Universidad de las Ciencias Informáticas

”Facultad 3”

Título: “El modelo de diseño del sistema HyperWeb. Módulos de Tratamiento Farmacológico y Configuración”

Trabajo de Diploma para optar por el título de

Ingeniero en Ciencias Informáticas

Autor(es): Raykenler Yzquierdo Herrera y René Lazo Ochoa

Tutor(es): Alfredo Morales Oliva

Consultante: Dr. C. Pedro Y. Piñero Pérez

Mayo de 2007

DECLARACIÓN DE AUTORÍA

Nosotros Raykenler Yzquierdo Herrera y René Lazo Ochoa declaramos que somos los únicos autores de este trabajo y autorizamos a la Facultad 3 de la Universidad de Ciencias Informáticas a hacer uso del mismo en su beneficio.

Para que así conste firmamos la presente a los 16 días del mes de mayo del año 2007.

Raykenler Yzquierdo Herrera

_____

Firma del autor

René Lazo Ochoa

_____

Firma del autor

Alfredo Morales Oliva

_____

Firma del tutor

AGRADECIMIENTOS

A todos aquellos hombres que el paso de nuestra historia cubrió de gloria.

A la Revolución, que nos dió la posibilidad de lo alcanzado.

A nuestro tutor y hermano Alfredo, que siempre estuvo compartiendo el mismo plato.

Al Dr. Pedro Piñero que ha sido más que un amigo. Cada paso en esta universidad llevó tu guía.

A la Ms. C. Eugenia Muñiz y a la memoria de Germinal Álvarez por su apoyo y guía.

A Ernesto Medina: hermano tu formas parte de este resultado, muchas gracias de corazón.

A Yelaine: gracias pollo Gutiérrez.

A nuestras hermosas Heidy y Doris, siempre presentes, que tanto han tenido que pasar con nosotros. Esto también es el resultado de sus petates, gracias.

A Ivis: por no cansarte nunca. Tu amor me hizo mejor persona.

A Ireldita, Juan Carlos, Rodelay y Roxana: nunca hubo tan buena combinación para dar amor.

A Medardo: tu también fuiste parte de este logro.

A Marlene: tu ejemplo me guió hasta aquí, tu que eres mi conciencia, mi inspiración y mi fuerza.

A Mundy y Dairón que me obligan a no cansarme de luchar por convertirme en un ejemplo digno. Ustedes lo harán todo mejor que yo mis hermanos.

A Rene Lazo: siempre presente, este resultado también es tuyo viejo, gracias por tu incondicional apoyo.

A nuestros abuelos, los que están y no están, ustedes nos trajeron hasta aquí, gracias por este viaje, que la vida nos deje retribuirles lo tanto.

A Maguy y Adircito, que siempre creyeron que llegaría, siempre presentes, mil gracias por tanta fe.

A Rey: gracias por lo enseñado y entregado en estos años, me diste un desafío y me mostrarte un camino. ¡He tratado de cumplir!. Mil gracias hermano.

A nuestros familiares, a los cuales no defraudaremos jamás, no los mencionamos porque ustedes son demasiados.

A Juan Carlos y Pocholo, hermanos de la vida, ustedes que sudaron la misma camiseta que yo, gracias.

A todas las personas que nos apoyaron el camino transitado y nos dieron razones para seguir adelante, de corazón un millón de gracias.

DEDICATORIA

A Fidel y a nuestros familiares y amigos…

RESUMEN

Numerosos proyectos de software nunca llegan a concretarse o nacen con problemas de flexibilidad, extensibilidad, bajo acoplamiento, reusabilidad de código y alta cohesión desde sus inicios. Esto se debe a la falta o incorrecta aplicación de un modelo de diseño de sistema en el desarrollo de los mismos. Las aplicaciones médicas no se excluyen de presentar estas problemáticas. En este trabajo se describe el diseño de sistema de los módulos de Tratamiento Farmacológico y Configuración del proyecto de software HyperWeb. Para ello se ha desarrollado un estudio de la producción de software dedicado a la medicina y de las diferentes vertientes que aborda el diseño de sistemas informáticos. Se han creado como parte de la solución propuesta algoritmos que emplean técnicas de Inteligencia Artificial para la ayuda a la toma de decisiones en el tratamiento farmacológico de pacientes hipertensos y se aplicaron convenientemente patrones de diseño estandarizados. Finalmente se evaluó el modelo propuesto mediante el uso de métricas dirigidas a garantizar la calidad del modelo de diseño, obteniendo resultados positivos.

ÍNDICE

Índice de contenidos:

Índice de Tablas:

Índice de Figuras:

INTRODUCCIÓN

La Hipertensión Arterial (HTA) existe en todas las regiones del mundo, atendiendo a múltiples factores económicos, sociales, culturales, ambientales y étnicos. La prevalencia ha estado en aumento, asociada a patrones alimentarios inadecuados, disminución de la actividad física y otros aspectos conductuales relacionados con hábitos tóxicos (PÉREZ CABALLERO *et al.* 2004).

En el mundo se estima que 691 millones de personas la padecen y 15 millones de muertes son causadas por enfermedades circulatorias, 7,2 millones son por enfermedades coronarias del corazón y 4,6 millones por enfermedad vascular encefálica, todas ellas enfermedades en las que la Hipertensión Arterial tiene alta influencia (PÉREZ CABALLERO *et al.* 2004).

El tratamiento hipertensivo y el seguimiento de los pacientes hipertensos es una tarea médica compleja, a pesar de ello, desde el punto de vista informático esta labor encuentra algunos intentos de realización. En nuestro país varias instituciones han trabajado y establecido diversos proyectos colaborativos con el objetivo de informatizar los procesos de diagnóstico, control y seguimiento de pacientes hipertensos. A continuación se muestra un conjunto de aplicaciones desarrolladas en este sentido:

El sistema HIPERTENCID obtuvo el Registro Médico otorgado por el Centro de Control de Equipos Médicos (CECEM) en 1994 y una extensión de su uso en el año 2000. El mismo cuenta con una eficacia que ha sido comprobada por especialistas médicos del país, indica tratamiento, sigue la evolución de los pacientes, controla su asistencia a consultas, modifica el tratamiento según fuera este siendo menos efectivo y realiza prevención y control de la Hipertensión Arterial a través de modificaciones del estilo de vida del paciente (MUÑIZ LODOS 1994), pero por el momento histórico en que fue diseñado, en el cual imperaba la programación estructurada, y el sistema Operativo sobre el que esta soportado, el cual es MS-DOS, este sistema ha quedado obsoleto y se ha hecho difícil la reutilización de las funcionalidades implementadas.

Otro sistema desarrollado en nuestro país es, Proyección del Centro de Desarrollo Electrónico hacia la Comunidad (TENSOFT II), el mismo formó parte de un proyecto de investigación conjunta entre la Universidad Central de Las Villas y la Universidad de Oviedo. En este proyecto especialistas en Medicina General Integral (MGI) pueden desarrollar la captura de los datos de los pacientes y el diagnóstico automatizado de la HTA a través del software. El mismo posee un módulo de Inteligencia Artificial que se fundamenta en la aplicación de reglas para la toma de decisión, basadas en los criterios obtenidos del

Comité de Expertos, emitiéndose el diagnóstico de manera general y detallada para cada caso (GONZÁLEZ RODRÍGUEZ *et al.* 2004a).

Por la complejidad de este tipo de aplicaciones y la constante exposición a las variaciones de las reglas de negocio, se hace imprescindible el desarrollo del análisis y diseño de sistema conforme lo que estipulan las metodologías de desarrollo, buscando mayor flexibilidad y adaptabilidad de los componentes de software que van siendo obtenidos, para así disminuir el impacto de los riesgos, ante la incertidumbre del equipo de desarrollo sobre el dominio del conocimiento médico, que siempre esta presente en el trabajo con especialistas de la salud. Por ello este trabajo de diploma estará concentrado en el desarrollo teórico y práctico de un modelo de diseño de sistema para los módulos de Tratamiento Farmacológico y Configuración del sistema HyperWeb.

El diseño del software se encuentra en el núcleo técnico de la ingeniería del software y se aplica independientemente del modelo de diseño de software que se utilice. Una vez que se analizan y especifican los requisitos del software, el diseño del software es la primera de las tres actividades técnicas (diseño, generación de código y pruebas) que se requieren para construir y verificar el software. Cada actividad transforma la información de manera que de lugar por último a un software de computadora validado (PRESSMAN 1998).

Por la importancia y alcance del diseño dentro del proceso de desarrollo de software, en este trabajo se realizó un análisis detallado de las características de esta actividad para buscar una solución concreta al problema planteado.

Algunos de los problemas más generalizados del diseño de software son la poca flexibilidad y extensibilidad de los diseños de clases, lo que imposibilita la capacidad adaptativa y de reutilización de los módulos o unidades de codificación. Otro problema que se observa es el alto acoplamiento entre los submódulos que encarecen el mantenimiento, y dificultan las posibilidades de integración entre subsistemas codificados y otros sistemas informáticos que podrían ser afines con el que se está desarrollando. Así también la baja cohesión del diseño de subsistemas o clases de los mismos, provoca la reprogramación de funcionalidades, influyendo de forma negativa en la productividad y en la calidad, encareciendo el mantenimiento, la adaptación a cambios y la reutilización de las unidades de software en desarrollo.

La diferencia de paradigma debido al momento histórico, con que fue concebido HIPERTENCID (Programación Estructurada) antecesor de HyperWeb y el paradigma utilizado para desarrollar este último

(Orientado Objeto), ha imposibilitado la reutilización de componentes de negocio que nos ayudarían a mitigar dificultades y disminuir el tiempo de desarrollo.

Por todo lo antes mencionado este trabajo de diploma se propone resolver el siguiente **problema**: la no existencia de un modelo de diseño de sistema para los módulos de Tratamiento Farmacológico y Configuración del sistema HyperWeb.

Por consiguiente el **objeto de estudio** de este trabajo, radica en el desarrollo de proyectos de software médico orientados a la HTA, y el **campo de acción** es la creación de diseños de sistemas de proyectos de software médico orientados a la HTA.

La correcta realización del diseño del sistema de los módulos de Tratamiento Farmacológico y Configuración pueden garantizar una alta probabilidad de obtener un sistema con flexibilidad, extensibilidad, bajo acoplamiento, reusabilidad, alta cohesión, y por tanto, facilidad de mantenimiento, facilidad en la Gestión de Cambios y una alta posibilidad de lograr buena calidad en el producto final.

El **objetivo general** de este trabajo es desarrollar el modelo de diseño del sistema HyperWeb para los módulos de Tratamiento Farmacológico y Configuración.

Para dar cumplimiento al objetivo antes planteado se proponen los siguientes **objetivos específicos**:

- Realizar un análisis del estado del arte de las diferentes tendencias del diseño de aplicaciones y específicamente el de sistemas médicos dirigidos a la HTA.
- Desarrollar el modelo de diseño del sistema del módulo Tratamiento Farmacológico.
- Desarrollar el modelo de diseño del sistema del módulo Configuración.
- Analizar los resultados obtenidos y la calidad del modelo de diseño del sistema desarrollado.

Para llevar a cabo este trabajo y dar cumplimiento a los objetivos del mismo se concibieron las siguientes tareas de investigación:

1. Estudio del estado del arte de otros sistemas dedicados al tratamiento de la HTA.
2. Estudio del estado del arte de las diferentes tendencias que abordan el diseño de sistema en el desarrollo de software.
3. Estudio del estado del arte de técnicas de Inteligencia Artificial (IA) que ayude a la toma de decisiones en el tratamiento de la HTA.

4. Diseño de algoritmos que permitan el tratamiento farmacológico y control de pacientes con HTA.
5. Selección y adaptación de la metodología para el diseño de sistema atendiendo a la tecnología seleccionada y al problema específico a resolver.
6. Aplicación de patrones de arquitectura y diseño.
7. Realización del diagrama de clases de los módulos de Tratamiento Farmacológico y Configuración.
8. Realización de los diagramas de secuencia de los módulos de Tratamiento Farmacológico y Configuración.
9. Realización del modelo de datos para los módulos de Tratamiento Farmacológico y Configuración.
10. Realización del diagrama de despliegue de los módulos de Tratamiento Farmacológico y Configuración.
11. Realización de un análisis crítico basado en indicadores de calidad al diseño del sistema.

El trabajo consta de tres capítulos, el primero trata la fundamentación teórica del trabajo, el mismo incluye un estudio del estado del arte de las tendencias, técnicas, tecnologías, metodologías y software usados en la actualidad referidos al objeto de estudio. El segundo capítulo realiza un análisis de las herramientas, metodologías y patrones empleados en el modelo de diseño propuesto. El capítulo tercero consiste en el desarrollo de la solución propuesta y muestra un análisis de los resultados obtenidos.

1

CAPÍTULO

FUNDAMENTACIÓN TEÓRICA

Introducción

En este capítulo se muestra la influencia de la Hipertensión Arterial como enfermedad que afecta a los seres humanos, enfatizando en las razones que hacen necesitar un apoyo en la toma de decisiones del tratamiento farmacológico correspondiente a un paciente hipertenso. Se lleva a cabo un análisis de las herramientas y sistemas informáticos que se encuentran actualmente en utilización a nivel mundial y nacional para ayudar en la atención de esta enfermedad.

Se realiza además una evaluación de las problemáticas existentes en materia de diseño, específicamente para aplicaciones médicas. Se da una valoración de los principales principios del diseño y se señalan las principales tendencias que llevan a malas prácticas y errores a la hora de desarrollar el diseño de sistemas. De igual forma se analizan técnicas de Inteligencia Artificial para una posible adaptación y aplicación en la problemática planteada.

Estado actual de la Hipertensión Arterial

La Hipertensión Arterial es una enfermedad que de forma común afecta la salud de cerca de 691 millones de personas alrededor del mundo. Según estadísticas realizadas en Cuba, durante el 2005, son 2190 080 las personas que padecen HTA (CUETO 2006).

Se considera también un factor de riesgo importante para otras enfermedades, fundamentalmente para la Cardiopatía Isquémica, Insuficiencia Cardíaca, Enfermedad Cerebro Vascular, Insuficiencia Renal, puede también producir afectaciones de la vasculatura periférica y de la retina.

Datos del Framinghan Herat Study sugieren que los individuos normotensos mayores de 55 años tienen un 90 % de probabilidad de riesgo de desarrollar HTA (CHOBANIAN *et al.* 2003).

La relación de Presión Arterial (PA) y el riesgo de eventos de Enfermedades Cardiovasculares (ECV) es continua, consistente e independiente de otros factores de riesgo. Cuanto más alto es la presión arterial, mayor es la posibilidad de Infarto de Miocardio, Insuficiencia Cardíaca, Ictus y Enfermedad Renal. Para

individuos de entre 40 y 70 años, cada incremento de 20 mmHg en la Presión Arterial Sistólica (PAS) ó 10 mmHg en la Presión Arterial Distólica (PAD) dobla el riesgo de ECV en todo el rango desde 115/75 hasta 185/115 mmHg (CHOBANIAN *et al.* 2003).

La prevención de la Hipertensión Arterial es la medida terapéutica sanitaria más importante, universal y menos costosa**.** El perfeccionamiento de la prevención y el control de la PA es un desafío importante para todos los países y debe constituir una prioridad de las instituciones de salud, la población y los gobiernos. La adecuada percepción del riesgo que significa padecer HTA obliga a ejecutar una estrategia poblacional con medidas de educación y promoción, dirigidas a la disminución de la presión arterial media de la población, impactando sobre otros factores de riesgo asociados a la Hipertensión Arterial, fundamentalmente la falta del ejercicio físico, niveles inadecuados de lípidos sanguíneos, elevada ingesta de sal, el tabaquismo, el alcoholismo y la obesidad, que puede lograrse mediante acciones dirigidas a las modificaciones del estilo de vida (CABALLERO 2006).

Por otra parte, es necesaria una estrategia individual, para detectar y controlar con medidas específicas de los servicios asistenciales, a los individuos que por estar expuestos a niveles elevados de uno o varios de los factores de riesgo antes señalados padecen de Hipertensión Arterial o tiene alta posibilidades de padecerla (CABALLERO 2006; CHOBANIAN *et al.* 2003).

De este modo, es imprescindible lograr la terapéutica más acertada para mantener un adecuado control de las cifras tensiónales.

Tratamiento farmacológico

Para iniciar la terapia farmacológica antihipertensiva, hay que tener confirmación de las cifras de PA elevadas, la presencia de Daño en Órganos Diana, la presencia de Enfermedad Cardiovascular u otros factores de riesgo asociados, ya que el tratamiento no debe limitarse al control de la Hipertensión Arterial (CABALLERO 2006).

El tratamiento debe formularse sobre la base de aspectos importantes del paciente, como puede ser su edad, necesidades individuales de fármacos y dosificación, así como el grado de respuesta a la terapia. Es trascendental lograr la adhesión del paciente al tratamiento por lo que debe constituir el objetivo primero. Las fórmulas más adecuadas y óptimas son las que garantizan un nivel de efectos terapéuticos durante 24 horas.

No existe un fármaco ideal de uso generalizado para todos los pacientes por lo que es imprescindible lograr la individualización del tratamiento, de forma escalonada y progresiva, hasta lograr los objetivos

deseados. Hay que considerar también los posibles efectos secundarios que pueden producir los diferentes fármacos (CABALLERO 2006).

El potasio sérico y la creatinina deberían ser medidas al menos 1 o 2 veces al año. Después de conseguir el objetivo y la estabilidad en la PA, las visitas de seguimiento pueden ser usualmente en intervalos de 3 a 6 meses. La terapia con dosis bajas de aspirina debería ser considerada solo cuando la PA está controlada, porque el riesgo de Ictus Hemorrágico está incrementado en pacientes con HTA no controlada.

Los Diuréticos tipo Tiazida deberían ser usados en el tratamiento farmacológico en la mayoría de los pacientes con HTA no complicada, bien solos o combinados con otras clases de fármacos. Ciertos estados de alto riesgo constituyen indicaciones para el tratamiento inicial con otras clases de fármacos antihipertensivos (IECAs, ARA-2, Betabloqueantes, Bloqueantes de los Canales del Calcio).

Un segundo fármaco de diferente clase debería introducirse cuando la monoterapia en dosis adecuadas falla para conseguir el objetivo de PA. Cuando la PA es mayor de 20/10 mmHg sobre el objetivo a lograr con el paciente, se debería considerar iniciar la terapia con dos fármacos, bien como prescripciones separadas o combinaciones en dosis fijas. La iniciación de la terapia farmacológica con más de un agente puede incrementar la posibilidad de conseguir el objetivo de PA de forma oportuna, pero es precisa una singular precaución en aquellos pacientes con riesgo de hipotensión ortostática, como diabéticos, disfunción autonómica, y algunas personas ancianas (CABALLERO 2006).

Si la PA es mayor de 20/10 mmHg que el objetivo de presión arterial, debería considerarse iniciar la terapia con dos fármacos, uno de los cuales debería ser por norma un Diurético tipo Tiazida.

Si un fármaco no es tolerado o está contraindicado, debería usarse uno de los de otra clase que haya demostrado reducción en eventos cardiovasculares.

Es posible indicar hasta 3 fármacos en dependencia de las características del paciente (CHOBANIAN *et al.* 2003).

Estado actual de las herramientas informáticas para la atención a pacientes con Hipertensión Arterial

Como se ha analizado la indicación y control del tratamiento farmacológico de los pacientes hipertensos constituyen procesos médicos complejos. Desde el punto de vista informático esto ha sido un reto

importante, que aunque difícil, ha sido acometido por diferentes instituciones y empresas. Nuestro país también ha incursionado en esta área como es el caso de HIPERTENCID, referenciado anteriormente. Se ha realizado el estudio de algunas herramientas desarrolladas o en desarrollo enfocadas en la automatización, en alguna medida, de los procesos de diagnóstico, tratamiento y seguimiento de pacientes con HTA. Algunos ejemplos se ilustran a continuación:

Proyecto HIPERTENCID

El sistema HIPERTENCID, fue utilizado durante años, demostrando su efectividad en las sugerencias de indicación de tratamiento y seguimiento de la evolución de los pacientes. El programa permitió el control de la asistencia a consultas, modificaba el tratamiento buscando efectividad y realizaba prevención y control de la HTA a través de modificaciones del estilo de vida del paciente (MUÑIZ LODOS 1994).

Programa Corresponde

Sistema informático que pretende dotar a los servicios asistenciales de instrumentos que faciliten y mejoren la gestión clínica de los pacientes e implicar a los mismos, de modo que conciban la prevención como un elemento decisivo en el tratamiento de este tipo de enfermedades. Todo esto a través de la Web.

Su despliegue se realizó a nivel nacional en España para fomentar la corresponsabilidad médico-paciente en prevención cardiovascular, para un mejor control de los factores de riesgo en los pacientes cardiovasculares, y reducir así, su morbi-mortalidad.

La Sociedad Española de Hipertensión-Liga Española para la Lucha contra la Hipertensión Arterial (SEH-LELHA) participa desde el inicio en el desarrollo e implementación de este programa, iniciativa de Novartis Farmacéutica, que cuenta con el reconocimiento y apoyo del Foro Español de Pacientes. Tiene como objetivo mejorar el tratamiento y manejo de la población hipertensa española.

Permite informar y formar claramente al paciente sobre sus objetivos individuales y opciones terapéuticas indicadas a nivel de cambios en estilos de vida, tratamiento farmacológico y exploraciones complementarias, de forma que éste pueda corresponsabilizarse con su médico en la consecución de los objetivos de control (TORREGROSA 2006).

Proyección del Centro de Desarrollo Electrónico hacia la Comunidad

La Universidad Central de Las Villas y la Universidad de Oviedo desarrollaron otros de los proyectos relacionados con la atención a la hipertensión. Hace algunos años se creó la "Proyección del Centro de

Desarrollo Electrónico hacia la Comunidad", teniendo como objetivo principal desarrollar un estudio de personas supuestamente normotensas. Como parte del proyecto, especialistas en Medicina General Integral (MGI) de los centros hospitalarios José Ramón León, Chiqui Gómez y Ramón Pando Ferrer realizaron la captura de los datos, mientras que el centro realizaba el diagnóstico automatizado de la HTA a partir del software TENSOFT II. El programa computacional utiliza Access 2000 de la firma Microsoft como bases de datos, y cuenta con un grupo de macros y módulos programados sobre Visual Basic for Applications, incluyendo también un módulo de seguridad construido apoyándose en SQL Server. Emplea un sistema de reglas para toma de decisiones respaldado por el conocimiento obtenido del Comité de Expertos, el cual le asignó a cada epígrafe de la Historia Clínica una puntuación, para ponderar los factores que más influyen en la HTA, emitiéndose el diagnóstico de manera general y detallada para cada caso (GONZÁLEZ RODRÍGUEZ *et al.* 2004b). El software cuenta con valoraciones estadísticas, sólidos sistemas de seguridad, información, ayudas y manuales de usuarios. También hace sugerencias en el diagnóstico y la clasificación de la HTA.

Modelo con RNA para Diagnosticar Hipertensión Arterial

El trabajo consiste en desarrollar, por parte de Universidad de Guadalajara y el Centro Universitario de Ciencias Exactas e Ingenierías de México, un sistema con aplicaciones en el ámbito hospitalario, para interpretar la detección biológica de personas hipertensas, clasifica y reconoce la HTA a través de técnicas de Inteligencia Artificial basadas en redes neuronales artificiales (RNA). Se optimiza el cuestionario médico de problemas con la HTA, generándose un arreglo de minimización matricial. Se hace referencia a antecedentes que datan de la década de los 40, tales como la máquina de Turing. Se especifican las principales características eléctricas de las neuronas biológicas. Se aplica una red neuronal realimentada con el método de aprendizaje para una memoria autoasociativa bidireccional (BAM). Mediante la implantación en Mathcad se realiza la simulación y modelación del arreglo en la RNA que apoya a los diagnósticos en los que se involucra dicho tipo de señales biológicas (MATEOS *et al.* 2003).

RCV-SEMG 2.0

La herramienta informática, RCV-SEMG 2.0, fue realizada por la Sección Informática de la Sociedad Española de Medicina General (SEMG). Este programa informático ofrece un dispositivo en el que ir almacenando la historia clínica de cada paciente desde la propia consulta, al tiempo que orienta al

profesional en el diagnóstico y tratamiento de los factores de riesgo cardivascular (FRCV), aplicando los criterios recogidos en las guías de más amplio consenso y utilizando aquellos modelos de estratificación del riesgo cardiovascular (RCV) más adecuados a cada entorno epidemiológico. Es decir, permite personalizar para cada paciente y momento toda acción del médico (SEMG 2006).

HyperTA Control

Es una aplicación diseñada con el programa Microsoft Access 2000, de la suite Microsoft Office, para el registro y consulta de datos de salud de pacientes adultos, en el marco de programas de control o de la atención médica de la Hipertensión Arterial. Ha sido diseñada para ser usada por médicos, enfermeros y personal de salud en general, que trabajan en Consultorios, Dispensarios, Salas de Enfermería, Centros de Atención de la Salud, Clínicas o Sanatorios, Hospitales, Obras Sociales, Centros de Jubilados, etc. Cuenta con numerosos formularios que facilitan el registro, la consulta y la edición de datos, e informes que permiten la edición e impresión de dichos datos. Para una mayor sencillez de su instalación y uso, ha sido desarrollada para un entorno monousuario, esto es, para su instalación y administración en una única PC. No ayuda en el proceso de indicación y control del tratamiento farmacológico. Tiene un precio de 130 dólares (RUGGERI 2004b).

HyperTA Calc

Es una aplicación diseñada con el programa Microsoft Access 2000, de la suite Microsoft Office, que permite ponderar los valores de presión arterial de niños, preadolescentes y adolescentes, cuyas edades van desde los 2 hasta los 18 años. Se trata de una calculadora diseñada para facilitarle, al personal de salud especializado, la valoración de la presión arterial en estos casos. Es de fácil operación, y permite la impresión de los resultados (RUGGERI 2004a).

Se analizaron un conjunto de herramientas que recogen el proceso de atención a pacientes hipertensos, algunas de estas hacen uso de técnicas de Inteligencia Artificial impulsados por la complejidad de las situaciones. La ayuda a la toma de decisiones en la indicación de tratamiento farmacológico de pacientes, el seguimiento y control pueden ser tareas difíciles, al igual que la configuración de un sistema que sea lo suficientemente flexible. Los sistemas analizados presentan problemas en este sentido ya sea porque no recogen o apoyan los procesos antes mencionados o porque no son lo suficientemente modernos. Es por esto que el diseño de la solución propuesta significa la

posibilidad del paso a la implementación de una herramienta integral en el tratamiento farmacológico de pacientes con HTA.

Estado actual del Diseño de software

Actualmente el concepto de diseño es orientado a diferentes aristas como la que plantea Grady Bosch, "el diseño es el proceso de determinar una implementación efectiva y eficiente que realice las funciones y tenga la información del análisis de dominio", o la planteada por R.S. Pressman, "el diseño del software se encuentra en el núcleo técnico de la ingeniería del software y se aplica independientemente del modelo de diseño de software que se utilice. Una vez que se analizan y especifican los requisitos del software, el diseño del software es la primera de las tres actividades técnicas - diseño, generación de código y pruebas - que se requieren para construir y verificar el software. Cada actividad transforma la información de manera que de lugar por último a un software de computadora validado" (PRESSMAN 1998). Otra idea importante que refiere Pressman consiste en la importancia del diseño del software, pues según él, este flujo se puede describir con una sola palabra -calidad- (PRESSMAN 1998). "El diseño es el lugar en donde se fomentan la calidad en la ingeniería del software. El diseño proporciona las representaciones del software que se pueden evaluar en cuanto a calidad. El diseño es la única forma de convertir exactamente los requisitos de un cliente en un producto o sistema de software finalizado. El diseño del software sirve como fundamento para todos los pasos siguientes del soporte del software y de la ingeniería del software. Sin un diseño, corremos el riesgo de construir un sistema inestable, un sistema que falla cuando se lleven a cabo cambios, un sistema que puede resultar difícil de comprobar; y un sistema cuya calidad no puede evaluarse hasta muy avanzado el proceso, sin tiempo suficiente y con mucho dinero gastado" (PRESSMAN 1998).

