina de desarrollo de software que persigue simplificar los procesos de desarrollo. Fue diseñada para ser usada con equipos de desarrollo pequeños que necesiten desarrollos ágiles y con requerimientos cambiantes.