En el caso de las metodologías, como por ejemplo RUP, los criterios sobre el diseño de sistemas plantean que es el flujo del trabajo donde el arquitecto y los diseñadores de sistema, describen de una forma más concreta y acorde al tipo de tecnología, el cómo serán implementadas las funcionalidades del sistema en construcción, basándose en los tipos de colaboración, la relación que existe entre las clases, y las responsabilidades de cada unidad de código (RUMBAUGH *et al.* 1998). Otra metodología muy utilizada es XP (Extreme Programming), la misma describe el diseño con un enfoque práctico y dinámico para el cual el diseño de sistema es una actividad que construye un proceso de diseño evolutivo que se basa en refacturar un sistema simple en cada iteración. Todo el diseño se centra en la iteración actual y no se hace nada anticipadamente para necesidades futuras. Sin dudas algunas uno de los principios de la

programación extrema es la simplicidad, de aquí que el diseño no escape a esta filosofía. El diseño debe ser lo más simple posible. El paradigma KISS ("Keep It Small and Simple" para unos o "Keep it Simple, Stupid" para otros) se lleva hasta las últimas consecuencias; por ejemplo, se hace énfasis en no añadir funcionalidad al diseño del sistema nunca antes de lo necesario, por las sencillas razones de que probablemente ahora mismo no sea lo más prioritario o porque quizás nunca llegue a ser necesaria (FOWLER 2003).

Algunas empresas, como DEADELUS[1], plantean que el diseño de sistemas se ocupa de desarrollar las directrices propuestas durante el análisis en términos de aquella configuración que tenga más posibilidades de satisfacer los objetivos planteados tanto desde el punto de vista funcional como del no funcional. El proceso de diseño de un sistema complejo se suele realizar de forma descendente: Diseño de alto nivel (o descomposición del sistema a diseñar en subsistemas menos complejos), diseño e implementación de cada uno de los subsistemas, especificación consistente y completa del subsistema de acuerdo con los objetivos establecidos en el análisis, desarrollo según la especificación, prueba, integración de todos los subsistemas, validación del diseño (DAEDALUS 2006).

Microsoft por su parte plantea en su modelo Microsoft Solution Framework (MSF) para el desarrollo de sistemas de software, que el diseño es una actividad que debe ser realizada con el fin de establecer el comportamiento dinámico del sistema, es decir, como debe reaccionar ante los acontecimientos. Para su desarrollo también se utilizan los lenguajes de modelado. El resultado obtenido de la etapa de diseño facilita enormemente la implementación posterior de los sistema, pues proporciona la estructura básica del sistema y como los diferentes componentes actúan y se relacionan entre ellos (MICROSOFT 2007a).

Partiendo del estado del arte de los conceptos que hoy se manejan en el mundo sobre la importancia y la responsabilidad del "Diseño del Sistema", planteados tanto por personalidades de la teoría del desarrollo de software, como por metodologías de desarrollo de software, o algunas empresas de desarrollo de software y teniendo en cuenta las características de la solución propuesta, se conceptualiza la actividad del diseño como un proceso y un modelado a la vez, de mucha importancia en la vida de un proyecto de software. Debido a que el proceso de diseño es además un conjunto de pasos repetitivos que permiten al diseñador describir todos los aspectos del sistema a construir, partiendo de las especificaciones que llegan del flujo de análisis; es el flujo que describe de una forma más concreta y teniendo en cuenta la tecnología y el lenguaje de programación que se utiliza, la representación

1Empresa de software © DAEDALUS - Data, Decisions and Language, S. A.

estructural del sistema que se desea construir. A lo largo del diseño se evalúa la calidad del desarrollo del proyecto apoyándonos en un conjunto de revisiones técnicas. El diseño debe implementar todos los requisitos explícitos contenidos en el modelo de análisis y debe acumular todos los requisitos implícitos que desea el cliente. Debe ser una guía que puedan leer y entender los que construyan el código y los que prueban y mantienen el software. El diseño debe proporcionar una completa idea de lo que es el Software, enfocando los dominios de datos, funcional y comportamiento desde el punto de vista de la implementación. En el diseño se deben concretar las representaciones funcionales de interacción entre los siguientes aspectos, Cliente - Sistema, Sistema – Dispositivos, Sistema – Soportes de almacenamientos, y es una base para garantizar un producto acorde con las necesidades del cliente.

Evolución del diseño

Si se analiza la evolución del diseño de software, se tendría que comenzar diciendo que los intentos por desarrollar programas con cierta modularidad y métodos para perfeccionar las estructuras de forma descendente fueron las primeras cosas hechas en materia de diseño (DENNIS 1972).

En la década del sesenta surgen manifestaciones de lo que más tarde se llamó Programación Estructurada, posteriormente se liberó el conjunto de las llamadas "Técnicas para mejoramiento de la productividad en programación", siendo la Programación Estructurada una de ellas (BAKER 1975).

Los programas fueron escribiéndose con un mayor grado de estructuración, lo cual les permitía ser fácilmente comprensibles en actividades tales como pruebas, mantenimiento y modificación de los mismos. Mediante la programación Estructurada todas las bifurcaciones de control de un programa se encuentran estandarizadas, de forma tal que es posible leer la codificación del mismo desde su inicio hasta su terminación en forma continua, sin tener que saltar de un lugar a otro del programa siguiendo el rastro de la lógica establecida por el programador, como es la situación habitual con codificaciones desarrolladas bajo otras técnicas (MILLS 1972).

Posteriormente se proponen métodos para el flujo de datos o estructura de datos en una definición de diseño (JACKSON 1975; W. STEVENS 1974). Enfoques más recientes proponen el diseño orientado a objetos.

Considerando el paradigma de la orientación a objetos, el diseño orientado a objetos es más complejo que el diseño estructurado clásico, ya que lo que se busca es crear un diseño genérico, abierto y concreto. El diseño orientado a objetos se define como un diseño de sistemas que utiliza objetos auto-contenidos y clases de objetos.

En la actualidad se busca básicamente un diseño basado en la arquitectura de software.

En su forma más simple, la arquitectura es la estructura jerárquica de los componentes del programa (módulos), la manera en que los componentes interactúan y la estructura de datos que van a utilizar los componentes. Sin embargo, en un sentido más amplio, los componentes se pueden generalizar para representar los elementos principales del sistema y sus interacciones. Un objetivo del diseño del software es derivar una representación arquitectónica de un sistema. Esta representación sirve como marco de trabajo desde donde se llevan a cabo actividades de diseño más detalladas. Un conjunto de patrones arquitectónicos permiten que el ingeniero del software reutilice los conceptos a nivel de diseño (BROWN 1998; BUSCHMANN 1996; ERICH GAMMA 1995).

Se puede señalar que independientemente del modelo de diseño que se emplee en el desarrollo de software, se deberán aplicar un conjunto de principios fundamentales y conceptos básicos para el diseño a nivel de componentes, de interfaz, arquitectónico y de datos.

Principios del diseño

El diseño de software es tanto un proceso como un modelo. El *proceso* de diseño es una secuencia de pasos que hacen posible que el diseñador describa todos los aspectos del software que se va construir. Sin embargo, es importante destacar que el proceso de diseño simplemente no es un recetario. Un conocimiento creativo, experiencia en el tema, un sentido de lo que hace que un software sea bueno, y un compromiso general con la calidad son factores críticos de éxito para un diseño competente.

El *modelo* de diseño es el equivalente a los planes de un arquitecto para una casa. Comienza representando la totalidad de todo lo que se va a construir y refina lentamente lo que va a proporcionar la guía para construir cada detalle. De manera similar, el modelo de diseño que se crea para el software proporciona diversas visiones de lo que se quiere construir. Los principios básicos de diseño hacen posible que el ingeniero del software navegue por el proceso de diseño con una mayor garantía de éxito que si se hace de modo improvisado y sin una guía metodológica. Algunos autores sugieren un conjunto de principios para el diseño del software, los cuales han sido adaptados y ampliados en la lista siguiente (DAVIS 1995; PRESSMAN 1998):

- *El proceso de diseño deberá pensarse dinámico.* Un buen diseñador deberá tener en cuenta enfoques alternativos, juzgando todos los que se basan en los requisitos del problema, los recursos disponibles para realizar el trabajo y los conceptos de diseño ya estandarizados.

- *El diseño deberá garantizar una trazabilidad desde el flujo de requerimientos hasta él.* Dado que un solo elemento del modelo de diseño puede o no hacer un seguimiento de los múltiples requisitos, es necesario tener un medio de rastrear como se han satisfecho los requisitos por el modelo de diseño.
- *El diseño no deberá pensarse desde el principio de la reutilización.* Los sistemas se construyen utilizando un conjunto de patrones de diseño, muchos de los cuales probablemente ya se han encontrado antes. Estos patrones deberán elegirse siempre como una alternativa para no reinventar. El tiempo de diseño se deberá invertir en la representación verdadera de ideas nuevas y en la integración de esos patrones que ya existen.
- *El diseño deberá presentar uniformidad e integración.* Un diseño es uniforme si parece que fue una persona la que lo desarrolló por completo. Las reglas de estilo y de formato deberán definirse para un equipo de diseño antes de comenzar el trabajo sobre el diseño. Un diseño se integra si se tiene cuidado a la hora de definir interfaces entre los componentes del diseño.
- *El diseño deberá estructurarse para admitir cambios.* El diseño debe ser lo más flexible posible para soportar posibles cambios.
- *El diseño deberá estructurarse con una estrategia de tratamiento de errores atenuante, incluso cuando se enfrenta con datos, sucesos o condiciones de operación aberrantes debe buscar alternativas de recuperación*. El diseño debe adaptarse a circunstancias inusuales, y si debe terminar de funcionar, que lo haga de forma suave.
- *El diseño deberá evaluarse en función de la calidad y guiados por los requisitos mientras se va creando, no después de terminarlo*. Para ayudar al diseñador en la evaluación de la calidad se dispone de conceptos de diseño, de medidas de diseño y del cumplimiento de los requerimientos implicados.
- *El diseño deberá revisarse para minimizar los errores conceptuales (semánticos).* Un equipo de diseñadores deberá asegurarse de haber afrontado los elementos conceptuales principales antes de preocuparse por la sintaxis del modelo del diseño.

Es importante comprender que no es suficiente con que un programa funcione, sino que es necesario que lo haga correctamente y para ello es imprescindible que se haya realizado un correcto diseño del mismo en función de los requerimientos.

Problemas del diseño y patrones básicos asociados

Conceptos como el de independencia funcional son sumamente importantes para el diseñador de sistemas, múltiples estrategias y técnicas se han implementado en este sentido. La independencia funcional se logra desarrollando módulos con una función "determinante" y una "aversión" a una interacción excesiva con otros módulos (PRESSMAN 1998).

Básicamente se busca obtener un software en el que cada módulo que lo conforma recoja de respuesta a un conjunto de requisitos y a la vez estos tengan una interfaz sencilla para cuando sean observados desde otras partes del programa.

Las ventajas que se tienen al realizar una correcta modularidad, que garantice al mismo tiempo independencia entre los módulos, es que resulta más fácil el proceso de desarrollo y las interfaces se simplifican. Los módulos independientes son más fáciles de mantener y probar porque se limitan los efectos secundarios originados por modificaciones de diseño o código; dado que disminuye la propagación o impacto, del defecto o cambio. El alto grado de independencia funcional logrado en el diseño, garantiza a su vez, un alto grado de reutilización, ya que el acoplamiento de este con otros módulos es mínimo. En resumen, la independencia funcional es la clave para un buen diseño y el diseño es la clave para la calidad del software.

La independencia se mide mediante dos criterios cualitativos: la cohesión y el acoplamiento.

La cohesión

La cohesión es una medida de la fuerza relativa funcional de un módulo. Un módulo cohesivo es aquel que realiza una sola tarea y para ello se relaciona o depende muy poco con otros procedimientos que se llevan a cabo en el resto de la aplicación.

En la perspectiva del diseño orientado a objetos, la cohesión es una medida de cuan relacionadas y enfocadas están las responsabilidades de una clase. Una alta cohesión caracteriza a las clases con responsabilidades estrechamente relacionadas que no realicen un trabajo enorme.

Una clase con baja cohesión hace muchas cosas no afines o un trabajo excesivo. No conviene este tipo de clases pues presentan los siguientes problemas:

- son difíciles de comprender
- son difíciles de reutilizar
- son difíciles de conservar
- son delicadas: las afectan constantemente los cambios

Las clases con baja cohesión a menudo representan un alto grado de abstracción o han asumido responsabilidades que deberían haber delegado a otros objetos (LARMAN 1998).

Se habla que la cohesión se puede representar como un espectro. Siempre se debe buscar la cohesión más alta, sin embargo puede ser aceptada la parte media del espectro. La escala de cohesión no es lineal. Se considera que la parte baja de la cohesión es mucho peor que el rango medio, y este último es casi tan bueno como la parte alta de la escala. En la práctica no hay que preocuparse por categorizar la cohesión en un módulo específico. Más bien, se deberá entender el concepto global, y así se deberán evitar los niveles bajos de cohesión al diseñar. En la parte inferior del espectro, encontraremos un módulo que lleva a cabo un conjunto de tareas que se relacionan con otras débilmente, si es que tienen algo que ver. Tales módulos se denominan coincidentalmente cohesivos (PRESSMAN 1998).

Acoplamiento

El acoplamiento es una medida de la independencia relativa entre los módulos. Pudiese entenderse por acoplamiento la medida de interconexión que existe entre módulos dentro de una estructura de software. El acoplamiento depende de la complejidad de interconexión entre los módulos, el punto donde se realiza una entrada o referencia a un modulo, y los datos que pasan a través de la interfaz.

Considerando el diseño orientado a objetos se puede decir que una clase con alto (o fuerte) acoplamiento recurre a muchas otras. Este tipo de clases no es conveniente, presentan los siguientes problemas (LARMAN 1998):

- Los cambios de las clases afines ocasionan cambios locales.
- Son más difíciles de entender cuando están aisladas.
- Son más difíciles de reutilizar porque se requiere la presencia de otras clases de las que dependen.

Para lograr un buen diseño se deben buscar el acoplamiento más bajo posible.

Como ventajas que muchas veces son desperdiciadas se tiene la facilidad de entender y lograr un software menos propenso a la situación de que un error se produzca en un lugar y este termine propagándose por el sistema.

Siempre que se tenga una lista convencional simple de argumentos (es decir, el paso de datos; la existencia de correspondencia uno a uno entre elementos), se presenta un acoplamiento bajo (Llamado acoplamiento de datos) en esta parte de la estructura. Una variación del acoplamiento de datos, llamado acoplamiento de marca (stamp), se da cuando una parte de la estructura de datos (en vez de argumentos simples) se pasa a través de la interfaz.

En niveles moderados el acoplamiento se caracteriza por el paso de control entre módulos. El acoplamiento de control es muy común en la mayoría de los diseños de software

Cuando los módulos están atados a un entorno externo al software se dan niveles relativamente altos de acoplamiento. Por ejemplo, la E/S acopla un módulo a dispositivos, formatos y protocolos de comunicación. El acoplamiento externo es esencial, pero deberá limitarse a unos pocos módulos en una estructura. También aparece un acoplamiento alto cuando varios módulos hacen referencia a un área global de datos, esto es conocido como el acoplamiento común. El diagnóstico de problemas en estructuras con acoplamiento común es costoso en tiempo y es difícil. Sin embargo, esto no significa necesariamente que el uso de datos globales sea malo. Significa que el diseñador del software deberá ser consciente de las consecuencias posibles del acoplamiento común y tener especial cuidado de prevenirse de ellos (PRESSMAN 1998).

El grado más alto de acoplamiento, acoplamiento de contenido, existe cuando un módulo hace uso de datos o de información de control mantenidos dentro de los límites de otro módulo. En segundo lugar, el acoplamiento de contenido ocurre cuando se realizan bifurcaciones a mitad de módulo. Este modo de acoplamiento puede y deberá evitarse.

Otro conjunto de problemas se originan habitualmente a la hora de realizar el diseño de un software, y un ejemplo de ello es la situación que se puede originar al asignar incorrectamente las responsabilidades.

Un modelo de clase puede definir docenas y hasta cientos de clases de software, y una aplicación tal vez requiera el cumplimiento de cientos o miles de responsabilidades. Durante el diseño orientado a objetos, cuando se definen las interacciones entre los objetos, se toman decisiones sobre la asignación de responsabilidades a las clases. Si se hacen en forma adecuada, los sistemas tienden a ser más fáciles de entender, mantener y ampliar, y se nos presenta la oportunidad de reutilizar los componentes en futuras aplicaciones.

Es entonces que se busca aplicar patrones de diseño como el Experto, el cual plantea que las responsabilidades deberían ser asignadas a quien tenga los recursos para darle solución al problema que pretendemos resolver (LARMAN 1998).

Entre las ventajas de su aplicación en el proceso de diseño se tiene:

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 al hecho de tener sistemas más robustos y de fácil manteniendo.

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 más fáciles de comprender y de mantener. Así se brinda soporte a una alta cohesión.

En otro sentido se puede señalar que la creación de objetos es una de las actividades más frecuentes en un sistema orientado a objetos. En consecuencia, conviene contar con un principio general para asignar las responsabilidades concernientes a ella.

El patrón Creador guía la asignación de responsabilidades relacionadas con la creación de objetos. El propósito fundamental de este patrón es encontrar un creador que debemos conectar con el objeto producido en cualquier evento. Entre las ventajas de la aplicación de dicho patrón encontramos que (LARMAN 1998):

Brinda soporte a un 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.

A la hora de concebir un buen diseño surgen preguntas como: ¿Quién debería encargarse de atender un evento del sistema?

La mayor parte de los sistemas reciben eventos de entrada externa, los cuales generalmente incluyen una interfaz gráfica para el usuario operado por una persona. Otros medios de entrada son los mensajes externos 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. El patrón Controlador ofrece una guía para tomar decisiones apropiadas que generalmente se aceptan.

Entre los beneficios de la aplicación de dicho patrón se tiene (LARMAN 1998):

Mayor potencial de los componentes reutilizables. Garantiza que la empresa o los procesos de dominio sean manejados por la capa de los objetos del dominio y no por la de la interfaz. Desde el punto de vista técnico, las responsabilidades del controlador podrían cumplirse en un objeto de interfaz, pero esto supone que el código del programa y la lógica relacionada con la realización de los procesos del dominio puro quedarían incrustados en los objetos interfaz o ventana. Un diseño de interfaz-como-controlador reduce la posibilidad de reutilizar la lógica de los procesos del dominio en aplicaciones futuras, por estar ligada a una interfaz determinada que rara vez puede utilizarse en otras aplicaciones. En cambio, el hecho

de delegar a un controlador la responsabilidad de la operación de un sistema entre las clases del dominio soporta la reutilización de la lógica para manejar los procesos afines del negocio en aplicaciones futuras.

A veces es necesario asegurarse de que las operaciones del sistema sigan una secuencia legal o poder razonar sobre el estado actual de la actividad y las operaciones en el caso de uso subyacente. De ser así, esta información sobre el estado ha de capturarse en alguna parte; el controlador es una buena opción, sobre todo si se emplea a lo largo de todo el caso.

Otro aspecto a tener en cuenta en el diseño de sistemas son las aplicaciones Web. Estas con un predominio mayoritario en Internet, y con una tendencia acelerada de imponerse como tecnología en las soluciones informáticas hoy acarrean un conjunto de problemáticas relacionadas directamente con las características del diseño Web, tales como la taxonomía de navegación, el etiquetado de la interfaz y la jerarquía de organización de la información o arquitectura de la información. Algunos de estos problemas se tratan de agrupar en la siguiente lista (NIELSEN 2005):

1. *Problemas de legibilidad:* Problemas de legibilidad derivados del uso de tipografías no adecuadas, cuerpos pequeños, falta evidente de contraste con el color de fondo.
2. *Enlaces alejados de su formato estándar:* Deben hacerse obvios los enlaces, hay que diferenciar los links visitados de los no visitados, usar textos descriptivos del enlace que contienen, no abrir enlaces en nuevas ventanas.
3. *Flash, Apples, Active X:* Hay un uso de esta tecnología que se debe encauzar hacia objetivos más acordes con las potencialidades de ésta. Úsese para hacer lo que el html no puede hacer, en lugar de para dar más alegría a los sitios Web.
4. *Contenido no escrito para la Web:* Hay contenido que se pública en la Web de forma poco adecuada para el medio en que se está. Los textos han de ser cortos, concisos, que se puedan identificar con un golpe de vista, y vayan al grano.
5. *Búsquedas deficientes:* Las búsquedas son uno de los elementos fundamentales de un sitio Web. Hacerlo bien es complejo y es uno de los factores principales de una experiencia de usuario positiva. Problemas con los criterios de búsqueda y con el tiempo de respuesta de las mismas, así como el rendimiento de estas debido a la falta de paginación, saturación de las cokkies y cache son aspectos que deben tenerse en cuenta a la hora de diseñar aplicaciones Web.
6. *Incompatibilidades entre navegadores:* En el desarrollo de software basado en aplicaciones Web es necesario invertir tiempo del proyecto para compatibilizar el código y éste sea multinavegador, el

uso de navegadores diferentes a Internet Explorer hoy día hace que se deba volver atrás y no se excluyan usuarios sólo porque usen una plataforma diferente, el html dinámico o las nuevas tecnologías emergentes que obvian la compatibilidad como asp.net 2.0 generan inconvenientes en usuarios de navegadores distintos al Internet Explorer como Mozilla Firefox, Netscape entre otros.

7. *Formularios incómodos:* Se identifican muchos problemas relacionados con la complejidad y uso de los formularios. Éstos se usan con mucha frecuencia en la Web y muchos de ellos son excesivamente largos y complejos de rellenar (usar) por parte de los usuarios. Es por ello que la selección de una correcta taxonomía de navegación facilita la calidad del diseño de los formularios. Es importante para dar solución a este problema del diseño de sistemas el estudio de patrones de workflow.
8. *Ausencia de vías de contacto con los responsables del sitio Web:* Uno de los signos de credibilidad y por tanto de confianza que se puede dar al usuario es mostrar una dirección (postal) física de contacto. Es fácil pensar que una empresa de la que no se ofrece la dirección de su ubicación difícilmente puede recibir pagos de sus clientes por una falta de confianza de éstos en aquélla.
9. *Maquetación con ancho fijo:* Con las desventajas que esto acarrea, tanto si tenemos un monitor demasiado grande y no podemos leer bien los textos si no aumentamos su tamaño, como si nuestro monitor, o resolución, es demasiado pequeño, que nos encontramos con un desagradable scroll horizontal.
10. *Ampliación inadecuada de las imágenes:* Este puesto en realidad estaba reservado para desaconsejar de nuevo el uso de pop-ups, pero Nielsen lo deja en la costumbre que algunos tienen de mostrar la misma foto tanto cuando se trata de un thumbnail como de la imagen ampliada.

Puntos de consideración a la hora de hacer un buen diseño Web.

- Que el texto sea legible
- Que los contenidos respondan a las expectativas
- Que los sistemas de navegación y búsqueda les ayuden a encontrar lo que se busca
- Formularios más cortos y simples
- Que no haya cosas que no funcionen, enlaces que no lleven a ninguna parte, contenido desactualizado

Arquitectura. Estilos arquitectónicos. Un vértice de contacto con el diseño de sistemas

Teniendo en cuenta el impacto que tiene el diseño arquitectónico sobre el diseño del software, y lo costoso que resulta una modificación de la línea base de la arquitectura debido a errores realizados a la hora de definir la arquitectura del sistema, se ha dedicado este epígrafe para ver la relación existente entre el diseño arquitectónico, el diseñó de software y la codificación del mismo, además de hacer una referencia general del estado del arte de los estilos arquitectónicos, sin llegar hacer un análisis profundo del tema, ya que no es objetivo de este trabajo los estilos arquitectónicos, ni los temas relacionados con arquitectura de los sistemas informáticos, si desea encontrar una información más detalladas a lo relacionado en este epígrafe consulte (REYNOSO marzo 2004).

Existen varias escuelas asociadas a la arquitectura y el diseño arquitectónico del software, si bien algunas hablan de cierta relación con el diseño mientras otras no, lo cierto es que todas estas escuelas coinciden en que el diseño arquitectónico, o de estilos arquitectónico, es una fase superior de abstracción al flujo de diseño, es la parte macro arquitectónica del software, mientras que los patrones de diseño y el modelo el paradigma de desarrollo orientado a objetos representa la parte micro arquitectónica del diseño de software.

Entre los diferentes puntos de vista del diseño arquitectónico se encuentran los que referencian la arquitectura como una **Etapa de ingeniería y diseño orientada a objetos.** Este es el modelo de James Rumbaugh, Ivar Jacobson, Grady Booch, Craig Larman y otros, ligado estrechamente al mundo de Unified Modeling Language (UML) y Rational. Esta es una corriente apostada por: Rumbaugh, Jacobson y Booch quienes han sido llamados "Los Tres Amigos"; tal ves relacionado este nombre por las posturas comunes entres ellos sobre el desarrollo del software y la arquitectura. En esta postura, la arquitectura se restringe a las fases iniciales y preliminares del proceso y concierne a los niveles más elevados de abstracción, pero no está sistemáticamente ligada al requerimiento del software o a la composición del diseño que viene después. Sobre esta arista a cualquier configuración, topología o morfología de las piezas del sistema se la llama arquitectura. En esta escuela, si bien se reconoce el valor primordial de la abstracción y del ocultamiento de información promovido por Parnas, estos conceptos tienen que ver más con el encapsulamiento en paquetes, clases y objetos que con la visión de conjunto arquitectónica. Para este movimiento, la arquitectura se confunde también con el modelado y el diseño, los cuales constituyen los conceptos dominantes. En esta corriente se manifiesta predilección por un modelado denso y una

profusión de diagramas, tendiente al modelo metodológico CMM(Modelo de Capacidad y Madurez[2]) o a UPM (Universidad Politécnica de Madrid); no hay sin embargo una prescripción formal del tema; para estos importa más la abundancia y el detalle de diagramas y técnicas disponibles que la simplicidad de la visión del conjunto. Cuando se habla en esta escuela de estilos, se les confunde con patrones arquitectónicos o de diseño (LARMAN 1998), casi nunca se hace referencia a los lenguajes de descripción arquitectónica, que representan uno de los assets reconocidos de la arquitectura de software; sucede como si la disponibilidad de un lenguaje unificado de modelado los tornara superfluos. La definición de arquitectura que se promueve en esta corriente tiene que ver con aspectos formales a la hora del desarrollo; esta arquitectura es isomorfa a la estructura de las piezas de código. Una definición típica y demostrativa sería la de Grady Booch; para él, la arquitectura de software es "la estructura lógica y física de un sistema, forjada por todas las decisiones estratégicas y tácticas que se aplican durante el desarrollo" (BOOCH 1991). Otras definiciones revelan que la arquitectura de software, en esta perspectiva, concierne a decisiones sobre organización, selección de elementos estructurales, comportamiento, composición y estilo arquitectónico susceptibles de ser descritas a través de las cinco vistas clásicas del modelo 4+1 de RUP (Rational Unified Process) (JACOBSON *et al.* 2000) y de Kruchten (KRUCHTEN Noviembre de 1995). Otro punto de vista sobre el diseño arquitectónico esta dado por una **arquitectura estructural, basada en un modelo estático de estilos, Lenguajes de descripción arquitectónica (ADLs) y vistas.** Este criterio constituye la corriente fundacional y clásica de la disciplina. Los representantes de esta corriente son en su mayoría académicos, mayormente de la Universidad Carnegie Mellon en Pittsburgh: Mary Shaw, Paul Clements, David Garlan, Robert Allen, Gregory Abowd, John Ockerbloom. Se trata también de la visión de la arquitectura de software dominante en la academia, y aunque es la que ha hecho el esfuerzo más importante por el reconocimiento de la arquitectura de software como disciplina, sus categorías y herramientas son todavía mal conocidas en la práctica industrial. En el interior del movimiento se pueden observar distintas divisiones. Hay una variante informal de "boxología" y una vertiente más formalista, representada por el grupo de Mark Moriconi en Menlo Park. En principio se pueden reconocer tres modalidades en cuanto a la formalización; los más informales utilizan descripciones verbales o diagramas de cajas, los de criterio intermedio se sirven de Lenguajes

2 Modelo de Capacidad y Madurez**(CMM)** es un modelo de calidad que aplica a los conceptos de process management y quality improvement en el desarrollo y mantenimiento de software, y describe 5 niveles a través de los cuales las organizaciones evolucionan, a medida que definen, implementan, miden, controlan y mejoran sus procesos de construcción de aplicaciones de software. Es un modelo basado en la Calidad Total y en la mejora continua.

descriptivos de modelado [3](ADL) y los más exigentes usan lenguajes formales de especificación. En toda la corriente, el diseño arquitectónico no sólo es el de más alto nivel de abstracción, sino que además no tiene por qué coincidir con la configuración explícita de las aplicaciones; rara vez se encontrarán referencias a lenguajes de programación o piezas de código, y en general nadie habla de clases o de objetos. Este nivel de estructuración de las temáticas arquitectónicas nunca son prudentes y lo increíble del punto es que aun hoy en la actualidad después de más de 30 años de desarrollo de la industria del software, no está muy claro el tema de la posición del modelado arquitectónico en el ciclo de vida, ni su relación explicita o implícita con el diseño del sistema. De todas las corrientes teóricas sobre el diseño arquitectónico y su vinculación más directa con el diseño de software esta la propuesta por la **Arquitectura basada en patrones.** Si bien reconoce la importancia de un modelo emanado históricamente del diseño orientado a objetos, esta corriente surgida hacia 1996 no se encuentra tan rígidamente vinculada a UML en el modelado, ni a CMM en la metodología. El texto sobre patrones que esta variante reconoce como referencia es la serie POSA de Buschmann y otros (BUSCHMANN 1996) y secundariamente el texto de la Banda de los Cuatro (ERICH GAMMA 1995). La diferencia entre ambos textos sagrados de la comunidad de patrones no es menor; en el primero, la expresión "Software Architecture" figura en el mismo título; el segundo se llama Design Patterns: Elements of reusable Object-Oriented software y su tratamiento de la arquitectura es mínimo. En esta manifestación de la arquitectura de software prevalece cierta tolerancia hacia modelos de procesos tácticos, no tan macroscópicos, y eventualmente se expresa cierta simpatía por las ideas de Martin Fowler y las premisas de la programación extrema. El diseño consiste en identificar y articular patrones preexistentes, que se definen en forma parecida a los estilos de arquitectura (SHAW, MARY 1984).

Cada una de las corrientes teóricas relacionadas anteriormente coincide en que existe una marcada diferencia entre el diseño de software y el diseño arquitectónico. La comunidad de arquitectura de software, en particular la académica, sostiene que ésta difiere sustancialmente del diseño. Pero Taylor y Medvidovic por ejemplo, señalan que la literatura actual mantiene en un estado ambiguo la relación entre ambos campos, albergando diferentes interpretaciones y posturas (RICHARD TAYLOR):

Una postura afirma que arquitectura y diseño son lo mismo.

---

3 **ADL**. Significa lenguaje descriptivo de modelado que se focaliza en la estructura de alto nivel de la aplicación. Los ADLs 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.

Otra, en cambio, alega que la arquitectura se encuentra en un nivel de abstracción por encima del diseño, o es simplemente otro paso (un artefacto) en el proceso de desarrollo de software.

Una tercera establece que la arquitectura es algo nuevo y en alguna medida diferente del diseño (pero de qué manera y en qué medida se dejan sin especificar).

Al igual que Taylor y Medvidovic los autores de este trabajo estiman que la segunda interpretación es la que se encuentra más cerca de la verdad. En alguna medida, la arquitectura y el diseño sirven al mismo propósito. Sin embargo, el foco de la arquitectura de software en la estructura del sistema y en las interconexiones la distingue del diseño de software tradicional, tales como el diseño orientado a objetos, que se concentra más en el modelado de abstracciones de más bajo nivel, tales como algoritmos y tipos de datos. A medida que la arquitectura de alto nivel se refina, sus detalles pierden el guión del diseño estructuralmente, distribuyéndose a través de los elementos arquitectónicos de más bajo nivel, pero su materialización radica en la transformación de la arquitectura en diseño, y posteriormente esta en código implementado, listo para la probar.

En su reciente libro sobre el arte de la arquitectura de software, Stephen Albin (ALBIN 2003) se pregunta en qué difiere ella de las metodologías de diseño bien conocidas como la orientación a objetos. Ante esta interrogante se puede plantear que la arquitectura de software, es una metáfora relativamente nueva en materia de diseño de software y en realidad abarca también las metodologías de diseño, así como metodologías de análisis. El concepto de arquitectura intenta introducir las actividades de análisis y diseño en un framework de diseño más amplio y más coherente. Las organizaciones se están dando cuenta que el alto costo del desarrollo de software requiere ser sometido a algún control y que muchas de las ventajas prometidas por las metodologías aún no se han materializado. Pero la arquitectura es algo más integrado que la suma del análisis por un lado y el diseño por el otro. La integración de metodologías y modelos, concluye Albin, es lo que distingue la arquitectura de software de la simple yuxtaposición de técnicas de análisis y de diseño.

En una presentación de 1997, Dewayne Perry, uno de los fundadores de la disciplina, bosquejó la diferencia entre arquitectura y diseño. La arquitectura, una vez más concierne a un nivel de abstracción más elevado; se ocupa de componentes y no de procedimientos; de las interacciones entre esos componentes y no de las interfaces; de las restricciones a ejercer sobre los componentes y las interacciones y no de los algoritmos, los procedimientos y los tipos. En cuanto a la composición, la de la arquitectura es de grano grueso, la del diseño es de fina composición procedural; las interacciones entre

componentes en arquitectura tienen que ver con un protocolo de alto nivel (en el sentido no técnico de la palabra), mientras que las del diseño conciernen a interacciones de tipo procedural (mensajes, llamadas a rutinas) (PERRY 1997).

Desde el punto de vista práctico, el homólogo de los patrones de diseño de software, pero de mayor abstracción son los estilos arquitectónicos, si bien no es objetivo de este trabajo hacer un análisis detallados de estos, a continuación hacemos una referencia bastante general sobre el tema, por la relación que tienen los estilos arquitectónicos con la parte macro del diseño de software de una aplicación, y el impacto del mismo en la calidad y la reutilización de los diseños, si se desea profundizar más en el tema se recomienda ver (REYNOSO Junio de 2004) . Entre ellos están los **estilos de Flujo de Datos.** Esta familia de estilos enfatiza la reutilización y la modificabilidad. Es apropiada para sistemas que implementan transformaciones de datos en pasos sucesivos. Ejemplares de la misma serían las arquitecturas de tubería-filtros y las de proceso secuencial en lote. En el caso de **tubería y filtros** siempre es encuadrado dentro de las llamadas arquitecturas de flujo de datos. Es sin duda alguna el estilo que se definió más temprano (GARLAN, ROBERT ALLEN Y DAVID Setiembre de 1992) y el que puede identificarse topológica, procedural y de menor ambigüedad taxonómica. Históricamente el se relaciona con las redes de proceso descriptas por Kahn hacia 1974 y con los proceso secuenciales comunicantes (CSP) ideados por Tony Hoare cuatro años más tarde. Ha prevalecido el nombre de tubería-filtros, pero su nombre no esta relacionado por tareas de filtrado, como eliminación de campos o registros, sino que ejecutan formas variables de transformación, una de las cuales puede ser el filtrado.

Una tubería (pipeline) es una popular arquitectura que conecta componentes computacionales (filtros) a través de conectores (pipes), de modo que las computaciones se ejecutan a la manera de un flujo. Los datos se transportan a través de las tuberías entre los filtros, transformando gradualmente las entradas en salidas. Debido a su simplicidad y su facilidad para captar una funcionalidad, es la arquitectura modelo cada vez que se trata de demostrar ideas sobre la formalización del espacio de diseño arquitectónico, igual que el tipo de datos stack (Pila) lo fue en las especificaciones algebraicas o en los tipos de datos abstractos.

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. En el estilo secuencial por lotes (batch sequential) los componentes son programas independientes; el supuesto es que cada paso se ejecuta hasta completarse antes que se inicie el paso siguiente. Garlan y Shaw

sostienen que la variante por lotes es un caso degenerado del estilo, en el cual las tuberías se han vuelto residuales (GARLAN, MARY SHAW Y DAVID 1996) .

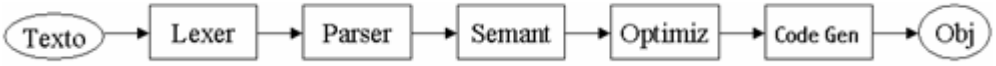

Figura 1Compilador en tubería-filtro

Varios autores establecen que este estilo podría ser usado cuando (REYNOSO Junio de 2004):

- Se puede especificar la secuencia de un número conocido de pasos.
- No se requiere esperar la respuesta asincrónica de cada paso.
- Se busca que todos los componentes situados corriente abajo sean capaces de inspeccionar y actuar sobre los datos que vienen de corriente arriba (pero no viceversa).

Otra familia de estilos arquitectónicos son los estilos **Centrados en Datos.** Esta familia de estilos enfatiza la integración de los datos. Se estima apropiada para sistemas que se fundan en acceso y actualización de datos en estructuras de almacenamiento. Subestilos característicos de la familia serían los repositorios, las bases de datos, las arquitecturas basadas en hipertextos y las arquitecturas de pizarra. En esta arquitectura hay dos componentes principales: una estructura de datos que representa el estado actual y una colección de componentes independientes que operan sobre él. En base a esta distinción se han definidos dos subcategorías principales del estilo (REYNOSO Junio de 2004):

- Si los tipos de transacciones en el flujo de entrada definen los procesos a ejecutar, el repositorio puede ser una base de datos tradicional (implícitamente no cliente-servidor).
- Si el estado actual de la estructura de datos dispara los procesos a ejecutar, el repositorio es lo que se llama una pizarra pura o un tablero de control.

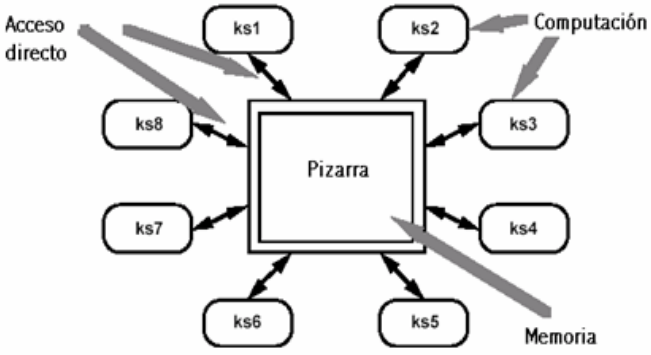

Figura 2 Modelo estilo Pizarra

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. También se han implementado estilos de este tipo en procesos en lotes de base de datos y ambientes de programación organizados como colecciones de herramientas en torno a un repositorio común.

El documento clásico que describe el estilo, Blackboard Systems, de H. Penny Nii (NII), bien conocido en Inteligencia Artificial. Los estilos de pizarra se les utilizan en exploraciones recientes de Inteligencia Artificial distribuida o cooperativa, en robótica, en modelos multi-agentes, en programación evolutiva, en gramáticas complejas, en modelos de crecimiento afines a los L-Systems de Lindenmayer, etc. Un sistema de pizarra se implementa para resolver problemas en los cuales las entidades individuales se manifiestan incapaces de aproximarse a una solución, o para los que no existe una solución analítica, o para los que sí existen pero es inviable por la dimensión del espacio de búsqueda. Todo modelo de este tipo consiste en las siguientes tres partes:

1. Fuentes de conocimiento, necesarias para resolver el problema.
2. Una pizarra que representa el estado actual de la resolución del problema.
3. Una estrategia, que regula el orden en que operan las fuentes.

Al comienzo del proceso de resolución, se establece el problema en la pizarra. Las fuentes tratan de resolverlo cambiando el estado. La única forma en que se comunican entre sí es a través de la pizarra. Finalmente, si de la cooperación resulta una solución adecuada, ésta aparece en la pizarra como paso final.

En el caso de los estilos de tipo **Llamada y Retorno** se enfatiza la modificabilidad y la escalabilidad. Son los estilos más generalizados en sistemas en gran escala. Miembros de la familia son las arquitecturas de programa principal y subrutina, los sistemas basados en llamadas a procedimientos remotos, los sistemas orientados a objeto y los sistemas jerárquicos en capas, un ejemplo de estos puede ser el también conocido como patrón Modelo Vista Controlador ver (ERICH GAMMA 1995) si desea profundizar más en este patrón o estilo arquitectónico.

Un estilo que por su difusión y su generalización en el mundo del desarrollo de software actual se hace de obligatoria referencia es el llamado, estilo basado en **Arquitecturas en Capas.** Los sistemas o arquitecturas en capas constituyen uno de los estilos que aparecen con mayor frecuencia mencionados como categorías mayores del catálogo, o por el contrario, como una de las posibles encarnaciones de

algún estilo más envolvente. En (SHAW, DAVID GARLAN Y MARY 1994) Garlan y Shaw definen el estilo en capas como una organización jerárquica tal que cada capa proporciona servicios a la capa inmediatamente superior y se sirve de las prestaciones que le brinda la inmediatamente inferior. En algunos ejemplares, las capas internas están ocultas a todas las demás, menos para las capas externas adyacentes, y excepto para funciones puntuales de exportación; en estos sistemas, los componentes implementan máquinas virtuales en alguna de las capas de la jerarquía. En otros sistemas, las capas pueden ser sólo parcialmente opacas. En la práctica, las capas suelen ser entidades complejas, compuestas de varios paquetes o subsistemas. El uso de arquitecturas en capas, explícitas o implícitas, es frecuentísimo; solamente en Pattern Almanac 2000 (RISING 2000) hay cerca de cien patrones que son variantes del patrón básico de capas. Patrones de uso común relativos al estilo son Façade, Adapter, Bridge y Strategy (ERICH GAMMA 1995).

En un estilo en capas, los conectores se definen mediante los protocolos que determinan las formas de la interacción. Los diagramas de sistemas clásicos en capas dibujaban las capas en adyacencia, sin conectores, flechas ni interfaces; en algunos casos se suele representar la naturaleza jerárquica del sistema en forma de círculos concéntricos. Las restricciones topológicas del estilo pueden incluir una limitación, más o menos rigurosa, que exige a cada capa operar sólo con capas adyacentes, y a los elementos de una capa entenderse sólo con otros elementos de la misma; se supone que si esta exigencia se relaja, el estilo deja de ser puro y pierde algo de su capacidad heurística (MICROSOFT 2004); también se pierde, naturalmente, la posibilidad de reemplazar de cuajo una capa sin afectar a las restantes del conjunto, disminuye la flexibilidad, aumentaría el acoplamiento, y el mantenimiento sería mucho más costoso. Las formas más rígidas no admiten ni siquiera pass-through: cada capa debe hacer algo, siempre. En la literatura especializada hay multitud de argumentos a favor y en contra del rigor de esta clase de prescripciones. A veces se argumenta que el cruce superfluo de muchos niveles involucra eventuales degradaciones de performance; pero muchas más veces se sacrifica la pureza de la arquitectura en capas precisamente para mejorarla: colocando, por ejemplo, reglas de negocios en los procedimientos almacenados de las bases de datos, o articulando instrucciones de consulta en la capa de la interface del usuario.

Casos representativos de este estilo son muchos de los protocolos de comunicación en capas. En ellos cada capa proporciona un sustrato para la comunicación a algún nivel de abstracción, y los niveles más

bajos suelen estar asociados con conexiones de hardware. El ejemplo más característico es el modelo OSI con los siete niveles: nivel físico, vínculo de datos, red, transporte, sesión, presentación y aplicación.

El número mínimo de capas es obviamente dos, y en ese umbral la literatura arquitectónica sitúa a veces al subestilo cliente-servidor como el modelo arquetípico del estilo de capas y el que se encuentra con mayor frecuencia en las aplicaciones en red (REYNOSO Junio de 2004).

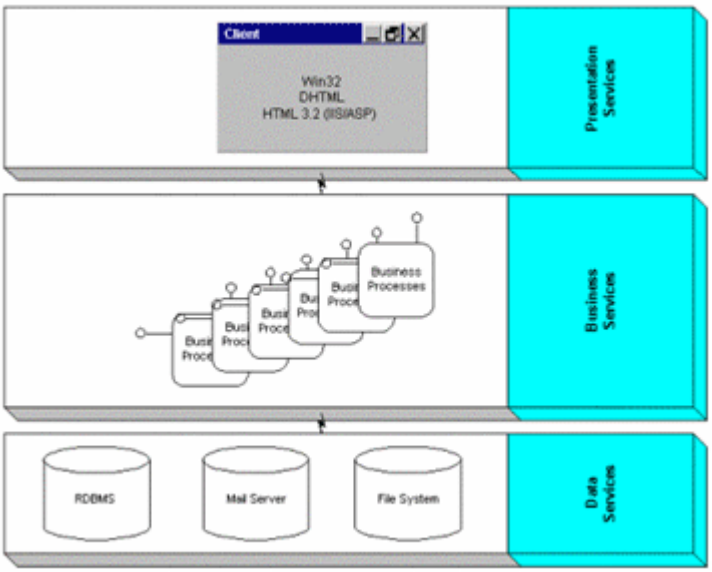

Figura 3 Arquitectura en 3 capas

Existen otros estilos arquitectónicos que tienen mucho más implicación en el diseño de software abordando este desde el punto micro, el procedural, que esta más bien relacionado con los procedimientos, los mensajes, el manejo de los datos, etc., uno de ellos es el estilo arquitectónico basado en **Arquitecturas Orientadas a Objetos.** Nombres alternativos para este estilo han sido Arquitecturas Basadas en Objetos, Abstracción de Datos y Organización Orientada a Objetos. Los componentes de este estilo son los objetos, o más bien instancias de los tipos de dato abstractos. Un rasgo importante de este aspecto es que la representación interna de un objeto no es accesible desde otros objetos. Un resumen de las características principales de las arquitecturas OO, podría ser (REYNOSO Junio de 2004):

Los componentes del estilo se basan en principios OO: 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 general la distribución de objetos es transparente, y en el estado de arte de la tecnología (lo mismo que para los componentes en el sentido de CBSE) apenas importa si los objetos son locales o remotos. El mejor ejemplo de OO para sistemas distribuidos es Common Object Request Broker Architecture (CORBA), en la cual las interfaces se definen mediante Interface Description Language (IDL); un Object Request Broker media las interacciones entre objetos clientes y objetos servidores en ambientes distribuidos.

En cuanto a las restricciones, puede admitirse o no que una interfaz pueda ser implementada por múltiples clases.

En tanto, los objetos interactúan a través de invocaciones de funciones y procedimientos. Hay muchas variantes del estilo; algunos sistemas, por ejemplo, admiten que los objetos sean tareas concurrentes; otros permiten que los objetos posean múltiples interfaces.

El otro estilo es el basado en **Arquitecturas Basadas en Componentes.** En un principio, hacia 1994, se planteaba como una modalidad que extendía o superaba la tecnología de objetos, como en un famoso artículo de BYTE cuyo encabezado rezaba así (REYNOSO Junio de 2004): "ComponentWare – La computación Orientada a Objetos ha fracasado. Pero el software de componentes, como los controles de Visual Basic, está teniendo éxito". Con el paso de los años el antagonismo se fue aplacando y las herramientas (orientadas a objeto o no) fueron adaptadas para producir componentes. En la mayoría de los casos, los componentes terminan siendo formas especiales de *Bibliotecas de Enlace Dinámico* [4] (DLL), que necesitan registración y que no requieren que sea expuesto el código fuente de la clase (SZYPERSKI Diciembre de 1995).

Hay un buen número de definiciones de componentes, pero por su nivel de concreción se propone, unidad de composición con interfaces especificadas contractualmente y dependencias del contexto explícitas (SZYPERSKI 2002).

Este tipo de arquitectura permite la paralización en el desarrollo, da un nuevo paradigma al desarrollo del software, pues tributa a la tendencia del ensamblado de sistemas más que a la construcción de los mimos, lo que posibilita minimizar el tiempo de desarrollo. Permite a los equipos de desarrollo crear una base tecnológica especializada según el tipo de aplicación que los mismos desarrollen, propiciando mayor

4 DLL es el acrónimo de Dynamic Linking Library (Bibliotecas de Enlace Dinámico), término con el que se refiere a los archivos con código ejecutable que se cargan bajo demanda del programa por parte del sistema operativo. Esta denominación se refiere a los sistemas operativos Windows siendo la extensión con la que se identifican los ficheros, aunque el concepto existe en prácticamente todos los sistemas operativos modernos.

escalabilidad, flexibilidad y rapidez de desarrollo a sus diseños. Por tanto sistemas que asumen menos riesgos para ser adaptados y que se ajustan en mejor medida al cumplimiento de costes estimados y contratados con los clientes.

Estándares de diseño para aplicaciones médicas. HL7

Hace unos 30 años empresas de transportes y envío de mercancías comenzaron a desarrollar un intercambio de datos informático que disminuía la complejidad del proceso y los retrasos del papeleo de ciertas transacciones mercantiles como facturas, órdenes de compra, entre otros, logrando asimismo, reducir costes. Así nació el Electronic Data Interchange (EDI) que permite intercambiar información entre empresas a través de un formato específico común, haciendo innecesaria la intervención humana, ya que las operaciones se llevan a cabo íntegramente a través de ordenadores. Con la generalización de Internet, EDI cobra de nuevo protagonismo en el escenario del business to business (B2B) (WIKIPEDIA 2007b).

El desarrollo de estándares y la generalización de la informática ha impulsado el uso de EDI en muchos otros sectores como pueden ser: seguros de atención médica, archivos, servicios financieros, compras gubernamentales y transacciones en Internet.

En la actualidad existen varios cientos de estándares para una amplia gama de transacciones B2B. El principal es el X12**,** desarrollado por el Accredited Standards Committee (ASC X12).

Asimismo se puede mencionar el Estándar internacional del Intercambio Electrónico de Datos para la Administración, Comercio y Transporte (EDIFACT[5]), el cual es un estándar de la ONU para el intercambio de documentos comerciales en el ámbito mundial. Existiendo subestándares para cada entorno de negocio relacionado sectores como la distribución, la automoción, el transporte, la aduana entre otros (ALAMEDA 2006).

El XML ha sido abundantemente utilizado en estos tiempos y conocido. XML no ha nacido sólo para su aplicación en Internet, sino que se propone como un estándar para el intercambio de información estructurada entre diferentes plataformas. Se puede usar en bases de datos, editores de texto, hojas de cálculo y casi cualquier cosa imaginable.

[5] Electronic Data Interchange For Administration, Commerce and Transport. Estándar internacional del Intercambio Electrónico de Datos para la Administración, Comercio y Transporte. Es un EDI con una sintaxis normalizada que haga inteligible las transacciones comerciales entre empresas.

XML, sigla en inglés de e**X**tensible **M**arkup **L**anguage (lenguaje de marcas extensible), es un metalenguaje extensible de etiquetas desarrollado por el World Wide Web Consortium (W3C). Es una simplificación y adaptación del SGML y permite definir la gramática de lenguajes específicos (de la misma manera que HTML es a su vez un lenguaje definido por SGML). Por lo tanto XML no es realmente un lenguaje en particular, sino una manera de definir lenguajes para diferentes necesidades. Algunos de estos lenguajes que usan XML para su definición son XHTML, SVG, MathML.

XML es una tecnología sencilla que tiene a su alrededor otras que la complementan y la hacen mucho más grande y con unas posibilidades mucho mayores. Tiene un papel muy importante en la actualidad ya que permite la compatibilidad entre sistemas para compartir la información de una manera segura, fiable y fácil (WIKIPEDIA 2007c).

En materia de estándares para aplicaciones médicas tenemos algunos que están siendo frecuentemente utilizados como es el caso de Intenational Clasification of diseases (ICD 9 y ICD 10), Logical Observation Identifiers Names and Codes (LOINC), Digital Imaging and Communication in Medicine (DICOM) usado en operaciones relacionadas con imágenes, el caso de Health Level 7 (HL7) con especial énfasis en las comunicaciones intrahospitalarias y el estándar Agrupación de Empresas de Sistemas de Farmacias (ADESFA) usado en operaciones relacionadas con sistemas de farmacias.

Los estándares mencionado se han debido a un conjunto de factores que relacionados a continuación (MARCH 2006):

- Explosión de costos en las áreas de salud y tecnología
- Peligro de parálisis tecnológica (red de comunicaciones ociosa, uso subóptimo de parque tecnológico instalado)
- Proliferación innecesaria de productos mínimamente diferenciados (medicamentos, nomencladores)
- Excesiva variedad (nomencladores, medicamentos, prácticas y procedimientos, dispositivos)
- Insuficiente colaboración entre sectores (consiguiente multiplicación de esfuerzos de desarrollo y amenaza de excesiva variedad)
- Falta de foco en desarrollo de dispositivos: Rangos extremos desde tecnologías de utilidad relativa (WAP, Portales, burbuja de Internet en general) hasta tecnologías de gran utilidad potencial (mensajería electrónica *ad-hoc*, informática clínica).

Descripción de los estándares médicos.

Se puede mencionar a estándares de codificación como la Intenational Clasification of diseases (ICD9 y ICD10) de la World haelth Oranization (WHO) o lo que es lo mismo de la Organización Mundial de la Salud (BIOCOM 2006b).

Logical Observation Identifiers Names and Codes (LOINC)

LOINC es un estándar que provee un conjunto de códigos, nombres y sinónimos fundamentalmente orientado a resolver situaciones clínicas y pruebas de laboratorio de análisis clínicos. La necesidad de contar con un lenguaje universal y controlado para intercambio de información en los sistemas de laboratorio de análisis clínicos, que permita una correcta y consistente representación de los conceptos médicos (resultados), intercambio en tiempo real, base para soporte en la toma de decisiones, estadística de los datos para medir resultados, calidad e investigación llevó la creación del LOINC. Actualmente, LOINC emerge como el estándar más difundido para definir los nombres de las determinaciones de laboratorio de análisis clínicos. Esto es de gran importancia porque facilita el intercambio de resultados clínicos entre distintos sistemas y bases de información de resultados (BIOCOM 2006c).

Digital Imaging and Communication in Medicine (DICOM)

DICOM es el estándar reconocido mundialmente para el intercambio de imágenes médicas, pensado para el manejo, almacenamiento, impresión y transmisión de imágenes médicas. Incluye la definición de un formato de fichero y de un protocolo de comunicación de red. El protocolo de comunicación es un protocolo de aplicación que usa TCP/IP para la comunicación entre sistemas. Los ficheros DICOM pueden intercambiarse entre dos entidades que tengan capacidad de recibir imágenes y datos de pacientes en formato DICOM.

DICOM permite la integración de escáneres, servidores, estaciones de trabajo, impresoras y hardware de red de múltiples proveedores dentro de un sistema de almacenamiento y comunicación de imágenes. Las diferentes máquinas, servidores y estaciones de trabajo tienen una declaración de conformidad DICOM (conformance statements) que establece claramente las clases DICOM que soportan. DICOM ha sido adoptado ampliamente por hospitales y está haciendo incursión en pequeñas aplicación de oficinas de dentistas y de doctores (WIKIPEDIA 2007a).

Los ficheros DICOM consisten en una cabecera con campos estandarizados y de forma libre, y un cuerpo con datos de imagen. Un objeto DICOM simple puede contener solamente una imagen, pero esta

imagen puede tener múltiples "fotogramas" (frames), permitiendo el almacenamiento de bloques de cine o cualquier otros datos con varios fotogramas. Los datos de imagen puede estar comprimidos usando gran variedad de estándares, incluidos JPEG, JPEG, Lossless, JPEG 2000, LZW y Run-length encoding (RLE).

DICOM tiene un conjunto muy amplio en servicios, la mayoría de los cuales implica transmisión de datos sobre la red, y el formato de fichero en que se sustenta es en realidad una ampliación posterior y de menor importancia del estándar.

Algunos de ellos son:

El servicio DICOM Store es usado para mandar imágenes u otros objetos persistentes (informes estructurados, etc.) a un PACS o a una estación de trabajo.

El servicio DICOM storage commitment es usado para confirmar que una imagen ha sido almacenada permanentemente por un dispositivo. El usuario de la clase de servicio (modalidad, estación de trabajo, etc.) utiliza la confirmación del la clase de servicio proveedor (estación de almacenamiento) para asegurarse de que puede borrar la imagen localmente (WIKIPEDIA 2007a).

Agrupación de Empresas de Sistemas de Farmacias (ADESFA)

El estándar ADESFA está dirigido a facilitar la gestión farmacéutica, estableciendo como deben ser los mensajes con la información necesaria para la validación de la receta médica.

El nombre de los archivos estará conformado por una serie numérica (que podría ser el número de mensaje) y con extensión ".xml". El nombre de archivo deberá tener 8 dígitos como máximo, al superar esta cifra deberá comenzar de cero nuevamente, deberán ser correlativos y no se deberán repetir en un lapso de tiempo razonable, a los efectos de que no se preste a confusión por números de mensajes iguales. En esta primera versión del mensaje no será obligatoria la pre-validación del XML. Por las características del mensaje no se puede utilizar DTD y por el momento está inmaduro para que se utilice otras prevalidaciones de XML más exigentes como puede ser XML SCHEMA (ADESFA 2007).

Health Level 7 (HL7)

El HL7 es una especificación para un estándar de intercambio de datos electrónicos en el ambiente de la atención de la salud, con especial énfasis en las comunicaciones intrahospitalarias. Es el resultado del trabajo de un Comité de proveedores de usuarios, vendedores y consultores de sistemas de aplicación al área de salud. El hospital promedio de la actualidad posee programas instalados que se ocupan del registro de los procesos de admisión y egreso de pacientes, de registro y producción de información de

laboratorio clínico, de informes de radiología y patología, de facturación y administración general, y otros. A menudo estas aplicaciones han sido desarrolladas por diferentes proveedores o grupos propios, poseyendo cada producto formatos de datos altamente específicos. A medida que los hospitales van expandiéndose, las operaciones de procesamiento de información se expanden en forma concomitante, y la necesidad de compartir los datos que encierran esa información se torna crítica. El desarrollo y disponibilidad de sistemas globales de informatización hospitalaria, que sólo algunos proveedores muy selectos han desarrollado hasta la fecha, mitigarían la necesidad de estándares para la transmisión externa de datos del tipo HL7. No obstante, estos programas son todavía escasos y su implementación requiere fuertes inversiones iniciales tanto de hardware como de software, lo cual hace que sigan existiendo y desarrollándose las aplicaciones específicas de bajo costo (MARCH 2006).

Por otro lado, las instituciones hospitalarias aún son el objeto de fuertes presiones en el sentido de la adquisición o el desarrollo de aplicaciones departamentales con un criterio modular. Fuente característica de este tipo de presiones son ciertas necesidades departamentales de procesamiento de información no adecuadamente resueltas por los sistemas "globales". Otra fuente característica de presiones es la necesidad de desarrollar un sistema finalmente global a través de sucesivas etapas, atendiendo a necesidades departamentales según un esquema de prioridades, y no a efectuar una sola adquisición, brusca y revolucionaria (ANSI 2006).

El término "Nivel 7" se refiere al más alto de los niveles del modelo OSI (Open Systems Interconnection) de la ISO (International Standards Organization). Esto no implica que el HL7 conforme específicamente a determinados elementos constitutivos del séptimo nivel del OSI. De hecho, el HL7 no especifica un conjunto de especificaciones propias del OSI para los niveles 1 al 6. En cambio, si conforma específicamente a la definición conceptual de un enlace de aplicación en lo que hace estrictamente al nivel 7 del modelo OSI (BIOCOM 2006a).

Técnicas de Inteligencia Artificial aplicadas a solución de problemas de tomas de decisión

Las técnicas de Inteligencia Artificial (IA) han sido ampliamente explotadas en la solución de problemas de ayuda a la toma de decisiones en el diagnóstico y tratamiento de pacientes por sus potencialidades. Mediante la utilización de estas técnicas se pretende emular la capacidad natural que posee el hombre en la toma de decisiones de cualquier tipo; imitando tanto su modo de aprendizaje como la manera que basándose en dicho conocimiento puede llegar a tomar decisiones; características que son las bases

fundamentales para diagnóstico y tratamiento. Para este fin la IA se apoya haciendo uso de sus propias ramas, distintas entre sí, dentro de las cuales se pueden destacar: los sistemas basados en el conocimiento (SBC) (GARCÍA *et al.* 2004).

En (Anexo 1 Aplicaciones que utilizan Inteligencia Artificial) se muestran alguno de los sistemas que se han hecho recientemente, dirigidas al área de la medicina. La mayoría de estas aplicaciones apoyan el diagnóstico y tratamiento.

Determinado por la Forma de Representación del Conocimiento se da lugar a diferentes variantes de SBC, entre los más conocidos y empleados en labores de clasificación encontramos: los sistemas basados en reglas, los sistemas basados en casos, las redes neuronales artificiales y los sistemas de inferencia borrosos y neuro-borrosos (GARCÍA *et al.* 2004).

La Forma de Representación del Conocimiento es sumamente importante y consiste en tomar el conocimiento extraído y representarlo de una manera inteligible, cuando se hace la adquisición del conocimiento este se va registrando de alguna manera no formal y es esta representación la base inicial de la representación del conocimiento que se ha extraído. Para los sistemas basados en el conocimiento se han determinado algunas representaciones que se han convertido en estándar para la representación del conocimiento, entre ellas están: la lógica preposicional y la lógica de predicados, las reglas de producción, las redes semánticas, los marcos, los guiones.

Explicación de algunas de las variantes:

Sistemas basados en reglas (SBR) se caracterizan porque la forma de representación del conocimiento son las reglas de producción y como método de inferencia utilizan la regla de modus ponens. Las reglas de este tipo de SBC expresan siempre una condicional, con antecedentes y un consecuente. La interpretación de una regla parte del hecho que si los antecedentes se satisfacen entonces se obtiene el consecuente, este tipo de reglas se conoce en la literatura como reglas duras. Como ventaja fundamental muestran su capacidad de interpretación y explicación de la inferencia. Pero solo deben ser utilizados en los casos donde el conocimiento pueda ser representado de forma eficiente en forma de reglas duras evitando situaciones donde haya inconsistencia e imprecisión de los datos y evitando su aplicación en problemas donde exista vaguedad en la definición de los conceptos.

Los sistemas basados en casos (CBR) basan sus potencialidades para el aprendizaje en los mecanismos de detección de similaridades entre casos y también en la propiedad de incorporar de forma natural nuevo conocimiento mejorando gradualmente su funcionamiento. Su principal limitación radica en

la capacidad de representación de los valores de los rasgos, aspecto que dificulta su aplicación en situaciones que requieren representaciones de casos complejas como por ejemplo las que se presentan cuando existe un alto grado de interrelación entre los rasgos. También hay que señalar que la definición de las funciones de semejanza y los mecanismos de selección de rasgos para cada problema específico son problemas abiertos en este campo, cuya solución depende en gran medida de la experiencia de los expertos (AHA 1991; MARTÍNEZ 2003).

Redes neuronales artificiales, también conocidos como modelos conexionistas, consisten en un conjunto de elementos computacionales simples, que están unidos por arcos dirigidos que le permiten comunicarse. Cada arco tiene asociado un peso numérico Wij que indica la significación de la información que llega por este arco, o sea, la influencia que tiene la activación alcanzada por la neurona Ui sobre la neurona Uj es precisamente en los pesos donde se almacena de forma intrínseca el conocimiento de este tipo de sistema. Hay tres aspectos que caracterizan a las redes neuronales y permiten distinguir a cada uno de los diversos modelos, ellos son: la topología de la red, las características de los nodos (modelo de la neurona), y el mecanismo de aprendizaje (HALGAMUGE and GLESNER 1994).

El uso de las redes neuronales ha mostrado una alta eficiencia en la resolución de diversos problemas, no obstante su efectividad se les señala que se comportan como cajas negras y no facilitan la interpretación de los resultados. Tendencias actuales en el uso de estas se presenta en combinación con otros modelos como partes integrantes de sistemas multiclasificadores, sistemas neuroborrosos y redes neuronales ensambladas.

Sistemas de inferencia borrosos: la base del funcionamiento de los sistemas de inferencia borrosos son los conjuntos borrosos. A diferencia de los conjuntos clásicos en estos conjuntos la pertenencia de un elemento se convierte en un problema de grado. Más formalmente podemos decir que un conjunto borroso A en un universo de discurso U está caracterizado por una función de pertenencia μa la cual a cada elemento en el dominio le asigna un grado de pertenencia al conjunto en el intervalo y se representa de la forma $\mu a : \cup \rightarrow [0,1]$. De esta forma un mismo elemento puede pertenecer a varios conjuntos simultáneamente solo que con cierto grado de pertenencia. Cada conjunto borroso tiene asociado además un término lingüístico de forma tal que la función de pertenencia asociada a un conjunto está ligada a una palabra como por ejemplo: bajo, medio, más o menos alto, alto, muy alto, etc.

Los Métodos de Solución de Problemas son otras de las técnicas de IA empleadas habitualmente en la búsqueda de soluciones para problemas complejos como pueden ser los de diagnosticar, indicar tratamiento y controlar a un paciente con determinada enfermedad.

La solución de problemas puede considerarse una forma de razonamiento muy compleja que requiere la generación y asimilación de nuevas estructuras de memoria a fin de contestar una interrogante. Es la actividad mental de encontrar una solución a un problema. En el contexto del procesamiento de la información el enfoque dado a la solución de problemas ha sido tratar de trazar la gráfica de la secuencia de eventos desde la formulación del problema hasta su solución final; o sea, tratar de comprender el proceso que interviene para derivar una solución (BELLO 1998).

Hay cuatro vías principales para encontrar la solución a un problema dado:

1. La aplicación de una fórmula explícita que da la solución.
2. El uso de una definición recursiva.
3. El uso de un algoritmo que converge a la solución.
4. La aplicación de otros procesos, en particular la prueba y error.

La última de estas vías es la utilizada para problemas en los cuales no hay una solución algorítmica conocida o es tan compleja que no es posible una implementación computacional conocida que pueda ser práctica. Sin embargo, estos problemas son resueltos por los humanos a pesar de su capacidad de procesamiento "inferior". De esta clase de problemas es de los que se ocupan los métodos de solución de problemas de la Inteligencia Artificial (S. RUSELL 1995).

Entonces la respuesta fue desarrollar nuevas técnicas de solución de problemas, similares a las humanas, una de las más importantes fue la búsqueda.

La búsqueda de la Inteligencia Artificial difiere de la búsqueda convencional sobre estructuras de datos esencialmente en que se busca en un espacio problema, no en una pieza de dato particular. Se busca un camino que conecte la descripción inicial del problema con una descripción del estado deseado para el problema, es decir, el problema resuelto. Este camino representa los pasos de solución del problema.

Sin embargo si se desea definir formalmente la solución de problemas en IA se tiene que considerar lo siguiente:

Un agente, quien actúa como resolvedor de problemas. Un problema es realmente una colección de información que el agente usará para decidir qué hacer. Los elementos básicos de la definición de un problema son los estados y las acciones. Los elementos son los siguientes:

El *estado inicial* donde se encuentra el agente.

El conjunto de *acciones* posibles disponibles al agente. El término operador se usa para denotar la descripción de una acción en términos de cual estado será alcanzado ejecutando la acción en un estado particular. Una formulación alternativa es usar una función sucesor S; dado un estado particular x, S(x) retorna al conjunto de estados alcanzables desde x mediante una acción simple.

El *espacio de estado* del problema es el conjunto de todos los estados alcanzables a partir del estado inicial mediante una secuencia de acciones cualquiera.

Un *camino* en el espacio de estado es una secuencia de acciones que conduce de un estado a otro.

El *criterio objetivo* es el criterio que el agente usa para aplicarlo a la descripción de un estado para determinar si este es un *estado objetivo* (lo que se desea obtener). Algunas veces hay un conjunto explícito de posibles objetivos y el criterio simplemente consiste en chequear si se ha alcanzado uno de ellos. Otra variante es especificar el objetivo por una propiedad abstracta, por ejemplo, el criterio de jaque mate del ajedrez.

El *costo de un camino* es una función que asigna un costo a un camino.

Una *solución* es un camino desde el estado inicial a un estado que satisface el criterio objetivo.

Otro concepto importante es el de *costo de la búsqueda* asociado con el tiempo y la memoria requisitos para encontrar una solución.

El *costo total* de la búsqueda es la suma del costo del camino y el costo de la búsqueda.

Dada la formulación de un problema la próxima acción es encontrar una solución, lo cual como ya se mostró, consiste en generar nuevos estados a partir del estado donde se encuentra el agente. El proceso consiste en generar nuevos estados a partir del estado donde se encuentra el agente. Este proceso se denomina *expandir* el estado. La expansión puede producir uno o varios nuevos estados. En el primer caso se toma este y se continúa, en el otro caso existen múltiples posibilidades para continuar la búsqueda por lo que es necesaria una selección.

Esta es la esencia de la búsqueda, seleccionar una opción y poner las otras alternativas a un lado para retomarla más tarde si la primera selección no conduce a una solución. La selección de cuál estado expandir primero se determina por la *estrategia de búsqueda* (BELLO 1998).

Se ha abarcado algunas de las variantes basadas en técnicas de IA a emplear a la hora de resolver complejos problemas que se pueden presentar en el desarrollo de aplicaciones médicas, cada una de estas técnicas tienen un determinado grado de adaptabilidad a las circunstancias, lo cual posibilita su

empleo, sin embargo es conveniente realizar un detenido y minucioso análisis de ellas para obtener los mejores resultados en su uso.

Conclusiones

Se han descrito los procesos por los que se debe transitar para la indicación de tratamiento farmacológico a un paciente, se puede concluir que este se ha convertido en un problema complejo de solucionar y que se manifiesta a nivel mundial. Se ha analizado como esta complejidad propicia que las soluciones informáticas cubran solo parcialmente el tema. Las herramientas analizadas solo abarcan hasta el proceso de evaluación clínica del paciente, procesos como la indicación de un tratamiento farmacológico y control no se tratan con la importancia necesaria.

Se analizaron un conjunto de conceptos y aspectos dirigidos al diseño de software, resaltando la incidencia de un mal diseño y los factores que un correcto diseño de una aplicación informática. Concluyendo que es necesario la utilización de patrones y estándares que propiciarían resultados positivos en materia de cohesión, acoplamiento, reutilización, calidad y facilidad de mantenimiento.

También se analizan un grupo de técnicas de Inteligencia Artificial que pueden ser empleadas para resolver los problemas de ayuda en la toma de decisiones en materia de indicación de tratamiento farmacológico.

La situación existente lleva a que se plantee el desarrollo de un correcto diseño de los módulos de Tratamiento Farmacológico y Configuración que ayudaran a los médicos en el tratamiento de la HTA en los pacientes, permitiendo esto una posterior implementación de una aplicación informática.

2

CAPÍTULO

METODOLOGÍAS, HERRAMIENTAS Y PATRONES DE DISEÑO

Introducción

En este capítulo se analizan un conjunto de metodologías de desarrollo adaptables a los requisitos de esta propuesta y se concluye en cuales fueron las razones que llevaron a la selección de una de ellas, Rational Unified Process (RUP) en este caso. Así mismo se dirige la atención hacia herramientas cases, valorándose la factibilidad de uso de cada una para el desarrollo de la solución.

Se desarrolla un análisis del estado del arte de las herramientas CASE del modelado de diseño y se justifica la decisión asumida para este trabajo. Buscándose así una correcta selección que se integre a la metodología seleccionada y puedan complementarse en el trabajo de obtener un diseño en equilibrio con las necesidades del cliente, las características del equipo de desarrollo y su productividad bajo las diversas condiciones existentes hoy en los entorno de trabajo para el desarrollo de software de la Facultad 3 de la Universidad de las Ciencias Informáticas.

También se revisan algunos de los patrones de diseño más utilizados, explicándose como se emplean en la producción de la solución propuesta algunos de ellos o propuestas hibridas de los mismos, incluyéndose además los principios que se asumieron en el diseño de la interfaz de la aplicación.

Metodologías ágiles y pesadas

No existe una metodología universal para hacer frente con éxito a cualquier proyecto de desarrollo de software. Toda metodología debe ser adaptada al contexto del proyecto según los recursos técnicos y humanos, tiempo de desarrollo, tipo de sistema, etc.

Existen situaciones que fuerzan a un proceso de desarrollo asociado a un marcado énfasis en el control del proceso mediante una rigurosa definición de roles, actividades y artefactos, incluyendo modelado y documentación detallada. Este esquema es considerado como "tradicional" y para abordar el desarrollo de software ha demostrado ser efectivo y necesario en proyectos de gran tamaño (respecto a tiempo y recursos), donde por lo general se exige un alto grado de ceremonia en el proceso. Sin embargo, este

enfoque no resulta ser el más adecuado para muchos de los proyectos actuales donde el entorno del sistema es muy cambiante, y en donde se exige reducir drásticamente los tiempos de desarrollo pero manteniendo una alta calidad. Ante las dificultades para utilizar metodologías tradicionales con estas restricciones de tiempo y flexibilidad, muchos equipos de desarrollo se resignan a prescindir del "buen hacer" de la ingeniería del software, asumiendo el riesgo que ello conlleva. Es así que, las metodologías ágiles emergen como una posible respuesta para llenar ese vacío metodológico. Por estar especialmente orientadas hacia proyectos pequeños, las metodologías ágiles constituyen una solución a la medida para ese entorno, aportando una elevada simplificación.

A finales de los años 90 del siglo pasado, reputados profesionales con renombre y eco en diferentes foros técnicos, comenzaron a cuestionar las metodologías formales, que representadas por CMM e ISO 15504, y respaldadas por la autoridad y los medios de sus respectivas organizaciones, estaban configurando una ingeniería del software basada en los procesos.

En marzo de 2001, 17 críticos de estos modelos, convocados por Kent Beck, que acababa de definir una nueva metodología denominada *Extreme Programming*, se reunieron en Salt Lake City para discutir sobre los modelos de desarrollo de software.

En la reunión se acuñó el término "Métodos Ágiles" para definir a aquellos que estaban surgiendo como alternativa a las metodologías formales, a las que consideraban excesivamente "pesadas" y rígidas por su carácter normativo y fuerte dependencia de planificaciones detalladas, previas al desarrollo.

Los integrantes de la reunión resumieron en cuatro postulados lo que ha quedado denominado como "Manifiesto Ágil", que compendia el espíritu en el que se basan estos métodos (PALACIO 2005).

Se hará una breve reseña de estas Metodologías Ágiles, para más adelante llegar a las "tradicionales".

Dynamic Systems Development Method (DSDM)

Es la metodología más veterana de las autodenominadas ágiles. Surgió en 1994 de los trabajos de Jennifer Stapleton, la actual directora del DSDM Consortium. DSDM es la metodología ágil más próxima a los métodos formales, de hecho la implantación de un modelo DSDM en una organización la lleva a alcanzar lo que CMM consideraría un nivel 2 de madurez.

Los aspectos que DSDM reprocha a CMM son:

- Aunque es cierto que se ha desarrollado con éxito en algunas organizaciones, lo que funciona bien en unos entornos no tiene por qué servir para todos.
- CMM no le da al diseño la importancia que debería tener.

- CMM plantea un foco excesivo en los procesos, olvidando la importancia que en nuestra industria tiene el talento de las personas.
- El tener procesos claramente definidos no es sinónimo de tener buenos procesos.

Al igual que los métodos ágiles, DSDM considera imprescindible una implicación y una relación estrecha con el cliente durante el desarrollo, así como la necesidad de trabajar con métodos de desarrollo incremental y entregas evolutivas.

DSDM cubre los aspectos de gestión de proyectos, desarrollo de los sistemas, soporte y mantenimiento y se autodefine como un marco de trabajo para desarrollo rápido más que como un método específico para el desarrollo de sistemas (STAPLETON 2003).

Extreme Programming (XP)

Este es el método que más popularidad ha alcanzado entre las metodologías ágiles, y posiblemente sea también el más transgresor de la ortodoxia basada en procesos.

Fue creado por Kent Beck. XP se levanta sobre la suposición de que es posible desarrollar software de gran calidad a pesar, o incluso como consecuencia del cambio continuo. Su principal asunción es que con un poco de planificación, un poco de codificación y unas pocas pruebas se puede decidir si se está siguiendo un camino acertado o equivocado, evitando así tener que echar marcha atrás demasiado tarde.

Los valores que lo inspiran son cuatro: simplicidad, feedback, coraje y comunicación (JOSÉ H. CANOS 2003).

Así mismo XP define *UserStories* como base del software a desarrollar. Estas historias las escribe el cliente y describen escenarios sobre el funcionamiento del software, que no solo se limitan a la Interfaz de Usuario si no también pueden describir el modelo, dominio, etc. A partir de las UserStories y de la arquitectura perseguida se crea un plan de *releases* (*liberación* o *entrega* del software) entre el equipo de desarrollo y el cliente.

XP no es un modelo de procesos ni un marco de trabajo, sino un conjunto de 12 prácticas que se complementan unas a otras y deben implementarse en un entorno de desarrollo cuya cultura se base en los cuatro valores citados:

Comunicación: XP pone en comunicación directa y continua a clientes y desarrolladores. El cliente se integra en el equipo para establecer prioridades y resolver dudas. De esta forma ve el avance día a día, y es posible ajustar la agenda y las funcionalidades de forma consecuente.

Feedback rápido y continuo: una metodología basada en el desarrollo incremental de pequeñas partes, con entregas y pruebas frecuentes y continuas, proporciona un flujo de retro-información valioso para detectar los problemas o desviaciones. De esta forma los fallos se localizan muy pronto. La planificación no puede evitar algunos errores, que sólo se evidencian al desarrollar el sistema. La retro-información es la herramienta que permite reajustar la agenda y los planes.

Simplicidad: la simplicidad consiste en desarrollar sólo el sistema que realmente se necesita. Implica resolver en cada momento sólo las necesidades actuales. "*Los costes y la complejidad de predecir el futuro son muy elevados, y la mejor forma de acertar es esperar al futuro.*"

Con este principio de simplicidad, junto con la comunicación y el feedback resulta más fácil conocer las necesidades reales.

Coraje: el coraje implica saber tomar decisiones difíciles. Reparar un error cuando se detecta. Mejorar el código siempre que tras el feedback y las sucesivas iteraciones se manifieste susceptible de mejora. Tratar rápidamente con el cliente los desajustes de agendas para decidir qué partes y cuándo se va a entregar.

Las 12 prácticas que aplicadas sobre esta cultura conforman Extreme Programming son:

Prácticas de codificación

1. Simplicidad de código y de diseño para producir software fácil de modificar.
2. Reingeniería continúa para lograr que el código tenga un diseño óptimo.
3. Desarrollar estándares de codificación, para comunicar ideas con claridad a través del código.
4. Desarrollar un vocabulario común, para comunicar las ideas sobre el código con claridad.

Prácticas de desarrollo

1. Adoptar un método de desarrollo basado en las pruebas para asegurar que el código se comporta según lo esperado.
2. Programación por parejas, para incrementar el conocimiento, la experiencia y las ideas.
3. Asumir la propiedad colectiva del código, para que todo el equipo sea responsable de él.
4. Integración continua, para reducir el impacto de la incorporación de nuevas funcionalidades.

Prácticas de negocio

1. Integración de un representante del cliente en el equipo, para encauzar las cuestiones de negocio del sistema de forma directa, sin retrasos o pérdidas por intermediación.
2. Adoptar el juego de la planificación para centrar en la agenda el trabajo más importante.
3. Entregas regulares y frecuentes para satisfacer la inversión del cliente.
4. Ritmo de trabajo sostenible, para terminar la jornada cansado pero no agotado.

Otros métodos ágiles son los que se relacionan a continuación.

Familia de métodos Crystal

La familia de metodologías Crystal ofrece diferentes métodos para seleccionar el más apropiado para cada proyecto (PALACIO 2005).

Crystal identifica con colores diferentes cada método, y su elección debe ser consecuencia del tamaño y criticidad del proyecto, de forma que los de mayor tamaño, o aquellos en los que la presencia de errores o desbordamiento de agendas implique consecuencias graves, deben adoptar metodologías más pesadas.

Los métodos Crystal no prescriben prácticas concretas, y se pueden combinar con técnicas como XP.

Adaptative Software Development (ASD), método que como alternativa a los procedimientos formales, aborda el desarrollo de grandes sistemas con el uso de técnicas propias de las metodologías ágiles. No se trata de una metodología, sino de la implantación de una cultura en la empresa, basada en la adaptabilidad.

Pragmatic Programming (PP) es la colección de 70 prácticas de programación, comunes a otros métodos ágiles, cuya aplicación resulta útil para solucionar los problemas cotidianos.

Agile Modeling (AM) es la presentación de un nuevo enfoque para realizar el modelado de sistemas (diseño), y basado en los principios de los métodos ágiles remarca la conveniencia de reducir el volumen de la documentación.

Internet-Speed Development (ISD) es el más reciente de los métodos ágiles, surgido como respuesta para las situaciones que requieren ciclos de desarrollo muy breves con entregas rápidas. Se centra en el talento de las personas sobre los procesos. ISD es un entorno de gestión orientado al negocio.

Feature Driven Development (FDD) prescribe un proceso iterativo de 5 pasos, con iteraciones de dos semanas. El punto de referencia son las características que debe reunir el software, y se centra en las fases de diseño e implementación del sistema por lo que se aborda un poco más en el proceso (MOLPECERES 2003).

Un proyecto que sigue FDD se divide en 5 fases:

1. Desarrollo de un modelo general
2. Construcción de la lista de funcionalidades
3. Plan de releases en base a las funcionalidades a implementar
4. Diseñar en base a las funcionalidades
5. Implementar en base a las funcionalidades

Las primeras tres fases ocupan gran parte del tiempo en las primeras iteraciones, siendo las dos últimas las que absorben la mayor parte del tiempo según va avanzando el proyecto, limitándose las primeras a un proceso de refinamiento.

FDD esta pensado para proyectos con tiempo de desarrollo relativamente cortos (menos de un año). Se basa en un proceso iterativo con iteraciones cortas (~2 semanas) que producen un software funcional que el cliente y la dirección de la empresa pueden ver y monitorizar.

Las iteraciones se deciden en base a *features* (de ahí el nombre del proceso) o funcionalidades, que son pequeñas partes del software con significado para el cliente.

Otros métodos con fuerte difusión en la industria del software son los que utiliza Microsoft e IBM Rational, estos son los casos de Microsoft Solution FrameWork y el Proceso unificado de software respectivamente.

Métodos de propiedad comercial, Microsoft Solutions Framework (MSF).

MSF es la metodología empleada por Microsoft para el desarrollo de software e interrelaciona una serie de conceptos, modelos y prácticas de uso que controlan la planificación, el desarrollo y la gestión de proyectos tecnológicos. Hasta su versión 3 (principios de 2005) MSF se definía como un marco de desarrollo flexible para adaptarse a las necesidades de cada proyecto, en oposición a lo que sería una metodología prescriptiva; porque parte de la base de que no hay una estructura de procesos óptima para las necesidades de todos los entornos de desarrollo posibles (MICROSOFT 2006).

El marco MSF se asienta sobre unos principios fundamentales que definen la cultura del entorno de desarrollo:

- Fomento de la comunicación abierta.
- Trabajo en torno a una visión compartida.
- Apoderar a los integrantes del equipo ("*empowerment*")

- Establecimiento de responsabilidades claras y compartidas.
- Centrar el objetivo en la entrega de valor para el negocio.
- Permanecer ágiles y esperar el cambio.
- Invertir en calidad.
- Aprender de la experiencia.

Para la aplicación de estos principios en los procesos y en las personas, MSF define un *Modelo de Equipo* y un *Modelo de Procesos* (ÁVILA 2005). Sobre los *Modelos*, y trabajando con la cultura de sus *Principios Fundamentales*, las *Disciplinas* que establece para el desarrollo del software son:

- Gestión de proyectos.
- Gestión de riesgos.
- Gestión de la mejora del talento.

MSF despliega la gestión de proyectos y la gestión de riesgos con algunas diferencias sobre las visiones clásicas de estas áreas. El marco de desarrollo incluye también *Conceptos Clave*, *Prácticas Contrastadas* y *Recomendaciones* para la ejecución de las tareas concretas en el desarrollo de software.

En 2005, el desarrollo del nuevo producto de Microsoft "Visual Studio 2005 Team System" ha ganerado la evolución de MSF hacia la nueva versión 4.0 con dos líneas paralelas:

- Microsoft Solutions Framework (MSF) for Agile Software Development (MICROSOFT 2005a).
- Microsoft Solutions Framework (MSF) for CMMI Process Improvement (MICROSOFT 2005b).

Proceso unificado de desarollo (RUP)

Analizando Rational Unified Process (RUP), considerada una metodología pesada que en la actualidad se utiliza frecuentemente. Es un proceso de Ingeniería del Software que proporciona una visión disciplinada para la asignación de tareas y responsabilidades en las organizaciones de desarrollo de software.

RUP es un "modelo-producto" desarrollado y mantenido por Racional Software, integrado en su conjunto de herramientas de desarrollo, y distribuido por International Business Machines (IBM) (PALACIO 2005).

RUP integra un conjunto de "buenas prácticas" para el desarrollo de software en un marco de procesos válido para un rango amplio de tipos de proyectos y organizaciones.

Las principales características del RUP son:

- Guiado por Casos de Uso
- Centrado en la arquitectura
- Iterativo incremental

Las principales buenas prácticas cubiertas son:

- Desarrollo iterativo.
- Gestión de requisitos.
- Uso de arquitecturas basadas en componentes.
- Uso de técnicas de modelado visual.
- Verificación continúa de la calidad.
- Gestión y control de cambios.

En su visión estática, el modelo RUP está compuesto por:

- Roles: analista de sistemas, diseñador, diseñador de pruebas, roles de gestión y roles de administración.
- Actividades: RUP determina el trabajo de cada rol a través de actividades. Cada actividad del proyecto debe tener un propósito claro, y se asigna a un rol específico. Las actividades pueden tener duración de horas o de algunos días; y son elementos base de planificación y progreso.
- Artefactos: Son los elementos de entrada y salida de las actividades. Son productos tangibles del proyecto. Las cosas que el proyecto produce o usa para componer el producto final (modelos, documentos, código, ejecutables, etc.)
- Disciplinas: son "contenedores" empleados para organizar las actividades del proceso. RUP comprende 6 disciplinas técnicas y 3 de soporte. *Técnicas*: modelado del negocio, requisitos, análisis y diseño, implementación, pruebas y desarrollo. *Soporte*: gestión de proyecto, gestión de configuración y cambio, y entorno.
- Flujos de trabajo: son el "pegamento" de los roles, actividades, artefactos y disciplinas, y constituyen la secuencia de actividades que producen resultados visibles.

Capítulo II. "Metodologías, Herramientas y Patrones de diseño"

En su visión dinámica, la visión de la estructura del ciclo de vida RUP se basa en un desarrollo iterativo, concretado por hitos para revisar el avance y planear la continuidad o los posibles cambios de rumbo. Cuatro son las fases que dividen el ciclo de vida de un proyecto RUP:

1. *Inicio.* Es la fase de la idea, de la visión inicial de producto, su alcance. El esbozo de una arquitectura posible y las primeras estimaciones. Concluye con el "hito de objetivo".
2. *Elaboración.* Comprende la planificación de las necesidades y el diseño de la arquitectura. Termina con el "hito de Arquitectura".
3. *Construcción*. Desarrollo del producto hasta que se encuentra disponible para su entrega a los usuarios. Termina con el "hito del inicio de la capacidad operativa".
4. *Transición*. Traspaso del producto a los usuarios. Incluye: manufactura, envío, formación, asistencia y el mantenimiento hasta lograr la satisfacción de los usuarios. Termina con el "hito de entrega del producto".

Del análisis de cada una de las posibilidades que brindan las metodologías estudiadas se decide escoger una que sea capaz de adaptarse a las características que tiene el entorno en que se enmarca la solución y los propios detalles del diseño de la problemática que soluciona este trabajo. La selección de RUP como metodología que guiaría el desarrollo de software estuvo determinada desde el comienzo del proyecto HyperWeb por razones relacionadas con el dominio de RUP en la Universidad de las Ciencias Informáticas en comparación con el resto de las metodologías, la imposición de una disciplina de desarrollo concreta y el volumen de artefactos que como metodología esta exige, que posibilita dejar un archivo documental muy rico del proyecto y los resultados investigativos. Así mismo se analizó la necesidad de que cada uno de los flujos de trabajo por los que se transitara existiese una correcta documentación que cumpliese con las exigencias del cliente y que impusiese una disciplina en todo el ciclo de desarrollo. En RUP las entregas basados en artefactos después de cada fase al contrario de las metodologías como XP no solo se limitan al código sino que van acompañadas de todas las características de una versión final (manual de ayuda, instrucciones de instalación, notas de la versión, entre otros documentos). RUP es por otra parte una metodología que genera gran cantidad de documentación, de ahí su clasificación como "pesada", aunque define un proceso de desarrollo genérico adaptable a las más diversas características, aspecto que se explota en la realización de la solución propuesta.

Capítulo II. "Metodologías, Herramientas y Patrones de diseño"

Debido al carácter general de la metodología seleccionada algunos autores consideran todos los demás procesos de desarrollo como casos particulares de esta. Para realizar un correcto diseño se enfatizan en un conjunto de artefactos, actividades que permitirán llegar a la implementación del los módulos seleccionados (en este caso Tratamiento Farmacológico y Configuración) a partir del logro de un correcto entendimiento por el equipo de desarrollo del objetivo a lograr y las tareas a realizar, y a la vez permite trabajar de la forma correcta. Siempre enmarcándose cada trabajador en su responsabilidad, en las fases y flujos de trabajo que recoge RUP.

Herramientas CASE para el modelado de diseño. Herramientas utilizadas

La herramientas Computer-Aided Software Engineering (CASE) dedicadas al desarrollo de software, están vinculadas a la asistencia dirigidas de analistas, ingenieros de software y desarrolladores, durante todos los pasos del Ciclo de Vida de desarrollo de un Software (Investigación Preliminar, Análisis, Diseño, Implementación e Instalación.) sea cual sea la metodología o proceso de desarrollo. Es difícil ver una separación entre estas herramientas y las metodologías de desarrollo de software formales que ellas modelan. En muchos otros casos como Embarcadero (ERwin), Rational (Rose, Clear Case, etc.), Microsoft (TEAM SYSTEM) los propios consorcios le dan soporte y desarrollan una especialización del estándar cuyas mejoras fortalecen más su paradigma, y con ello garantizan su supremacía en el mercado.

En este epígrafe se desea abordar algunas herramientas CASE, que pueden ser usadas tanto para plataforma propietaria (Windows, MAC) como libres (Linux), sin intención de hacer un estudio profundo en las mismas, pero si dar elementos que permitan comparar este tipo de aplicaciones y entender las razones que llevaron a la selección de las que se emplean en la solución de este trabajo. Si se desea profundizar más en el tema de las herramientas CASE ver: (TOLLS 2005)

Algunos autores plantean que existen generalmente tres tipos de sistemas CASE:

Las herramientas de Diseño, las de Ambientes de Construcción, y aquellas que resultan un hibrido de los dos enfoques anteriores. Las herramientas de diseño CASE auxilian a los equipos de ingenieros en la especificación de sistemas de software y ayudan en la automatización de la escritura de arquitecturas, documentación, y además integran automáticamente esas piezas generadas en un Entorno Integrado de Desarrollo (IDE). Muchas herramientas CASE utilizan el Lenguaje de Modelado Unificado (UML) desarrollador por Grady Booch, Jim Rumbaugh, e Ivar Jacobsen. Su compañía, Rational Software es una de la más conocidas en cuanto a la creación de sistemas CASE. La disponibilidad de UML ha

revolucionado la habilidad de los ingenieros de software para crear especificaciones de sistemas que pueden ser relativamente fáciles de traducir en código mantenible y que funcione. Hay herramientas CASE para casi todo tipo de especialización desde diseño de bases de datos, hasta el modelado de algunos flujo de trabajo como requisito, análisis y diseño, plantación, gestión de proyecto, entre otros.

En el caso de las Herramientas de construcción, estas auxilian a los equipos en la construcción y administración de liberación de paquetes de software, mientras las Herramientas híbridas son un nuevo fenómeno, como puede ser la creación de una aplicación de Servicios Web para crear un sistema distribuido que puede manejar múltiples estilos de desarrollo y la flexibilidad de agregar nuevas herramientas y servicios sin mucho trabajo. Buenos ejemplos incluyen a Sourceforge, Collab.NET, y todas sus variantes (CARRIÓN PINZÓN MARIO ISRAEL 2007).

A continuación son referenciados algunas herramientas CASE y plataformas de herramientas CASE con gran popularidad hoy en el mundo, y se muestran algunas de sus principales características:

Visual Paradigm. La gama de productos Paradigm modelan todos los flujos del desarrollo del software, son multiplataforma, pertenecen a la compañía Visual Paradigm International. No son gratis y permiten la integración con plataformas de desarrollo como Visual Studio.Net (Microsoft), Eclipse, NetBeans, JDeveloper, JBuilder ,Suntm One, IntelllJ Ideatm, WebLogic Workshoptm . Los productos más importantes son: Visual Paradigm for UML, Smart Development Environment, DB Visual Architect, Business Process Architect, Team work server. Entre sus potencialidades de modelado están: soporte a UML, administración de requisitos, modelado de procesos de negocio, modelado de bases de datos, generación de código de esquemas persistentes a través del modelo de mapeo relacional para diferentes lenguajes, incluidos entre ellos, .net, PHP. Modelado visual del análisis y el diseño, incluyendo sus diagramas, posibilidad de configuración de estilos y formatos de la documentación que genera vía automática de todo el proceso modelado por la herramienta. Estas herramientas permiten además el modelado de la teoría de trabajo en equipo, generación de reportes y gestión de la información y los involucrados en el desarrollo, generación de código, proceso reingeniería, modelado de arquitecturas entre otras (VISUAL_PARADIGM_INTERNATIONAL_COMPANY. 2007).

Enterprise Architect. Esta herramienta ha sido desarrollada por Sparx Systems, no es multiplataforma, solo soportado por la plataforma Windows. No es gratis, sus funcionalidades abarcan integralmente el ciclo de vida de desarrollo, cubren el desarrollo de software desde el levantamiento de requisitos, las etapas de análisis y diseño, hasta prueba(Testing), mantenimiento y re-uso. Puede ser utilizado para el

desarrollo de varios tipos de software, en un amplio rango de industrias, incluyendo: bancos, desarrollo Web, ingeniería, finanzas, medicina, investigación, educación, transporte, ventas, energía, ingeniería electrónica. Soporta el lenguaje UML 2.1 y permite la modelación de arquitecturas de negocio. Entre los artefactos del modelado que soporta están diagramas de Estructura, diagramas de Comportamiento, extensiones Temporales, análisis personalizados (requisitos, diseño de Interfaz de Usuario) (SYSTEMS 1996 - 2007).

Embarcadero Tecnologies. La plataforma de productos de Embarcadero está especializada en el modelado de bases de datos, y en este sentido son evaluadas por muchos clientes como una de las mejores de su tipo existentes hoy en día. No es gratis. Entre sus productos más usados se encuentran ER/Estudio para el modelado escalable y práctico de bases de datos, con soporte de drivers para casi todos los gestores de bases de datos existentes; DT/Studio, sistema que permite la asistencia en el modelado de análisis y diseño, con utilidades para la planificación de tareas, reporte y comunicación entre los involucrados en el proyecto, administración y mantenimiento del proyecto. Las plataformas que soportan los productos de Embarcadero son Oracle, Microsoft SQL Server, MySQL, IBM DB2 UDM para Windows y Linux. Las soluciones de modelado de bases de datos de embarcadero están orientadas a los siguientes roles: Administrador de bases de datos, arquitecto de bases de datos, administración de información segura y desarrollador de bases de datos. Sus productos tiene orientación multiplataforma (EMBARCADERO_TECHNOLOGIES 2007).

Rational Software. IBM. La familia de productos de Rational está sustentada en los principios de desarrollo de software que plantean RUP y el lenguaje de modelado UML, aunque soportan otras metodologías y tecnologías. Entre las plataformas y metodologías de desarrollo que pueden ser modeladas por sus productos se encuentran .Net, J2EE, J2SE, JDK, VC6, RUP. Tiene soporte para plataforma Windows y UNIX. Es una de las plataformas de herramientas CASE más difundidas del mundo, y su calidad la ha mantenido por mucho tiempo entre los lideres de este tipo de herramientas. Es una plataforma de productos y servicios desarrollados por la compañía IBM. Los productos CASE de Rational Software se agrupan según la prestación, los relacionados con:

- Analysis Modeling & Design (Ejemplo: Rational Rose)
- Change Configuration & Release Management (Ejemplo: Rational ClearCase y Rational ClearQuest)
- Process and Portfolio Management (Ejemplo: Rational Portfolio Manager)
- Requirements & Analysis (Ejemplo: Rational RequisitePro)

- Software Quality Management (Ejemplo: Rational Test RealTime)

Entre las potencialidades de los productos Rational Software están, el modelado de procesos de negocio, el modelado y administración de requisitos, el modelado de análisis y diseño de sistemas de software, así como un asistente para la aplicaciones de patrones de diseño y estilos arquitectónicos, también posibilita la generación de código de un sin números de lenguaje y tecnologías, incluidos entre ellos VC++, Java, C++, ADA, Corba, Oracle 8, COM, Visual Basic, etc. Permiten además la reingeniería de sistemas así como el modelado de arquitecturas entre otras (IBM_RATIONAL_SOFTWARE 2007).

Visual Team System. Este producto es desarrollado por la empresa Microsoft y es lo más representativo en cuanto a concepto de integración de servicios que se ha logrado en herramientas CASE dedicadas al desarrollo de software, basa sus funcionalidades en la teoría de MSF, no es gratis, pero a diferencia de las demás herramientas propietarias, esta compañía goza de muy mala reputación por sus políticas monopólicas de mercado. Sus productos tienen un precio razonable en comparación a sus servicios y generan una alta dependencia a toda la plataforma tecnológica de la compañía, lo que impide sistemas altamente escalables en caso de que estos productos quieran ser extendidos fuera de plataforma Windows. Visual Team System cuenta con los siguientes módulos Team Foundation Server concebido para la colaboración en equipo, control de versiones, gestión de cambios, administración de la generación y elaboración de informes. Los elementos de trabajo pueden personalizarse para requisitos, seguimiento de errores y asignaciones de tareas. Admite la implementación de metodología de procesos. El módulo Team Suite incluye todas las funciones de Team Edition. Ofrece a los miembros del equipo de desarrollo todo el conjunto de herramientas para el desarrollo del software. El módulo Team Edition para arquitectos que contiene herramientas de diseño visual para el diseño y la arquitectura de aplicaciones distribuidas. Facilita la validación del diseño con respecto a un entorno operativo de destino, y reduce el riesgo de problemas en la implementación. El módulo Team Edition para desarrolladores, contiene herramientas de desarrollo avanzadas que permiten a los desarrolladores añadir calidad, desde el principio y con frecuencia, durante todo el ciclo de vida. Gracias a las herramientas de análisis de rendimiento y seguridad, los desarrolladores pueden medir, evaluar y centrarse en el rendimiento y en los problemas relacionados con la seguridad del código de forma anticipada. El módulo Team Edition para profesionales de comprobación facilita una solución para pruebas en la Web y ensayos de carga de rendimiento. Amplia capacidad para realizar pruebas unitarias. El módulo Team Edition para profesionales

de bases de datos contiene herramientas que ayudan a los desarrolladores y administradores de bases de datos a gestionar cambios, pruebas y la implementación de bases de datos SQL Server, así como a crear aplicaciones relacionadas con bases de datos y por último el módulo Team Test Load que permite la simulación de unos 1.000 usuarios por procesador. Permite la simulación y comprobación más exacta de rendimiento de aplicaciones y servidores Web (MICROSOFT 2007b).

Para el desarrollo de las actividades del diseño de este trabajo se escogieron dos de las herramientas anteriormente referenciadas, teniendo en cuenta las prestaciones que brindan las mismas y la plataforma de desarrollo con que se decidió construir la aplicación, de modo que se consiga la mayor integración posible entre las herramientas de desarrollo y la metodología que se use para el modelado de diseño de software. Para el modelado del diseño fue utilizado el **Enterprise Architect 6.5**, el mismo posee un ambiente de trabajo amigable, da soporte al modelado de diseño basado en la metodología RUP, posibilita una alta productividad en el trabajo, posee gran aceptación en los clientes que lo utilizan en el mercado y su coste de adquisición (Patente para el desarrollo) es uno de los más económicos. El **Enterprise Architect 6.5**, presenta prestaciones productivas con garantía de una alta calidad, en la elaboración de diagramas de clases, diagramas de iteración, el modelado de la arquitectura y la generación de código C#, así como la reingeniería de software.

Para el diseño del modelo de datos preliminar se usó ER/Estudio, herramienta de las mejores del mundo para estas actividades, pertenece al fabricante Embarcadero, es propietaria y sus prestaciones permiten un diseño productivo, con capacidad de integración con casi todos los gestores de bases de datos existentes en el mundo. La utilización de herramientas propietarias en las actividades de modelado de diseño de estos módulos se justifica por el contrato del proyecto con el cliente y el ajuste de los presupuestos para la adquisición de patentes.

Patrones de diseño. Aplicación de Patrones y estilos arquitectónicos

Los patrones de diseño surgen ante la necesidad de la reutilización de diseños y de soluciones a problemas frecuentes encontrado por los ingenieros. Con la reutilización se consigue la reducción de tiempos y la disminución del esfuerzo de mantenimiento, esto trae consigo mayor eficiencia y consistencia en el diseño de la solución. Existen varias formas de hacer reutilización de diseño en las tareas del diseño de sistema como por ejemplo el diseño basado en Componentes, Frameworks, Objetos distribuidos o los Patrones de diseño. Para crear reutilización en el diseño no necesariamente se necesita de un

consumidor de reutilización, porque el propio tiempo de desarrollo convertirá a el equipo de diseño en consumidores de diseño reutilizable anteriormente desarrollado.

Los patrones de diseño surgen primeramente en el campo de la arquitectura y la ingeniería civil, con el desarrollo de la complejidad de los problemas a automatizar, y con la evolución de los paradigmas de desarrollo fueron extrapolados a la informática (TEJADA 2002), ya bien como estilos arquitectónicos o patrones de arquitectura, o bien como patrones de diseño, que son los que ocupan el objetivo de este epígrafe por su importancia en el diseño de software.

Cada patrón describe un problema que ocurre una y otra vez en nuestro entorno, para describir después el núcleo de la solución a ese problema, de tal manera que esa solución pueda ser usada más de un millón de veces sin hacerlo ni siquiera dos veces de la misma forma (TEJADA 2002).

Las características que presentan los patrones son las siguientes:

- Son soluciones concretas. Proponen soluciones a problemas concretos, no son teorías genéricas.
- Son soluciones técnicas. Indican resoluciones técnicas basadas en Programación Orientada a Objetos (POO). En ocasiones tienen más utilidad con algunos lenguajes de programación y en otras son aplicables a cualquier lenguaje.
- Se utilizan en situaciones frecuentes. Ya que se basan en la experiencia acumulada al resolver problemas reiterativos.
- Favorecen la reutilización de código. Ayudan a construir software basado en la reutilización, a construir clases reutilizables. Los propios patrones se reutilizan cada vez que se vuelven a aplicar.
- El uso de un patrón no se refleja en el código. Al aplicar un patrón, el código resultante no tiene por que delatar el patrón o patrones que lo inspiró. No obstante últimamente hay múltiples esfuerzos enfocados a la construcción de herramientas de desarrollo basados en los patrones y frecuentemente se incluye en los nombres de las clases el nombre del patrón en que se basan facilitando así la comunicación entre desarrolladores.

Graig Larman en su libro UML y Patrones (LARMAN 1998) menciona un grupo de patrones relacionados con el diseño de software, los cuales son llamados patrones GRASP (General Responsibility Asignment Software Patterns) y describen los principios fundamentales de la asignación de responsabilidades a objetos, expresados en forma de patrones, algunos de ellos tiene mucha relación con los problemas básicos del diseño y deben su nombre a estos. La asignación correcta de las

responsabilidades en el diseño orientado a objetos garantiza la alta cohesión de las clases y el bajo acoplamiento de los mismos, lo que posibilita más extensibilidad, adaptabilidad y menos tiempo para el mantenimiento del diseño, por ello estos patrones son parejas de problema/solución con un nombre, que codifican buenos principios y sugerencias relacionados frecuentemente con la asignación de responsabilidades. Estos patrones pueden ser aplicados preferiblemente durante la preparación de un diagrama de interacción o colaboración ya que es en esta actividad del diseño donde el diseñador estructura las responsabilidades y la colaboración entre los componentes de diseño (clases).

Los patrones GRASP son los siguientes:

1. Experto
2. Creador
3. Alta Cohesión
4. Bajo Acoplamiento
5. Controlador

En el capítulo anterior se abordaron aspectos relacionados de una forma u otra con estos cinco patrones por lo que a continuación se relacionará información dirigida a como los mismos tienden a solucionar los problemas que se originan en el diseño.

El patrón ***experto*** trata de resolver el problema de diseño de como asignan las responsabilidades en el diseño orientado a objetos. Un modelo de clase puede definir docenas y hasta cientos de clases de software, y una aplicación tal vez requiera el cumplimiento de cientos o miles de responsabilidades.

Para dar solución a esta problemática el patrón experto plantea asignar una responsabilidad al experto en información: la clase que cuenta con la información necesaria para cumplir la responsabilidad (LARMAN 1998).

El patrón ***Creador*** trata de resolver el problema de diseño de quién debería ser responsable de crear una nueva instancia de alguna clase. La creación de objetos es una de las actividades más frecuentes en un sistema orientado a objetos. En consecuencia, conviene contar con un principio general para asignar las responsabilidades concernientes a ella. El diseño, bien asignado, puede soportar un bajo acoplamiento, una mayor claridad, el enclapsulamiento y la reutilizabilidad.

Para dar solución a esta problemática el patrón creador plantea asignarle a la clase B la responsabilidad de crear una instancia de clase A en uno de 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 a A cuando este objeto sea creado (así que B es un Experto respecto a la creación de A).
- B es un creador de los objetos A.

Si existe más de una opción, prefiera la clase B que agregue o contenga la clase A.

Este patrón es sumamente básico, y su solución en muchos casos no cubre los problemas que se presentan actualmente en el diseño de persistencia de software, pero es la base de todas las demás familias de patrones y framework relacionados con la creación de objetos; por su importancia ha sido incluido en este trabajo (LARMAN 1998).

El patrón ***Bajo Acoplamiento*** trata de resolver el problema de diseño de cómo dar soporte a una dependencia escasa y a un aumento de la reutilización.

Para dar solución a esta problemática el patrón bajo acoplamiento plantea asignar una responsabilidad a cada clase o grupo de clases según su contexto que no involucre recursos concurrentes de otras clases que no están enmarcadas en este contexto de modelación del problema, para mantener de este modo bajo el acoplamiento en el diseño (LARMAN 1998).

El patrón ***Alta Cohesión*** trata de resolver el problema de diseño de cómo mantener la complejidad dentro de límite manejable.

Para dar solución a esta problemática el patrón alta cohesión plantea asignar una responsabilidad a cada clase o grupo de clases según su contexto que no asuma responsabilidades fuera de su dominio de modelación, si una clase esta modelando la gestión de personas, no debe recibir la responsabilidad de autenticar un usuario, esa función le debe corresponde a otra clase, de esta forma se puede lograr una alta cohesión en el diseño del sistema. La alta cohesión al igual que el bajo acoplamiento facilita una diseño mantenible, fácil de reutilizar y adaptar, más legibilidad para los programadores y diseñadores en general; la extensibilidad y flexibilidad del diseño aumentan, ya que las clases usan solo lo que necesitan y en su mayoría son recursos propios, por otro lado, cada cual hace lo que le corresponde hacer según la parte del problema que esta modelando (LARMAN 1998).

El patrón ***Controlador*** trata de resolver el problema de diseño de quién debería encargarse de atender un evento del sistema. Los eventos del sistema pueden ser varios en un mismo formulario, y cada uno

demanda de una validación de precondiciones definidas en la especificación de los requerimientos, en ocasiones suele ser un problema atenderlos a todos de forma consistente, y que ante la repetición de algún de ellos dentro del mismo formulario no exista la necesidad de reprogramar el mismo.

Un Controlador es un objeto de interfaz no destinada al usuario que se encarga de manejar un evento del sistema. Define además el método de su operación.

Para dar solución a esta problemática el patrón Controlador plantea asignar la responsabilidad del manejo de un mensaje de los eventos de un sistema a una clase que represente una de las siguientes opciones:

- el "sistema" global (controlador de fachada).
- la empresa u organización global (controlador de fachada).
- algo en el mundo real que es activo (por ejemplo, el papel de una persona) y que pueda participar en la tarea (controlador de tareas).
- un manejador artificial de todos los eventos del sistema de un caso de uso, generalmente denominados **"Manejador<NombreCasodeUso>"** (controlador de casos de uso).
- Utilice la misma clase de controlador con todos los eventos del sistema en el mismo caso de uso.

Corolario: Nótese que en esta lista no figuran las clases "ventana", "aplicación", "vista" ni "documento". Estas clases ***no*** deberían ejecutar las tareas asociadas a los eventos del sistema; generalmente las reciben y las delegan al controlador.

Este patrón ha sido superado ya por otros patrones y estilos arquitectónicos más complejos que cubren este propósito, pero por ser la inspiración de todas las soluciones relacionadas con la problemática que existen hoy y el punto de partida para desarrollar un buen diseño, ha sido referenciado en este trabajo (LARMAN 1998).

Si se desea abundar más en los patrones GRASP consultar (LARMAN 1998).

Los patrones GRASP sin dudas son la base de inspiración de otros patrones más poderosos y que han ganado muchísima reputación en el estado del arte del diseño de software actual, además de consolidar una norma de consulta perenne para desarrollar un diseño, eficiente, flexible, adaptable, reutilizable y eficaz.

En el diseño del software también surgen otro gran número de problemas como son los relacionados con el uso de memoria, la creación de trazas de operaciones, la creación de familias de objetos, o incluso

la gestión y configuración de entornos entre otros que han constituido el punto de partida para muchos diseñadores, teniéndose en la actualidad una gran gama de patrones a utilizar con estos fines, por el grado de encapsulamiento que da la solución al problema, por la limpieza y facilidad de entendimiento, además de la calidad y del aporte creativo al diseño, se incluyen en este trabajo el grupo de patrones conocidos como Patrones del "Gang of Four".

Los autores del trabajo "Design Patterns: Elements of reusable object-oriented software" (ERICH GAMMA 1995), proponen un número grande de patrones agrupados en tres categorías, que resuelven casi todas las problemáticas básicas del diseño de software, a estos patrones se les conoce en el mundo cómo Patrones del "Gang of Four" y es muy difícil ver hoy un diseño de software que no haya reutilizado o se haya inspirado una de las propuestas de solución a problemas que se plantee en los patrones antes mencionados.

A continuación se hace referencia a las categorías en la que se agrupan estos patrones, si se desea profundizar cada uno de estos patrones ver (ERICH GAMMA 1995). No es objetivo de este trabajo hacer un análisis detallado de las características de cada patrón, sino fundamentar el modo en que estos fueron combinados y aplicados en la solución propuesta:

1. **Creational Patterns.** Esta categoría agrupa los patrones que dan solución a los problemas de procesos de instanciación, los mismos proponen formulas independientes para la creación, composición y representación de objetos. Para ello usan conceptos de la programación orientada a objetos como interfaces, herencia, polimorfismo, la delegación de responsabilidades, y los principios de las estructuras de datos, para hacer más eficientes estos procesos. Los patrones que se encuentran en esta categorías son Abstract Factory, Builder, Factory Method, Prototype, Singleton.
2. **Structural Patterns.** Esta categoría agrupa los patrones que dan solución a los problemas de creación de estructuras de datos formadas por clases que se agrupan para resolver una parte de la modelación del problema en cuestión. Estos patrones usan herencia e interfaces para componer las estructuras de implementación. Su uso principal está relacionado con el desarrollo de librerías de clases de trajo independientes, de modo que se obtengan diseños dinámicos capaces de modificarse en tiempo de ejecución. Estos son Adapter, Bridge, Composite, Decorador, Facade, Flyweight, Proxy.

3. **Behavioral Patterns.** Esta categoría agrupa los patrones que dan solución a los problemas de modelación de algoritmos y la asignación de responsabilidades entre objetos en el diseño, toca de cerca el modo de interconexión entre los objetos, el traspaso de flujo, y trata de modelar cada llamada a una nueva responsabilidad de la estructura como una operación abstracta o interna. Estos son Chain of Responsibility, Command, Interpreter, Iterator, Mediator, Memento, Observer, State, Strategy, Template Method, Visitor.

En los módulos Tratamiento Farmacológico y Configuración del proyecto HyperWeb algunos de los patrones tratados con anterioridad, han sido motivo de inspiración a la hora de diseñar un comportamiento determinado de la solución, o dar una solución concreta. Por ahora solo concierne explicar el motivo del uso y la estructura de clases que fue concebida para modelar cada uno de los patrones utilizados o variante de estos en diferentes casos puntuales de la solución en diseño.

Estructura arquitectónica

Para el desarrollo de los módulos de Tratamiento Farmacológico y Configuración se han definido desde el punto de vista arquitectónico un grupo de aspectos:

Se utilizarán ficheros en formatos XML para realizar configuraciones dinámicas como es el caso de las Reglas para evaluar las pruebas en el proceso de diagnóstico.

La arquitectura de la aplicación que integran los módulos de Tratamiento Farmacológico y Configuración es Web, la misma estará en comunicación con el servidor de base de datos SQL Server 2000 a través ADO.NET. Está concebido que los clientes se conecten al servidor Web Internet Information Server 6.0 (IIS) el cual será el encargado de procesar sus pedidos.

Los módulos desarrollados (Tratamiento Farmacológico y Configuración) son subsistemas del sistema HyperWeb, los mismos permiten obtener resultados medibles y completos de los procesos que ellos modelan. Ver Figura 4 donde se representan las responsabilidades de los subsistemas con los casos de uso.

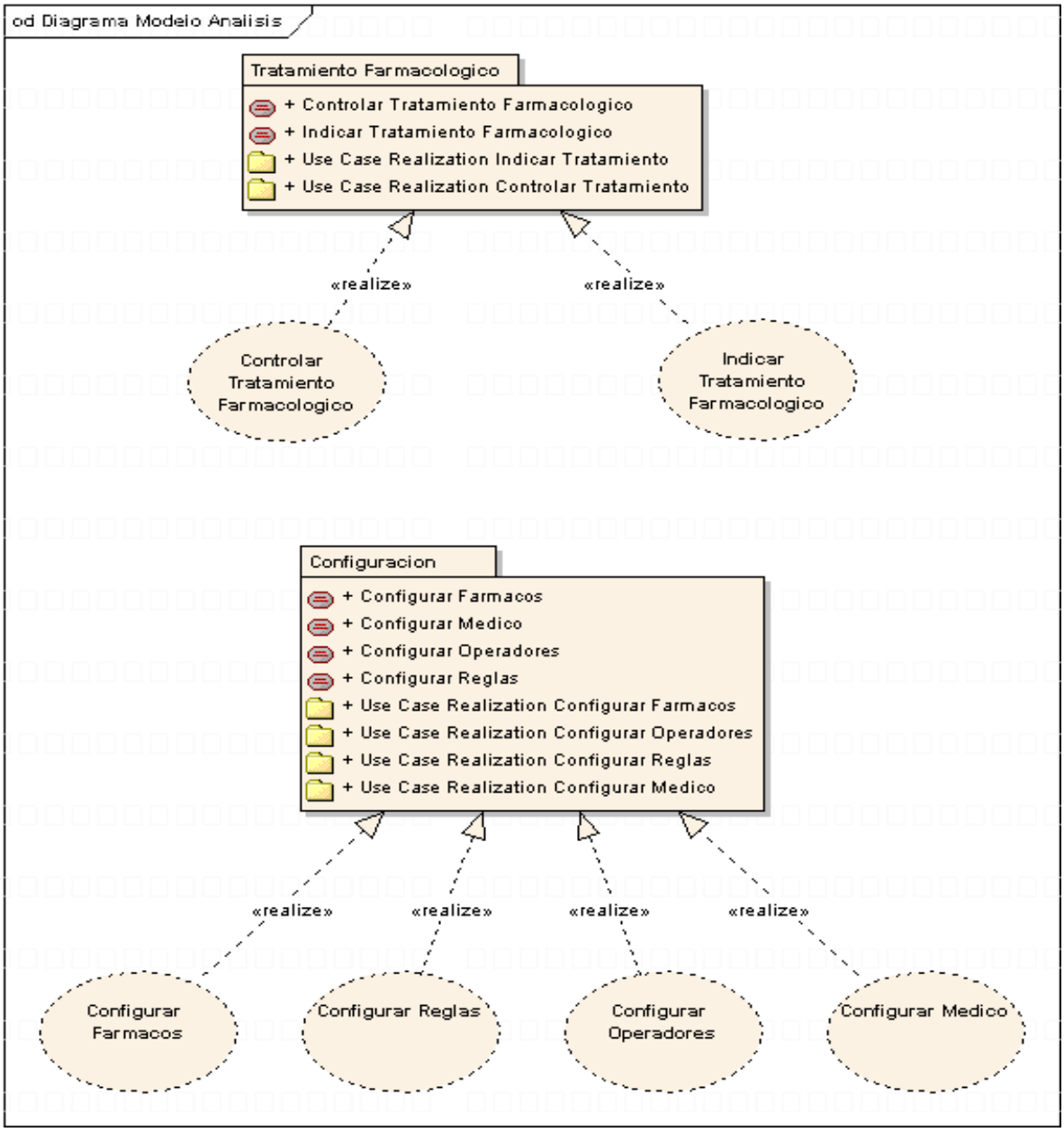

Figura 4 Relación Subsistema-Realización de caso de uso

En (Anexo 2 Distribución de casos de uso por subsistemas) se muestra la distribución de los casos de uso por subsistemas.

La arquitectura taxonómica de cada modulo en el proyecto HyperWeb ha sido modelada en una variante basada en el modelo de tres capas, como se aprecia en la Figura 5 y Figura 6. La capa de

presentación constituida por los formularios Web ve la capa de negocio y se comunica con esta empleando objetos de la capa de negocio y parámetros de consulta representados en tipos de datos del lenguaje C#. La capa de negocio esta compuesta de las entidades y los gestores de lógica, no tiene acceso a la capa de presentación, y se comunica con esta capa a través de las entidades del negocio y los tipos de datos abstractos implementados en la tecnología de desarrollo .NET para el transporte de datos entre capas arquitectónicas como dataset[6] o datatable[7]. Por otra parte la capa de negocio tiene visibilidad sobre la capa de acceso a datos y se comunica con ella a través de las entidades del negocio, mediante objetos Datasets, Datatables o mediante atributos. La capa de acceso a datos solo tiene visibilidad a las entidades de la capa de negocio y es quién implementa las especificaciones del tipo de fuente de persistencia de estas entidades, responsabilizando a clases que modelan función de fábricas de cada una de las clases del negocio que necesitan de persistencia. La Figura 7 muestra un esquema general del modelo de tres capaz usado en la estructura arquitectónica de HyperWeb.

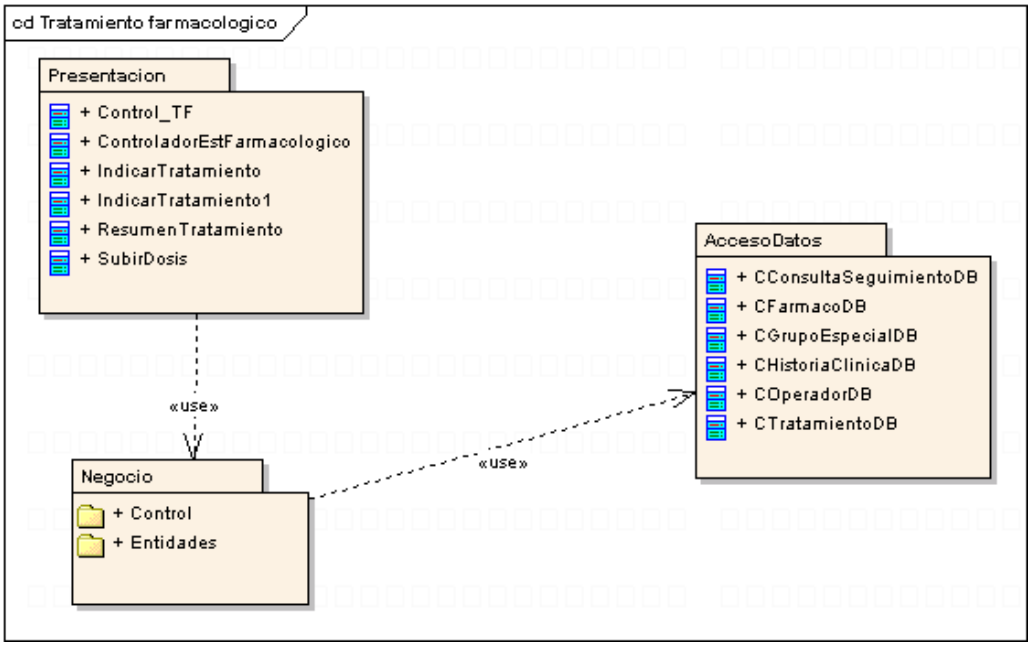

Figura 5 Arquitectura en capas del subsistema Tratamiento Farmacológico

[6] Tipo de dato contenedor de información, permite ser llenado a partir de la lectura de diferentes fuentes de datos. La información contenida en el objeto podrá ser transportada entre las capaz de la arquitectura.
[7] Objeto que los datos de una tabla en memoria.

Esquema de persistencia

La implementación del esquema de persistencia de los módulos de Tratamiento Farmacológico y Configuración, estuvo inspirado en el patrón **Factoría Abstracta**, pero sin la inclusión del uso de interfaces, debido a que se busca mayor rapidez en la etapa de construcción.

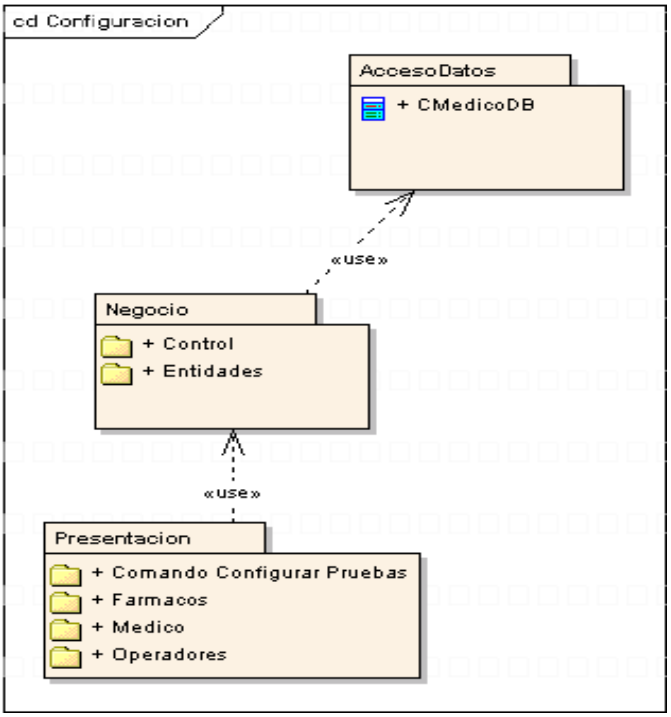

Figura 6 Arquitectura en capas del subsistema Configuración

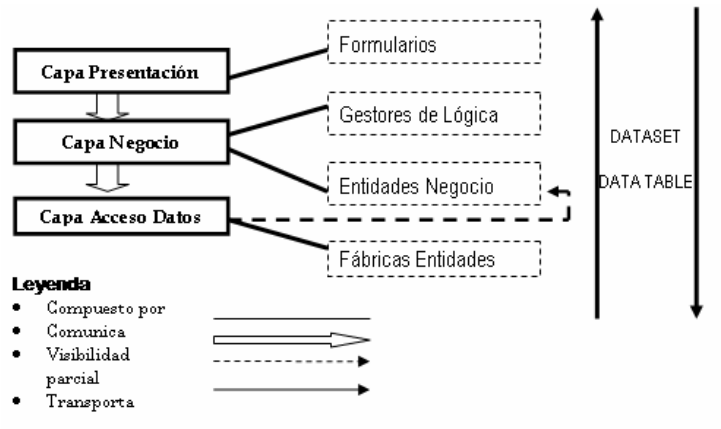

Figura 7 Arquitectura en tres capas

Modelación en el diseño de las reglas del negocio

En los módulos Tratamiento Farmacológico y Configuración de HyperWeb fueron aplicados los patrones GRASP Experto, Bajo Acoplamiento y Alta Cohesión. Se buscó que cada clase incluida en el diseño fuese responsable de los aspectos que le conciernen, y tratando de tener la menor dependencia funcional y de datos posible con otras clases. Esto se pone de manifiesto en la creación de un gestor de negocio por cada proceso del mismo, como son los casos del flujo de tratamiento farmacológico y el Gestor de Tratamiento Farmacológico, así como en la configuración intervienen gestores en relación con el aspecto a configurar, como es el caso de la configuración de los Grupos de Fármacos y Fármacos que intervienen en el tratamiento de un paciente los cuales son manejados por el Gestor de Tratamiento Farmacológico. Así mismo la gestión de los médicos se realiza por el Gestor de Usuarios. En correspondencia con lo antes planteado se resuelven las problemáticas de configuración de las pruebas que se les indican a los pacientes y la configuración de operadores que facilitan la indicación de tratamiento farmacológico. En el diseño de estos módulos se puede ver una colaboración entre gestores en función del grado de experticidad de cada uno, como es el caso del Gestor de Tratamiento Farmacológico y el Gestor de Usuario, que es un gestor que interviene en otros módulos del sistema.

Esquema de gestión de eventos del módulo de Configuración

La gestión de eventos de los escenarios del módulo Configuración, debido a la complejidad de los flujos de navegación de los escenarios, es aplicado un esquema de diseño inspirado en el estilo arquitectónico Modelo Vista Controlador y el patrón Estado, este hibrido disminuye el costo de implementación que requiere el estilo Modelo Vista Controlador (algunos autores lo consideran un patrón (TEJADA 2002)), y facilita la flexibilidad del desarrollo de validación de los formularios Web durante el tiempo de desarrollo.

El mismo está compuesto por una clase controladora asociada a cada formulario, que encapsula todos los eventos de este. La clase controladora de eventos, delega para cada evento la responsabilidad de validar el estado del formulario Web para este evento del sistema a un objeto validador, después de que este realiza la validación cede el paso a la clase controladora para que esta delegue el flujo del evento a la clase controladora de lógica correspondiente a la tarea especifica. En la Figura 8 se puede ver el diagrama de clases del esquema y en la Figura 9 un diagrama de secuencia de un ejemplo genérico que muestra la forma en que interactúan y colaboran las clases que conforman el esquema.

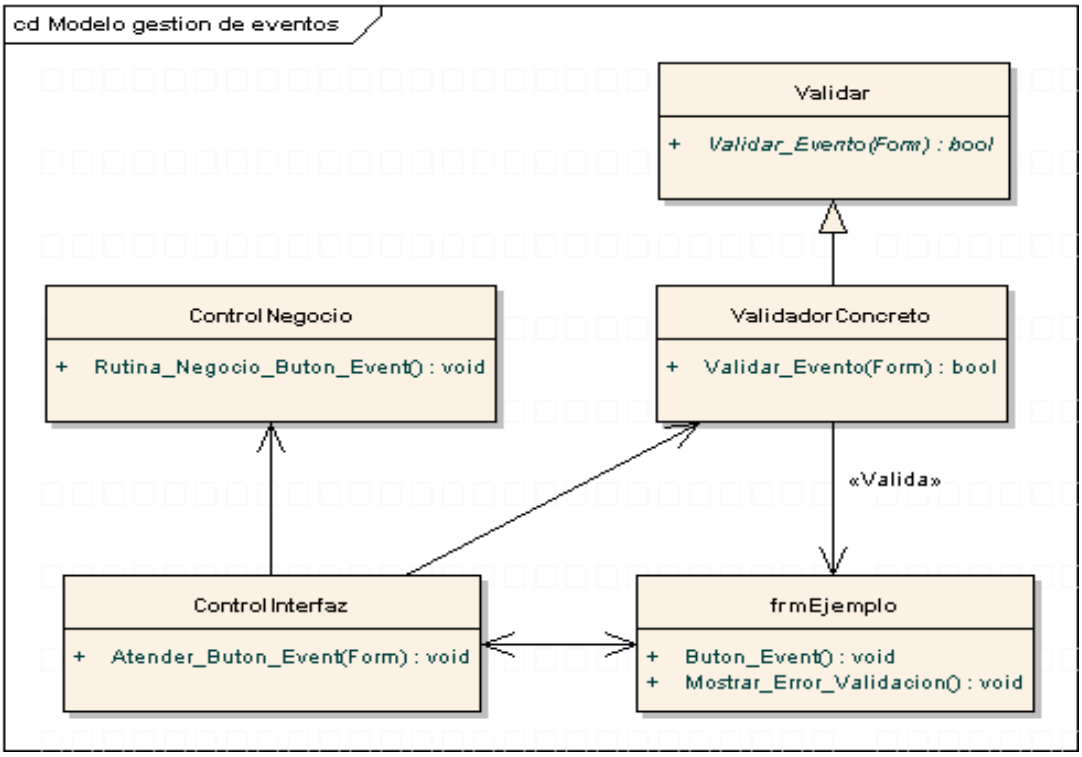

Figura 8 Diagrama de clases genérico entre las clases involucradas en el esquema

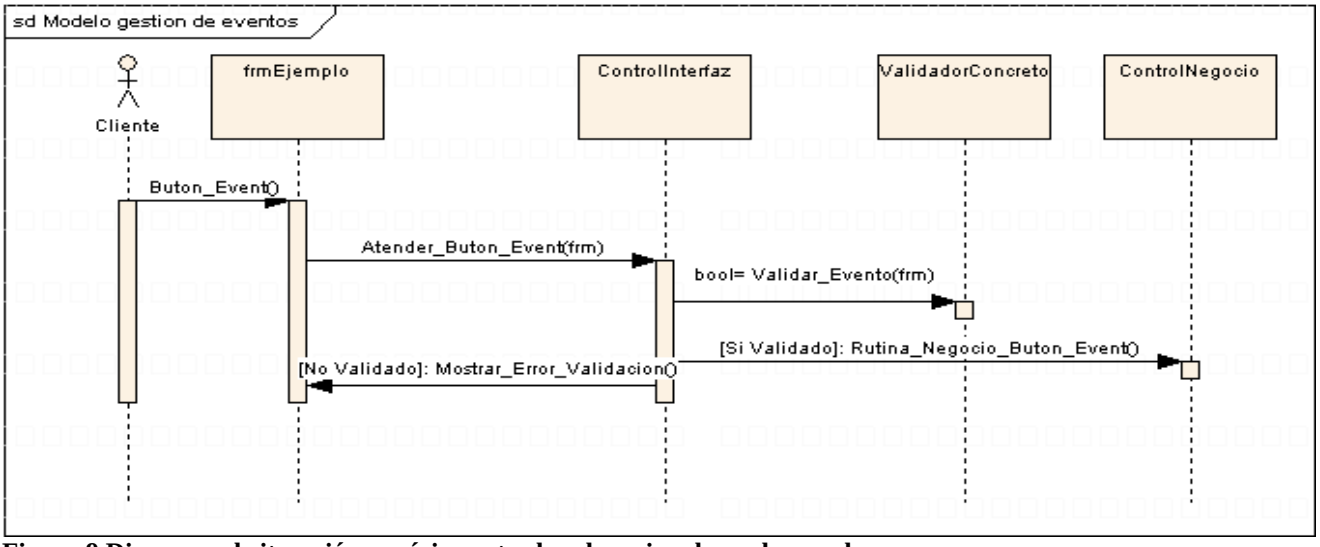

Figura 9 Diagrama de iteración genérico entre las clases involucradas en el esquema

Estándares en la interfaz de la aplicación.

Uno de los principios indispensables para el diseño de interfaces Web es el de reutilización de la experiencia del usuario, el cual consiste en exponer al usuario ante lo conocido. Actualmente está proliferando en demasía el uso de ciertos elementos que dan a la Web mucha flexibilidad y riqueza, en los cuales prima un efecto visual y la realización de todo tipo de piruetas estilísticas sobre la facilidad de uso consiguiendo confundir al usuario. En el entorno Web es particularmente poco razonable crear interfaces que requieran excesivo aprendizaje ya que cualquier dificultad presentada al usuario suele significar una pérdida considerable de legibilidad y flexibilidad. El uso de estructuras de navegación no estándar es el error más grave que se pueda cometer, justificado solo bajo ciertas excepciones.

Teniendo esto en cuenta, para la confección de esta aplicación se sigue de manera general el diseño de páginas con la siguiente estructura (ver Figura 10):

- Encabezado: En esta región se incluye el logotipo representativo de la aplicación y las distintas etiquetas de navegación que serán visibles en dependencia del rol del usuario en ese momento.
- Panel de navegación izquierdo: En este panel se mostrarán generalmente los enlaces de navegación hacia los distintos contenidos de la aplicación.
- Panel de navegación derecho: Cumple el mismo objetivo que el panel izquierdo, solo que será mostrado en situaciones opcionales que requieran la evaluación de múltiples contenidos por parte del usuario.
- Panel de contenido: En este contenedor se despliega toda la información solicitada por el usuario, así como el contenido generado por el sistema.

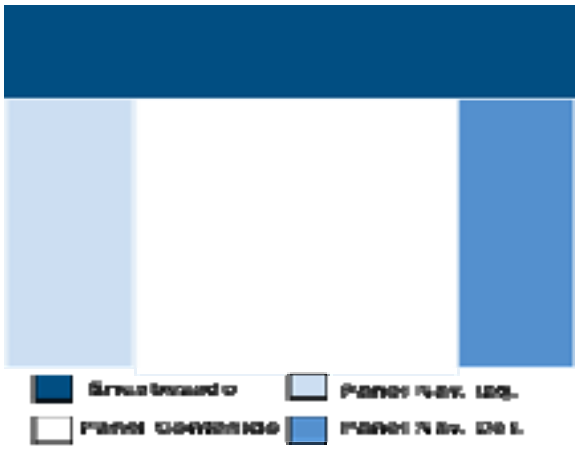

Figura 10 Estructura de las Páginas Web

Además el sistema se apoya en el uso de una hoja de estilo en cascada (CSS) general para todas las páginas de la aplicación, en la que se establecen los estilos de letra para las distintas informaciones que se muestran, el uso de colores, estilo de los controles de entrada, entre otros. De forma general, se evitaron el uso de técnicas no compatibles con otros navegadores como marcos (frames o iframes) o la utilización de lenguajes clientes como Javascript, teniendo en cuenta que una de las opciones que se maneja actualmente para el mantenimiento de la compatibilidad entre navegadores es la no dependencia de scripts no estándar del lado del cliente (ALFREDO MORALES OLIVA 2006).

Conclusiones

A manera de conclusiones se puede decir que existe una amplia diversidad de metodologías de desarrollo de software, herramientas CASE y patrones de diseño que pueden ser adaptables a lo que se desea obtener. Con relación a la herramienta CASE la decisión final no solo está apoyada en el valor monetario que representaría la compra de la licencia sino en la facilidad de uso que brinde esta en contraste con el dominio de la misma. En todo este proceso de selección y aplicación existió un balance real entre las necesidades del cliente, las características del equipo de desarrollo y su productividad bajo las diversas condiciones.

En otro orden de información a partir de patrones y estilos arquitectónicos existentes, se realizó un análisis de los mismos, y se especificó la forma en que estos fueron adaptados o mezclados en el modelo de diseño de los subsistemas Tratamiento Farmacológico, Configuración y Comunes, con el fin de dar solución a problemas de modelado presentes en este trabajo como fueron, el esquema de persistencia de los objetos del negocio, la distribución de responsabilidades según la definición de requerimientos funcionales extraídos de los flujos del negocio, la estructura arquitectónica concebida en capas, y el modelo de gestión de eventos para interfaces compleja utilizados en el modulo de configuración.

Fue tratado además las características del diseño Web que debe soportar la solución.

CAPÍTULO 3

ANÁLISIS Y DISEÑO DE LA SOLUCIÓN

Introducción

En este capítulo se abordan los modelos de análisis y diseño de los módulos Tratamiento farmacológico y Configuración del proyecto HyperWeb. Así como una descripción detallada de los algoritmos y esquemas propuestas en la solución de la problemática que atañe cada modulo. Finalmente se evalúa la solución propuesta mediante instrumentos que se crean inspirados en métricas para el control de la calidad del diseño.

Modelo de Análisis

Los módulos Tratamiento Farmacológico y Configuración han sido diseñados como dos subsistemas, con características propias cada uno, y cuyos usuarios finales están representados por roles independientes. La Figura 11 muestra esta relación.

Proceso de indicación y control de tratamiento farmacológico

Indicar y controlar el tratamiento farmacológico a un paciente requiere apoyarse básicamente en un conjunto de Grupos Especiales a los que pertenece el paciente por su condición física y en un conjunto de riesgos que pueden existir en relación a la forma de vida del paciente (CABALLERO 2006; CHOBANIAN *et al.* 2003; PÉREZ CABALLERO *et al.* 2004).

Los grupos especiales no son más que agrupaciones poblacionales determinadas por las condiciones físicas de estas personas, un ejemplo de ello es el grupo especial Diabetes en el que se incluyen todas las personas que padecen de Diabetes Melitus.

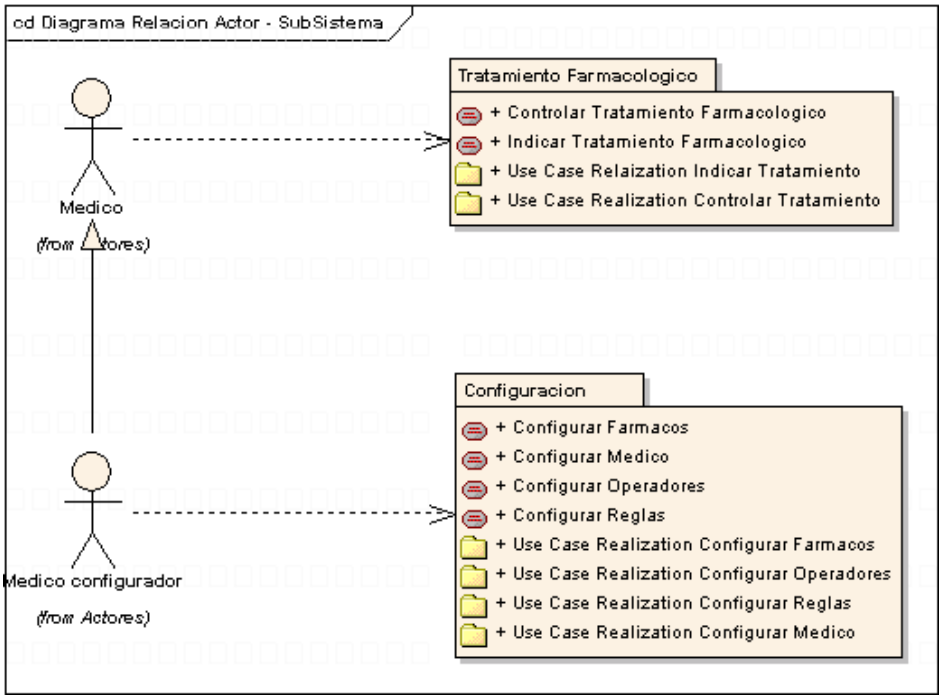

Figura 11 Representación en subsistemas de los módulos tratamiento Farmacológico y Configuración, y su relación con los actores del sistema.

La selección de determinados fármacos en la indicación de tratamiento para estas situaciones especiales está determinada por la pertenencia del paciente a un grupo o grupos especiales. La información que recoge esa correspondencia entre grupos poblacionales y grupos de fármacos a indicar está basada en resultados favorables de ensayos clínicos.

Durante el tratamiento podría ser necesaria una combinación de fármacos en determinadas circunstancias. Mientras que otras consideraciones para la elección son medicaciones ya en uso, tolerancia, y cifras de PA deseadas.

A partir del diagnostico y evaluación clínica de un paciente se determina a que grupos puede asociar el mismo. Para el desarrollo del módulo de Tratamiento Farmacológico partimos de un conjunto de Grupos Especiales (PÉREZ CABALLERO *et al.* 2004):

1. ANCIANOS
2. NIÑOS (Representan a los niños que padecen HTA, pero no se tratan por el módulo propuesto)
3. NEGROS
4. DIABETICOS
5. DISLIPIDEMICOS

6. EMBARAZADAS (Representan a las embarazadas que padecen HTA, pero no se tratan por el módulo propuesto)
7. HVI
8. EPOC
9. INSUFICIENCIA RENAL (Representan a las personas que padecen de Insuficiencia Renal y que presentan alteraciones en la Tensión Arterial, pero no se tratan por el módulo propuesto)
10. HIPERURICEMICOS
11. ENF. VASCULAR PERIFERICA
12. ENF. CEREBROVASCULAR
13. ENF: ARTERIA CORONARIA

Cubrir la explicación de las condiciones que deben existir para pertenecer a los grupos especiales antes referidos no es un objetivo de este trabajo dado que el mismo pertenece al proceso de Diagnostico y Evaluación Clínica del paciente, para encontrar detalles al respecto consultar (CABALLERO 2006; CHOBANIAN *et al.* 2003; PÉREZ CABALLERO *et al.* 2004).

Cada uno de estos grupos especiales identificados propone un orden de preferencia con respecto a los grupos de fármacos que deben ser indicados y cuales grupos de fármacos están contraindicados. Un ejemplo es el caso de los ancianos donde las drogas de elección son Diuréticos Tiazídicos, Anticálcicos de acción retardada, Inhibidores de la ECA, Beta-Bloqueadores (sobre todo si hay cardiopatía isquémica asociada) en combinación con diuréticos a baja dosis.

Para obtener más detalles de el resto de los grupos especiales dirigirse a (CABALLERO 2006).

En los casos en los que un paciente pertenece a varios grupos especiales se tienen que eliminar los grupos de fármacos contraindicados y luego balancear los propuestos por cada grupo especial en relación al orden de los mismos y a cuantos fármacos deben indicársele al paciente.

No se desglosa el orden propuesto en la indicación de grupos de fármacos y grupos contraindicados en relación con cada uno de los grupos especiales porque el módulo de Configuración recoge una sección de Configuración de Operadores en la que se pueden crear nuevos grupos especiales y describir como influyen en el tratamiento farmacológico. Es decir, esta configuración dinámica de los aspectos necesarios para el Tratamiento Farmacológico no es responsabilidad de los autores de este trabajo sino de un médico.

Para lograr indicar y controlar se siguió el siguiente algoritmo de tratamiento propuesto en Figura 12 que aparece en (CABALLERO 2006). Hay que señalar que los grupos de fármacos que se indican en el algoritmo están determinados porque al paciente que se le aplican no pertenece distintivamente a ningún grupo especial, de lo contrario se seguiría el orden que determinarían los grupos especiales a los que pertenece la persona enferma.

La definición de los conceptos asociados a “Clasificación” y “Riesgos” que aparecen en el algoritmo aparece en (Anexo 3 Estratificación del riesgo cardiovascular en Hipertensión Arterial).

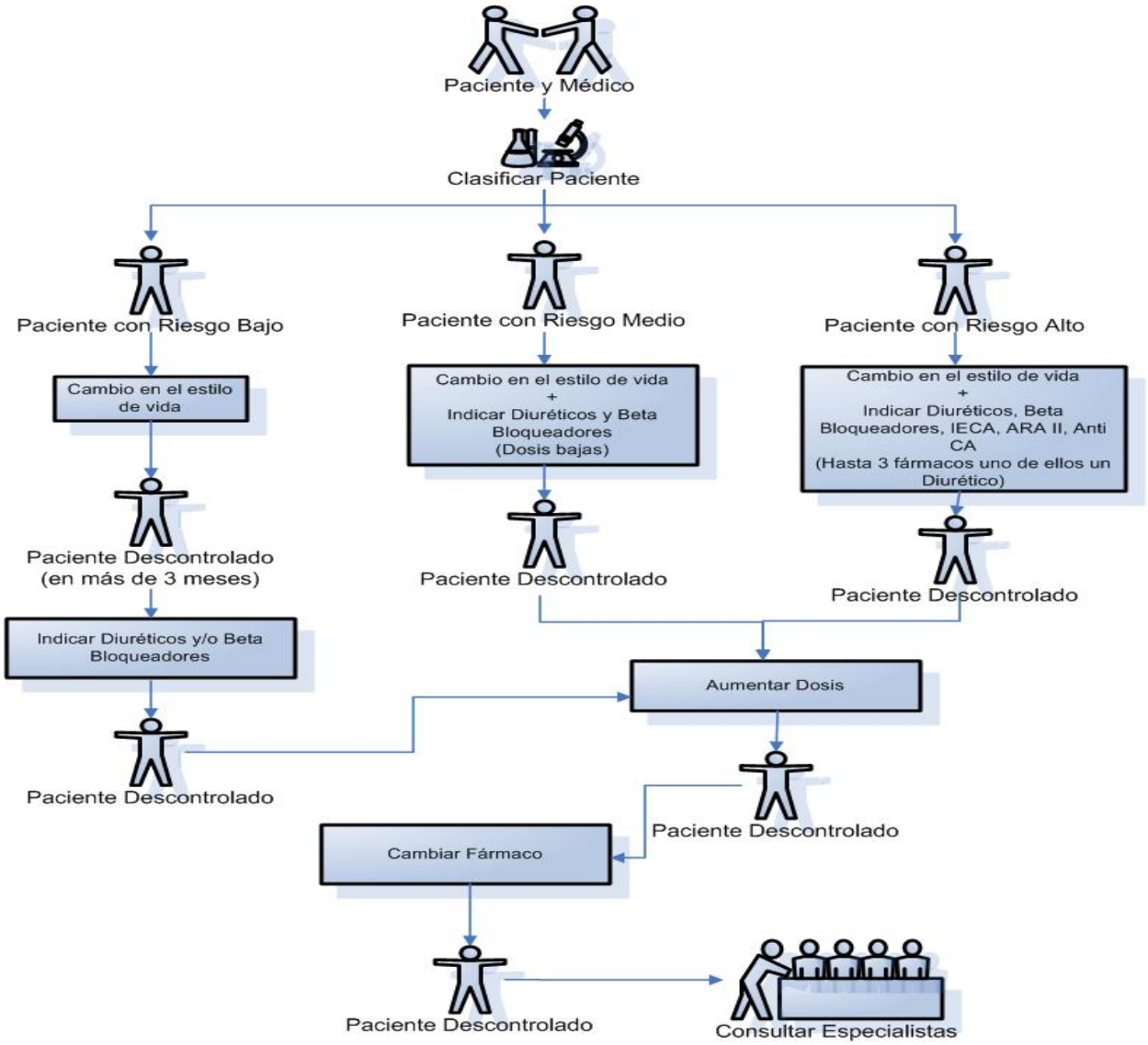

Figura 12 Algoritmo de tratamiento del paciente hipertenso según sus riegos

Capítulo III. Análisis y diseño de la solución

El proceso de control del tratamiento indicado a un paciente puede ser complejo dado que la mayoría de los hipertensos requieren dos o más medicaciones antihipertensivas para conseguir sus objetivos de PA. Las tres opciones posibles durante el control en caso de no estarse logrando una adecuación al tratamiento por parte del paciente son: subir dosis (hasta la dosis máxima de acuerdo al fármaco), cambiar el fármaco y aumentar el número de fármacos (solo hasta 3 fármacos). Las opciones se deben escoger preferentemente en el orden en el que se mencionan.

Un segundo fármaco de diferente clase debería introducirse cuando la monoterapia en dosis adecuadas falla para conseguir el objetivo de PA. Cuando la PA es mayor de 20/10 mmHg sobre el objetivo, se debería considerar iniciar la terapia con dos fármacos, bien como prescripciones separadas o combinaciones en dosis fijas. La iniciación de la terapia farmacológica con más de un agente puede incrementar la posibilidad de conseguir el objetivo de forma oportuna. El uso de fármacos genéricos o combinaciones de fármacos deberían considerarse para reducir el costo de la prescripción. Estas combinaciones se pueden hacer solo si los fármacos son sinérgicos (no producen efectos no deseados al combinarse dos fármacos). Hay que considerar que efectos secundarios pueden producirse al usar determinados fármacos por lo que cuando se modifique el tratamiento debe tenerse en cuenta lo antes mencionado para evitar indicar un fármaco que produjo en el paciente un efecto no esperado.

El diseño propuesto en este trabajo para el módulo Tratamiento Farmacológico fue concebido para que le proponga al médico que grupo de fármacos debe indicar y cuales no de acuerdo con las características del paciente y considerando todo los aspectos médicos antes mencionados (los mismos se desglosan en (CABALLERO 2006; CHOBANIAN *et al.* 2003; PÉREZ CABALLERO *et al.* 2004)). Sin embargo la decisión de indicarle un determinado tratamiento a un paciente queda en manos del médico, él decide si acepta la propuesta hecha por el módulo de Tratamiento Farmacológico o la modifica. Para lograr que el sistema proponga un tratamiento desde el punto de vista del diseño se empleó un algoritmo que se apoya en los Métodos de Solución de Problemas incluidos entre las técnicas de Inteligencia Artificial existentes. El algoritmo se describe en el siguiente epígrafe.

Los medicamentos hipotensores más usados universalmente por haber resistido las pruebas terapéuticas en estudios multicéntricos que han incluido miles de pacientes con Hipertensión Arterial y cuyos resultados se registran en los más importantes metanálisis divulgados en la literatura médica y que por ello son considerados de primera línea en el tratamiento de la Hipertensión Arterial, son: *los diuréticos,*

los betabloqueadores, los bloqueadores de los canales del calcio, los inhibidores de la enzima convertidora de la angiotensina y más recientemente los antagonistas de los receptores de la angiotensina II. Otros medicamentos como los alfabloqueadores, los simpaticolíticos centrales, los antagonista adrenérgicos periféricos y los vasodilatadores directos, se consideran de segunda o tercera línea en el tratamiento de la Hipertensión Arterial y algunos de ellos reservado para situaciones muy específicas (CABALLERO 2006). Ver más detalles en (Anexo 4 Medicamentos más usados en el Tratamiento de la HTA) extraído de (CABALLERO 2006).

Los grupos de fármacos sinérgicos aparecen en (Anexo 5 Combinaciones de fármacos para el tratamiento de la HTA)**.**

Uso de los operadores en la indicación del Tratamiento Farmacológico

Para la modelación del proceso de indicación de tratamiento farmacológico se utilizan la técnica de Inteligencia Artificial conocida como Métodos de Solución de Problemas. En este marco se definió que a través de Operadores se podía representar el hecho de que en dependencia de los Grupos Especiales a los que pertenece el paciente esta dado el tratamiento de los mismos. Partiendo del concepto de que un Operador representa un grupo o grupos especiales existentes en los pacientes.

Ejemplos de grupos: Ancianos, Negros, Diabéticos, Dislipidemia.

Las características del operador consisten en un peso, tipo de medicamentos a discriminar, y propuesta ordenada de medicación.

Se infiere que la función del operador consiste en discriminar y organizar la propuesta de la medicación ideal para cada paciente, según las características propias que describe el operador. El resultado final de tratamiento, es el subconjunto de grupos de fármacos resultante después de aplicado cada uno de los operadores según las peculiaridades del paciente en estudio.

Este proceso ocurre a partir del conjunto inicial de todos los tipos de medicamentos registrados en el sistema y que pueden ser usados para patologías relacionadas con la HTA.

El resultado de haber evaluado los operadores necesarios según las características del caso de estudio, es el camino que describe la vía al nodo final, de modo que de aquí obtenemos la explicación del porque del tratamiento propuesto por el modelo, y la hoja de esa rama de solución descrita, seria el conjunto resultante de tipos de medicamentos a indicar en un orden definido por el peso que se fue acumulado a

partir de la evaluación de los operadores. Para esto se utiliza la formula de ponderancia definida como:

$$O = \sum 1/\left(2^{pos-1}\right) \quad \text{si } pos \geq 1$$
$$O = 0 \qquad OtroCaso$$

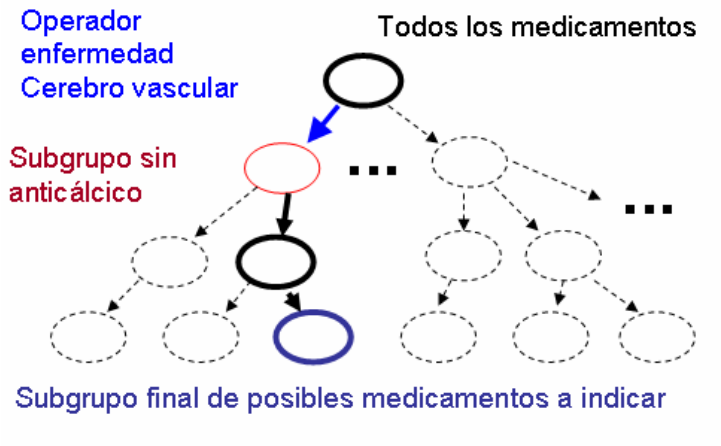

Figura 13 Mecanismo de aplicación de los Operadores

La principal ventaja que trae este algoritmo es que a partir de ahora el sistema desde el punto de vista médico nunca quedará obsoleto en cuanto a estado de arte de patrones de decisión para el tratamiento farmacológico, es decir si mañana apareciese una nueva característica que influyera en la decisión para el tratamiento de la HTA, con su respectivo grupo de medicamento asociado, simplemente se agregaría desde el módulo de Configuración a las bases de conocimiento donde deben ser guardados los operadores y este al arrancar el proceso de evaluación incluiría los nuevos operadores.

Análisis. Realización de los casos de uso

En el análisis, la realización de los casos de uso centra su objetivo en modelar un algoritmo para cada caso de uso del proceso de negocio de la problemática. El modelo de análisis es la entrada del modelo de diseño, de ahí la importancia que reviste desarrollar el mismo con la mayor calidad posible. No es objetivo del análisis hacer un cubrimiento total de los requerimientos del sistema, se trata más bien de hacer un acercamiento preliminar a los conceptos y funcionalidades que sirvan de base para posteriores tareas de diseño.

En el (Anexo 6 Diagramas de clases del Análisis) se muestran los diagramas de clases del análisis asociado cada caso de uso que se recoge en la solución.

En el (Anexo 7 Diagramas de Interacción) se muestran los diagramas de interacción más importantes asociado a los caso de uso que se recogen en la solución.

Modelo de Diseño de sistema

El modelo de diseño de los módulos Tratamiento Farmacológico y Configuración del proyecto de software HyperWeb representa una traza cartesiana del modelo de análisis. En este sentido es meritorio aclarar que siendo Web la arquitectura de estos módulos, el esteriotipo del diagrama Web de esta aplicación no es el soportado por UML 1.0, sino por la versión de UML incluida en el Enterprise Architect 6.5 que es el UML 2.0, el mismo cubre un nuevos esteriotipo Web (en el caso de la tecnología en que se concibe desarrollar este trabajo ASP.NET 1.2 el estereotipo asociado es el mostrado en la Figura 14**)**, relacionado sobre todo con tecnologías que ya se han vuelto estándares internacionales por su nivel de desarrollo y por su impacto en el área del desarrollo Web, como es el caso de Asp.net 1.2, 2.0 o Java. En este sentido se ha logrado unificar en un formulario Web la gestión de los tres elementos que históricamente habían caracterizado la arquitectura de las aplicaciones Web, estos son los casos de las páginas del servidor (Server Page), páginas clientes (Client Page) y los formularios (HTML Form). Esta nueva arquitectura de ASP>NET, brinda mayor productividad en el desarrollo, más abstracción del desarrollador ante los principios de la Web, ya que puede modelar y desarrollar sus aplicaciones en gran medida como si fuesen aplicaciones de escritorio tradicionales. Por otra parte permite gestionar mejor el precompilado del código que debe correr en el servidor y facilita un nivel de integración mayor a las arquitecturas Web.

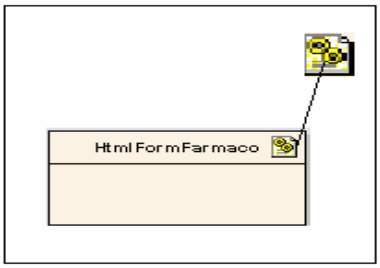

Figura 14 Esteriotipo soportado por UML 2.0 para paginas asp.net. <<ASP PAGE>>.

Pautas de diseño

Entre las pautas de diseño se establecieron los siguientes criterios.

- Clases entidades. Son las clases que representan conceptos o construcciones abstractas del sistema que se esta modelando. Más frecuentemente usadas para cargar y transportar información que puede ser o no persistente. Otras de sus funciones también recae en modelar las operaciones relacionadas únicamente con las acciones del dominio que estas abstraen. La pauta acordada fue poner una C delante del nombre de la clase <<CNombreDeLaClase>. Ejemplo **CFarmaco**
- Clases Controladoras o Gestoras. Son las clases que gestionan la lógica del negocio, en ellas quedan implementadas los requisitos funcionales. La pauta acordada fue poner una C + Gestor + Nombre clase, que debe estar relacionado con la responsabilidad de la misma. <<CGestorNombreClase>> ejemplo **CGestorTratamiento**.
- Clases acceso datos. Son las clases que abstraen el proceso de persistencia de las clases entidades persistentes. La pauta acordada fue poner una C + Nombre del concepto que modela+ DB << CNombreDelCconceptoDB >> ejemplo **CFarmacoDB.**

Subsistemas de Diseño

El modelo de diseño de los módulos Tratamiento Farmacológico y Configuración contiene 3 subsistemas, donde dos de ellos representan el como implementar los casos de usos modelados en los subsistemas del análisis Tratamiento Farmacológico y Configuración, el tercero es el subsistema Comunes, que agrupa los elementos del diseño que son de uso común para los módulos anteriores, en esta situación se encuentran algunos artefactotes del diseño como gestores y clases auxiliares para la gestión de datos.

Un principio del diseño de estos subsistemas fue darle la responsabilidad de contenedor e iterador de las colecciones de datos a la clase que brinda el FrameWork 1.2 de .Net, el tipo de datos abstracto ArrayList. Por ello las relaciones uno a mucho, son resueltas en el diseño como un atributo colección del tipo ArrayList en la clase del lado uno, y luego se usa el principio de indexados inteligentes, de modo que se pueda acceder a cada elemento de la colección sin la necesidad de crear una recurrencia excesiva de conversión de variable de forma explicita, sino más bien por la abstracción implícita que brindan los indexados del C# 1.2, que es el lenguaje de programación que soporta la tecnología Asp.net 1.2.

La selección del lenguaje C# y la tecnología Asp.Net 1.2 se debió a un requerimiento concreto de la capacidad del grupo de desarrollo y los acuerdos pactados con el cliente. De modo que si se iba a trabajar en una plataforma propietaria como .net, lo más sensato era trabajar sobre el lenguaje natural de la

tecnología, en este caso C# 1.2. Actualmente esta tecnología ha evolucionado en el tiempo y su más reciente versión es Asp.Net 2.0 con el C# 2.0, pero a los efectos del diseño del sistema es irrelevante, pues el diseño asume las mismas características al nivel en que se trabajó, solo cambia en las facilidades de implementación de este diseño, tarea que no es objetivo de este trabajo de diploma.

La Figura 15 muestra la relación de integración de los subsistemas Tratamiento Farmacológico, Configuración, Comunes y el FrameWork 1.2 que porta la tecnología Asp.Net 1.2.

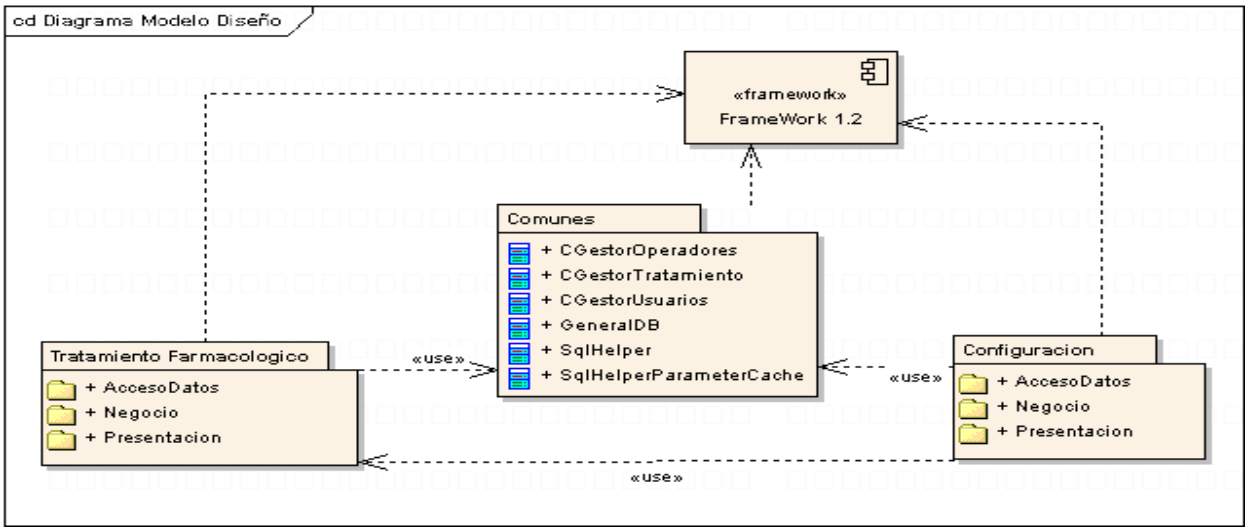

Figura 15 Relación de los subsistemas Tratamiento Farmacológico, Configuración, Comunes y el FrameWork 1.2

La Figura 16 Representa la conformación del subsistema Tratamiento Farmacológico. En el que se representan las capas de Presentación, Negocio, Acceso a Datos y la relación con el FrameWork 1.2 que porta la tecnología Asp.Net 1.2.

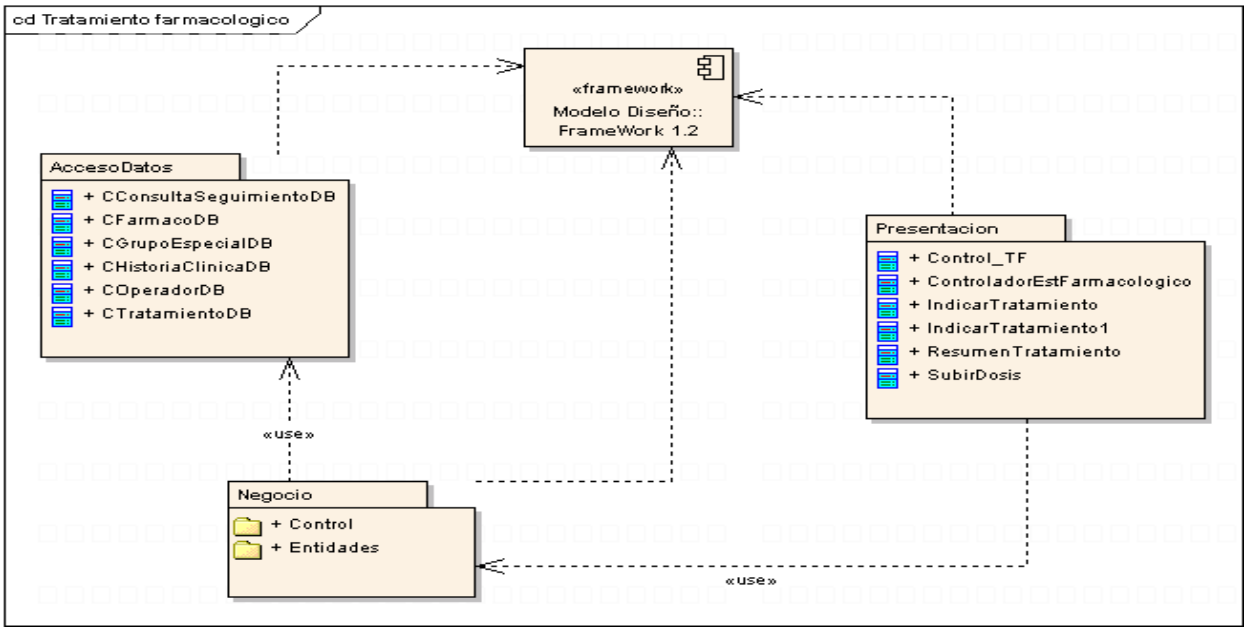

Figura 16 Subsistema de Tratamiento Farmacológico

La Figura 17 Representa la conformación del subsistema Configuración. En el que se representan las capas de Presentación, Negocio, Acceso a Datos y la relación con el FrameWork 1.2 que porta la tecnología Asp.Net 1.2

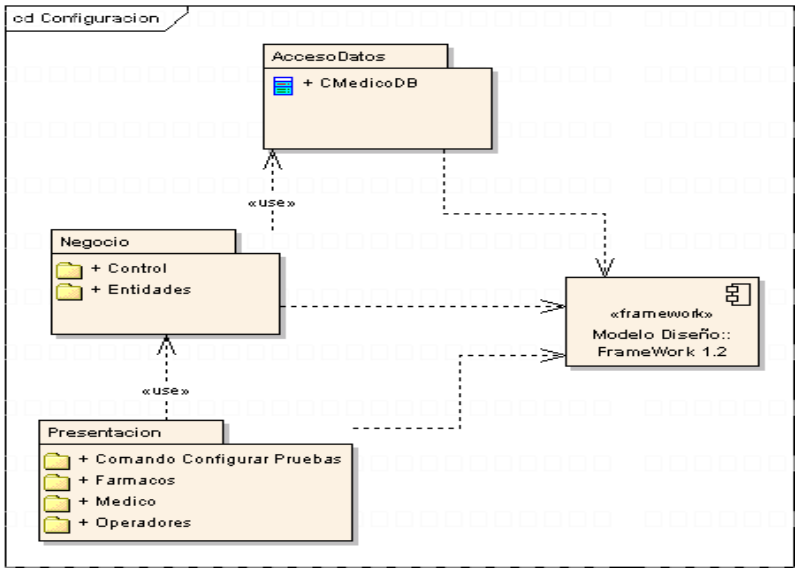

Figura 17 Subsistema de Configuración

Diagrama de clases de Diseño

En el (Anexo 8 Diagramas de clases asociados a los casos de uso de Indicar Tratamiento Farmacológico y Controlar Tratamiento Farmacológico) se muestran los diagramas de clases asociados los casos de uso Indicar Tratamiento Farmacológico y Controlar Tratamiento Farmacológico.

En el (Anexo 9 Diagramas de clases asociados los casos de uso Configurar Fármaco, Configurar Médico, Configurar Operador y Configurar Reglas) se muestran los diagramas de clases asociados los casos de uso Configurar Fármaco, Configurar Médico, Configurar Operador y Configurar Reglas.

En el (Anexo 10 Clases con sus operaciones y atributos que forman parte la solución) se muestran las operaciones y atributos de las clases que forman parte la solución.

Modelo de dato

El modelo de dato ha sido diseñado como un subsistema integro, ver Figura 18, debido al grado de interdependencia existente entre los datos persistentes del subsistema Tratamiento Farmacológico, el subsistema Configuración y el subsistema Comunes. La interdependencia existente no esta dada por la violación de los principios de diseño Bajo Acoplamiento y Alta Cohesión, sino por la dependencia semántica del funcionamiento de estos subsistemas, es decir el acoplamiento existente entre estos subsistemas es de tipo común, pues esta dado por compartir los mismos objetos del negocio, situación impuesta por la propia semántica del negocio. Es decir el módulo de configuración tiene la responsabilidad en el negocio de configurar los valores de las entidades persistentes que utiliza el subsistema Tratamiento Farmacológico para la gestión de la lógica del negocio.

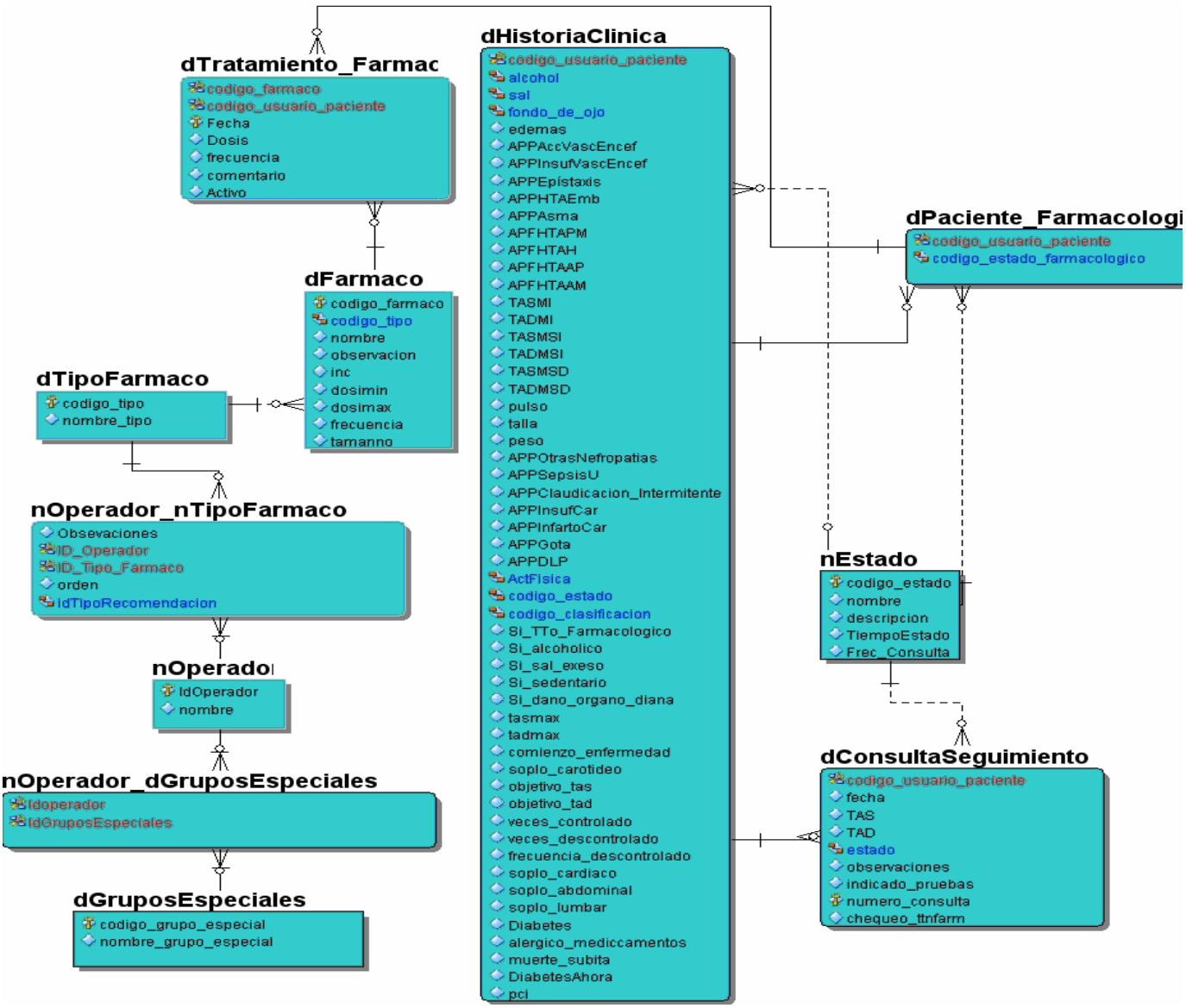

Figura 18 Modelo de datos

En este modelo de datos ha sido incluida además una tabla que muestra la relación existente entre entidades persistentes, clases de acceso a dato, tablas relacionales que modelan la fuente de persistencia de las entidades persistentes asociada y los procedimientos almacenados que agrupan lógicas de negocio. De modo que permita realizar una traza coherente desde la capa de negocio hasta la persistencia, para impedir obviar alguna información de persistencia del negocio en el esquema de

persistencia relacional e identificar aquellos aspectos el negocio (requerimientos funcionales) que fueron delegados al gestor de datos SQL Server 2000 mediante procedimientos almacenados.

Relación Entidades persistentes – tablas relacionales.

El esquema de relación existente entre las entidades modeladas en el diseño y el modelo de los datos relacional (tablas) concebido en la actividad de diseño de sistema de los subsistemas Tratamiento Farmacológico y Configuración se muestra en la Tabla 1 Relación Entidades persistentes – tablas relacionales.

Tabla 1 Relación Entidades persistentes – tablas relacionales

Entidades persistentes	Clase Acceso Datos	Tablas	Procedimientos Almacenados
Consulta	• CConsultaSeguimientoDB	• dConsultaSeguimiento • nEstado	• HyperWeb_Ultima_Consulta_Seguimiento • HyperWeb_insert_dConsultaSeguimiento • HyperWeb_Select_TodasConsultas • HyperWeb_SelectConsulta • HyperWeb_SelectTomasPresion
CFarmaco	• CFarmacoDB	• dFarmaco	• HyperWeb_Select_dFarmaco_Tipo • HyperWeb_insert_dFarmaco • HyperWeb_delete_dFarmaco • HyperWeb_update_dFarmaco • HyperWeb_Select_dFarmaco • HyperWeb_Select_nReacciones_Adversas_Farmaco
• CFarmaco_Paciente • Tratamiento_Paciente	• CTratamientoDB	• dPaciente_Farmacologico • dTratamiento_Farmacologico	• HyperWeb_ObtenerTodosTiposFarmacos • HyperWeb_GruposFarmacosPaciente • HyperWeb_Select_REaccionesAdv • HyperWeb_update_dTratamiento_Farmacologico_Activo • HyperWeb_Select_Farmacos_Efectos_Secundarios • HyperWeb_insert_dPaciente_FarmacologicodEfectosSecundarios • HyperWeb_Select_dTratamiento_Farmacologico_Paciente • HyperWeb_update_dTratamiento_Farmacologico_Dosis_Frecuencia
CGrupoEspecial	• CGrupoEspecialDB	• dGruposEspeciales	• HyperWeb_Select_GruposEspeciales_by_Paciente • HyperWeb_Select_dGrupoEspecial • HyperWeb_insert_dPaciente_dGruposEspecial

			es • HyperWeb_Update_dPaciente_dGruposEspeciales
CHistoriaClinica	• CHistoriaClinicaDB	• dHistoriaClinica	• HyperWeb_EstadoPaciente • HyperWeb_SelectT_dHistoriaClinica_PresionObjetivo • HyperWeb_update_dHistoriaClinica_DesControlado • HyperWeb_update_dHistoriaClinica_Controlado • HyperWeb_Select_dHistoriaClinica_nEstado_FrecConsult • HyperWeb_Update_dHistoriaClinica_codigo_estado • HyperWeb_Select_dHistoriaClinica_nEstado_dConsultaSeguimiento_Fecha • HyperWeb_dHistoriaClinica_nEstado_TiempoEstado • HyperWeb_dHistoriaClinica_nEstado_dConsulSeg_TAS_TAD • HyperWeb_Select_HistoriaClinica_GetHistoriaClinica • HyperWeb_Select_MedicamentosUso_GetMedicamentosPacienteHC • HyperWeb_Select_Estado_getNombreEstado • Hyperweb_select_usoMedicamentos_GetMedicamentosSegunID • HyperWeb_update_dHistoriaClinica_Actualizacion • HyperWeb_delete_Medicamentoa_en_Uso • HyperWeb_insert_dMedicamentoa_en_Uso
COperador	• COperadorDB	• nOperador • nOperador_dGruposEspeciales	• HyperWeb_select_GruposEspeciales_del_OP • HyperWeb_select_TipoFarmacosRecomendados_OP • HyperWeb_select_nOperador • HyperWeb_delete_dGruposEspeciales • HyperWeb_insert_dGruposEspeciales • HyperWeb_Existe_Operador • HyperWeb_insert_nOperador • HyperWeb_insert_nOperador_dGruposEspeciales • HyperWeb_insert_nOperador_nTipoFarmaco

			• HyperWeb_Select_nOperadores • HyperWeb_delete_nOperador
CMedico	• CMedicoDB	• dMedico	• HyperWeb_Add_Medico • HyperWeb_Update_Medico • HyperWeb_Select_Medico_ID
CGrupoFarmaco	• CFarmacoDB	• dTipoFarmaco • nOperador_nTipoFarmaco	• HyperWeb_Select_dTipoFarmaco • HyperWeb_insert_dTipoFarmaco • HyperWeb_update_dTipoFarmaco • HyperWeb_delete_dTipoFarmaco

Otro aspecto importante a tener en cuenta en el modelo de persistencia diseñado para los subsistemas Tratamiento Farmacológico y Configuración es el diseño del esquema del XML de persistencia de las reglas del negocio utilizadas en el diagnóstico médico de la Hipertensión Arterial que son configuradas a través del subsistema Configuración; esta funcionalidad esta relacionada con el Caso de Uso Configurar Reglas. La decisión de no persistir las reglas de negocio en un esquema relacional y dejarle la responsabilidad al Gestor de Base de Datos de la gestión de las mismas, viene dada, por las facilidades y la simplificación de persistencia que nos facilita la tecnología .net en su lenguaje c# 1.2 y 2.0 con la serialización de objetos a través de ficheros XML. Teniendo en cuenta que estas reglas no cambian casi en el tiempo, y que su acceso es bastante continuo, es más eficiente dejarlo en un fichero de reglas, que será leído una única vez cuando se habré el sistema y de ser modificado lo que se guarda, es el nuevo estado del objeto reglas, con toda la información de su composición incluida. La Figura 19 Esquema del XML de los ficheros de reglas Pruebas de Rutina y Pruebas adicionales. muestra el esquema de los ficheros XML que guardan la información de las Pruebas de Rutina y las Pruebas Adicionales.

Diagrama de despliegue

Aunque no es responsabilidad según la metodología utilizada para el desarrollo de este trabajo (RUP), incluir entre los artefactos a desarrollar en el flujo de trabajo de Análisis y diseño el modelo de despliegue de la solución informática, y teniendo plena conciencia de que el modelo de despliegue es un artefacto que debe ser creado durante el flujo de trabajo implementación, se decide incluirlo en el mismo, como instrumento de apoyo que permita visualizar de forma más clara y concreta la macro arquitectura de los subsistemas tratamiento farmacológico y configuración desarrollados en este trabajo de diploma.

```
<?xml version="1.0" encoding="utf-8" ?>
uebas xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<Pruebascompuestas>
   <anyType xsi:type=" ">
      <Necesidades />
      <Pruebashijas>
         <anyType xsi:type=" ">
            <Idprueba>" "</Idprueba>
            <Nombre>" "</Nombre>
            <Obligatoria>" "</Obligatoria>
            <UM>" "</UM>
            <Numerico>" "</Numerico>
            <Rangomin>" "</Rangomin>
            <Rangomax>" "</Rangomax>
            <RangopatminM>" "</RangopatminM>
            <RangopatmaxM>" "</RangopatmaxM>
            <RangopatminF>" "</RangopatminF>
            <RangopatmaxF>" "</RangopatmaxF>
            <Valornumerico>" "</Valornumerico>
         </anyType>
      </Pruebashijas>
      <Condiciones>
         <anyType xsi:type=" ">
            <Subcondiciones />
               <Nombre>" "</Nombre>
               <Comentario>" "</Comentario>
               <Nombrepruebaadicional>" "</Nombrepruebaadicional>
               <Idpruebaadicional>" "</Idpruebaadicional>
         </anyType>
      </Condiciones>
      <Nombre>" "</Nombre>
      <Id>" "</Id>
      <ClasPrueba>" "</ClasPrueba>
   </anyType>
</Pruebascompuestas>
<MaxId>" "</MaxId>
'CPruebas>
```

Figura 19 Esquema del XML de los ficheros de reglas Pruebas de Rutina y Pruebas adicionales.

El modelo de despliegue representado en la Figura 20 **y** Figura 21 muestran la distribución arquitectónica de los subsistemas Tratamiento farmacológico y Configuración, así como las características de la arquitectura cliente servidor que se propone para los mismos. Es importante destacar que la aplicación Web se comunica a través de las clases de acceso a datos implementadas en el FrameWork 1.2 que soportan ADO.NET con el servidor de base de datos y la comunicación entre los exploradores Web clientes y el servidor Web es mediante el protocolo HTTP. El servidor Web esta soportado por el ISS (Internet information Server) y el servidor de datos es SQL Server 2000.

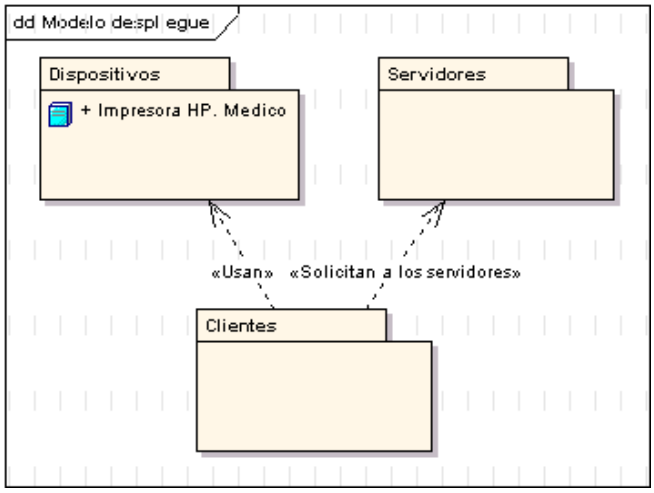

Figura 20 Unidades del Modelo de Despliegue

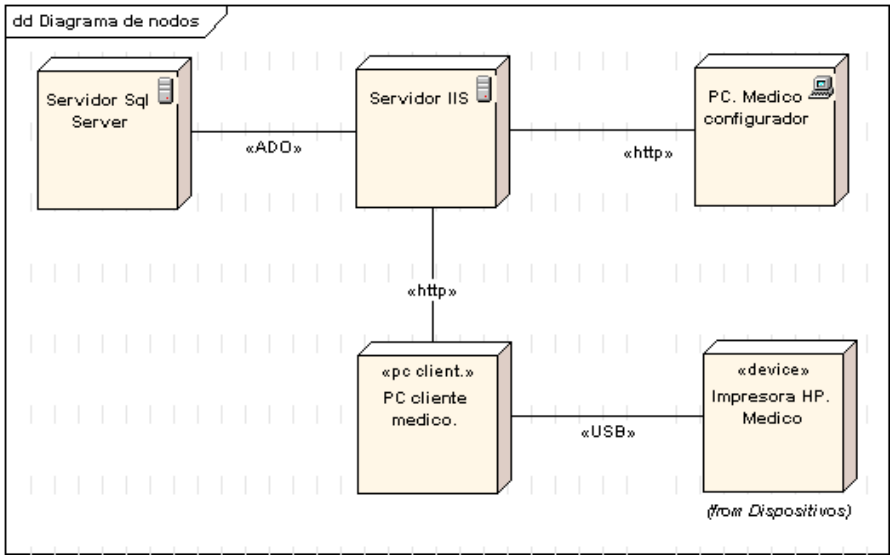

Figura 21 Diagrama del Modelo de Despliegue

Evaluación del modelo de diseño propuesto

Son varios los puntos de vista relacionados con la calidad del software. Desde metodologías hasta las distintas normas de calidad, que pueden estar orientados tanto a los procesos de desarrollo como a los productos de software. No es objetivo de este trabajo abundar sobre los temas de calidad, pero si desarrollar una evaluación del diseño obtenido en la solución propuesta de diseño de software a los módulos Tratamiento Farmacológico y Configuración del proyecto de software HyperWeb.

Para ello se realizó en una primera fase un mapeo entre las características funcionales asociados a los casos de usos de los módulos de Tratamiento Farmacológico y Configuración y el diseño de clases construidos para dar solución a estos requisitos funcionales. De este modo se garantizó el chequeo de cumplimiento del atributo interno de calidad más importante que plantea la norma ISO 9126 (ISO/IEC, 2001), la Funcionalidad, que consiste en la capacidad del software de proveer las funciones que cumplen con las necesidades implícitas y explícitas cuando el mismo es utilizado bajo ciertas condiciones (ARREGUI Septiembre 2005). Demostrándose que las características funcionales más importantes definidas para ambos módulos han sido cubierta en el diseño propuesto.

Subsistema Tratamiento Farmacológico.

Tabla 2 Requisitos asociados a Tratamiento Farmacológico

Caso Uso	Requerimiento funcional asociado al caso de uso.	Clase el diseño que lo cubre	Método de la clase que lo cubre
Indicar Tratamiento Farmacológico	• Proponer grupos de fármacos a indicar al paciente.	• CGestorTratamiento	• TratamientoPersonalizado(idpaciente: int, pedido: int, (inout) contraindicados: ArrayList):ArrayList
	• Obtener los fármacos que conforman un grupo de fármacos.	• CGestorTratamiento	• ObtenerFarmacosGrupo(idgrupo : int):SqlDataReader
	• Salvar tratamiento farmacológico	• CGestorTratamiento	• SalvarTratamientoPaciente(idpaciente: int, farmacos_paciente: ArrayList):bool

Controlar tratamiento Farmacológico.	• Obtener los efectos secundarios producidos por uno o varios fármacos.	• CGestorTratamiento	• Efectos_secundarios_farmacos(listaf: ArrayList): ArrayList
	• Salvar los fármacos que produjeron efectos secundarios en el paciente	• CGestorTratamiento	• Insertar_TF_con_ES(idpaciente: int, listafarmacos: ArrayList): bool
	• Elaborar sugerencia de opciones a seguir con el paciente en el Tratamiento Farmacológico	• CGestorTratamiento	• SugerenciaTF(idpaciente: int, (inout) opciones: ArrayList): string
	• Mostrar historial de los tratamientos farmacológicos que ha tenido el	• CGestorTratamiento	• Historial_TF(idpaciente: int): ArrayList

	paciente.		
	• Subir dosis de un fármaco indicado.	• CGestorTratamiento	• Actualizar_Farmaco_Paciente(id paciente: int, farmaco: CFarmaco_Paciente): bool

Subsistema Configuración.

Tabla 3 Requisitos asociados a Configuración

Caso Uso	**Requerimiento funcional asociado al caso de uso.**	**Clase el diseño que lo cubre**	**Método de la clase que lo cubre**
Configurar médico	• Insertar medico	• CGestorUsuarios	• InsertarMedico(string username, string passwd, string nombre, string apellido1, string apellido2, string registro, int graduacion):int
	• Actualizar médico	• CGestorUsuarios	• ActualizarMedico(CMedico med):int
	• Eliminar médico	• CGestorUsuarios	• EliminarMedico(int id):void
Configurar Fármaco	• Insertar un fármaco	• CGestorTratamiento	• AdicionarFarmaco(int idgrupo, CFarmaco farmaco):void

	• Actualizar un fármaco	• CGestorTratamiento	ActualizarFarmaco(CFarmaco farmaco):void
	• Eliminar un fármaco	• CGestorTratamiento	EliminarFarmaco(int idfarmaco):void
	• Cada fármaco debe estar asociado a un grupo de fármaco.	• CGestorTratamiento	ActualizarGrupoFarmaco(int idgrupo, string nombre):void
		CGrupoFarmaco	
Configurar Operador	• Insertar operador.	• CGestorOperadores	InsertarOperador(string nombre, ArrayList idgrupoespecial, ArrayList recomedado, ArrayList contraindicado):bool
	• Eliminar operador.	• CGestorOperadores	EliminarOperador(int idoperador):bool
	• Un operador cubre grupos especiales.	• CGestorOperadores • CGrupoEspecial •	COperador.gruposespeciales: ArrayList
	• Insertar grupo especial	• CGestorOperadores	AdicionarGrupoEspecial(string nombre):bool
	• Eliminar grupo especial	• CGestorOperadores	EliminarGrupoEspecial(int idgrupo):bool

	• UN operador define fármacos indicados y contraindicados.	• CGestorOperadores	COperador.contraindicado:ArrayList COperador.recomendado:ArrayList
Configurar Reglas	• Definir una regla	• CGestorAccesoXml	SalvarPruebas(CPruebas pruebas, TipoPrueba tipo):bool
	• Actualizar una reglas	• CGestorAccesoXml	SalvarPruebasAdicionales(CPruebas pruebas):bool
	• Eliminar una regla	• CGestorAccesoXml	SalvarPruebasRutina(CPruebas pruebas):bool

Otro aspecto importante a tener en cuenta en la fase dos de evaluación de la calidad del diseño ha sido la creación de métricas básicas inspiradas en el estudio de la calidad del diseño orientado a objeto referenciadas por Pressman en (PRESSMAN 1998); teniendo en cuenta que este estudio brinda un esquema sencillo de implementar y que a la vez cubre los principales atributos de calidad de software. Siendo esto la principal razón de la concepción de las métricas inspiradas en lo propuesto por Pressman.

Atributos de calidad que se abarcan:

1. Responsabilidad. Consiste en la responsabilidad asignada a una clase en un marco de modelado de un dominio o concepto, de la problemática propuesta.
2. Complejidad del diseño. Consiste en la complejidad que posee una estructura de diseño de clases.
3. Complejidad de implementación. Consiste en el grado de dificultad que tiene implementar un diseño de clases determinado.
4. Reutilización. Consiste en el grado de reutilización de presente en una clase o estructura de clase, dentro de un diseño de software.
5. Acoplamiento. Consiste en el grado de dependencia o interconexión de una clase o estructura de clase, con otras, esta muy ligada a la característica de Reutilización.

6. Complejidad del mantenimiento. Consiste en el grado de esfuerzo necesario a realizar para desarrollar un arreglo, una mejora o una rectificación de algún error de un diseño de software. Puede influir indirecta, pero fuertemente en los costes y la planificación del proyecto.
7. Cantidad de pruebas. Consiste en el número o el grado de esfuerzo para realizar las pruebas de calidad (Unidad) del producto (Componente, modulo, clase, conjunto de clases, etc.) diseñado.
8. Nivel de Cohesión. Consiste en el grado de especialización de las clases concebidas para modelar un dominio o concepto específico.
9. Abstracción del diseño. Consiste en la capacidad de modelar lo más cercano posible a la realidad un concepto o dominio determinado.

Las métricas concebidas como instrumento para evaluar la calidad del diseño de los subsistemas de Tratamiento Farmacológico, Configuración y Comunes y su relación con los atributos de calidad definidos en este trabajo son las siguientes:

Tamaño operacional de clase (TOC): Está dado por el número de métodos asignados una clase.

Tabla 4 Tamaño operacional de clase (TOC)

Atributo que afecta	Modo en que lo afecta
Responsabilidad	Un aumento del TOC implica un aumento de la responsabilidad asignada a la clase.
Complejidad de implementación	Un aumento del TOC implica un aumento de la complejidad de implementación de la clase.
Reutilización	Un aumento del TOC implica una disminución en el grado de reutilización de la clase.

Relaciones entre clases (RC): Está dado por el número de relaciones de uso de una clase con otras.

Tabla 5 Relaciones entre clases (RC)

Atributo que afecta	Modo en que lo afecta
Acoplamiento	Un aumento del RC implica un aumento del Acoplamiento de la clase.
Complejidad del mantenimiento	Un aumento del RC implica un aumento de la complejidad del mantenimiento de la clase.

Reutilización	Un aumento del RC implica una disminución en el grado de reutilización de la clase.
Cantidad de pruebas	Un aumento del RC implica un aumento de la Cantidad de pruebas de unidad necesarias para probar una clase.

Profundidad de Herencia (PH): Está dado por la profundidad en al herencia de las clases heredades de un nodo padre.

Tabla 6 Profundidad de Herencia (PH)

Atributo que afecta	**Modo en que lo afecta**
Complejidad del mantenimiento	Un aumento del PH implica una disminución de la complejidad del mantenimiento de la clase.
Complejidad del diseño	Un aumento del PH implica una disminución de la complejidad del diseño de la clase.
Reutilización	Un aumento del PH implica un aumento en el grado de reutilización de la clase.

Número de Descendientes (ND): Está dado por la cantidad de clases que heredan de un padre.

Tabla 7 Número de Descendientes (ND)

Atributo que afecta	**Modo en que lo afecta**
Abstracción del diseño	Un aumento del ND puede diluir la abstracción del diseño de clases.
Cantidad de pruebas	Un aumento del ND implica un aumento de la Cantidad de pruebas de unidad necesarias para probar una clase.
Reutilización	Un aumento del ND implica un aumento en el grado de reutilización de la clase.
Nivel de Cohesión	Un aumento del ND implica una disminución en el nivel de cohesión de la clase.

Número de Operaciones Redefinidas para una clase hija (NOR): Está dado por la cantidad de operaciones redefinidas en cada clase hija.

Tabla 8 Número de Operaciones Redefinidas para una clase hija (NOR)

Atributo que afecta	Modo en que lo afecta
Abstracción del diseño	Un aumento del NOR implica que se ha logrado una buena definición de la abstracción del diseño de clases.
Cantidad de pruebas	Un aumento del NOR implica un aumento de la Cantidad de pruebas de unidad necesarias para probar una clase.
Complejidad del mantenimiento	Un aumento del NOR implica un aumento de la complejidad del mantenimiento de la clase.

A continuación se presentan los resultados obtenidos de la aplicación de los instrumentos de evaluación para medir las métricas anteriormente descritas, y una valoración de los mismos.

Resultados del instrumento de evaluación de la métrica Tamaño operacional de clase (TOC)

Ver instrumentos y tabla de resultados en (Anexo 11 Instrumento de medición de la métrica Tamaño operacional de clase (TOC)).

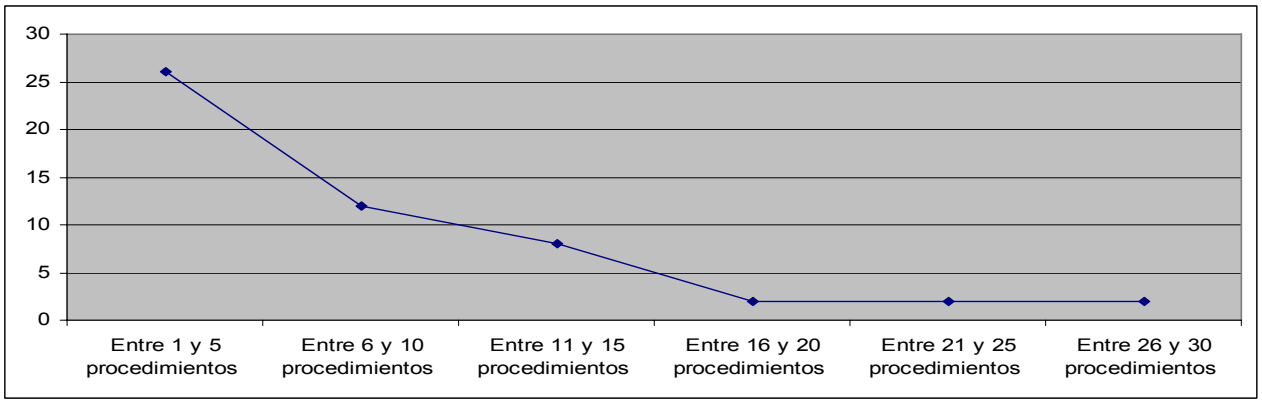

Figura 22 Representación de los resultados obtenidos en el instrumento agrupados en los intervalos definidos

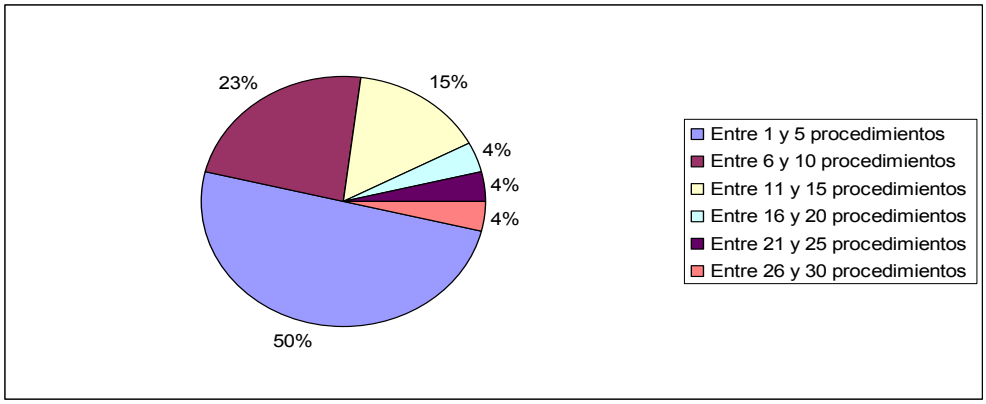

Figura 23 Representación en % de los resultados obtenidos en el instrumento agrupados en los intervalos definidos

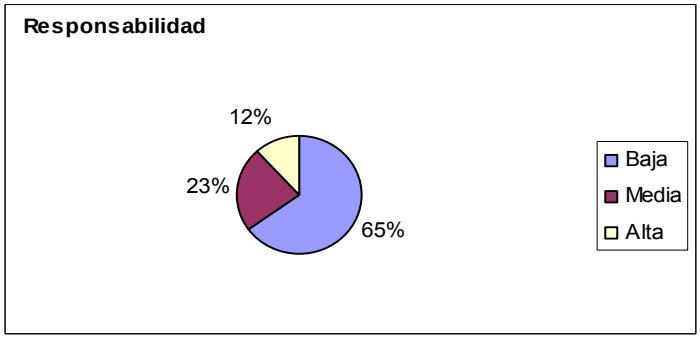

Figura 24 Representación de la incidencia de los resultados de la evaluación de la métrica TOC en el atributo Responsabilidad

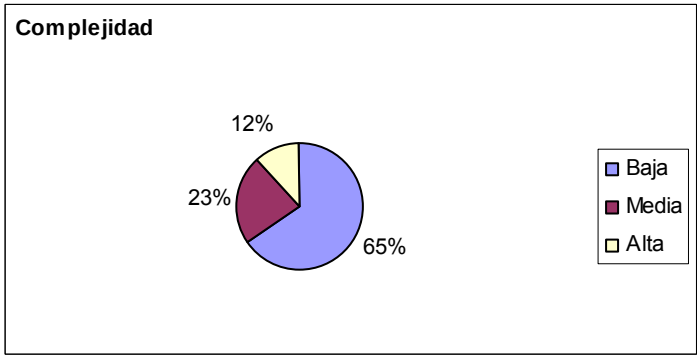

Figura 25 Representación de la incidencia de los resultados de la evaluación de la métrica TOC en el atributo Complejidad de Implementación

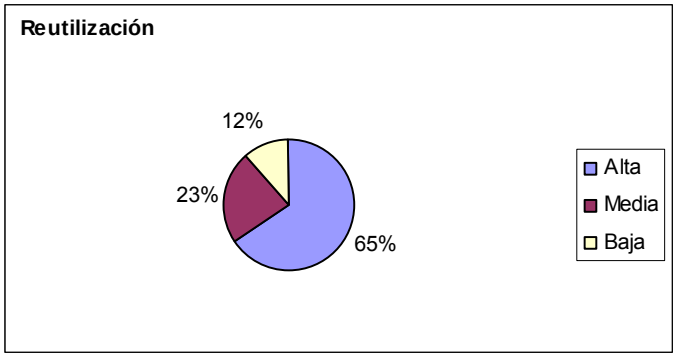

Figura 26 Representación de la incidencia de los resultados de la evaluación de la métrica TOC en el atributo Reutilización

Haciendo un análisis de los resultados obtenidos en la evaluación del instrumento de medición de la métrica TOC, se puede concluir que el diseño de los subsistemas Tratamiento Farmacológico, Configuración y Comunes tienen una calidad aceptable teniendo en cuenta que el 88 % de las clases incluidas en estos subsistemas posee menos cantidad de operaciones que la mitad del valor máximo registrado en las mediciones. Además el 88% de las clases poseen evaluaciones positivas en los atributos de calidad (Responsabilidad, Complejidad de Implementación y Reutilización).

Resultados del instrumento de evaluación de la métrica Relaciones entre Clases (RC)

Ver instrumentos y tabla de resultados en (Anexo 12 Instrumento de medición de la métrica Relaciones entre clases (RC)).

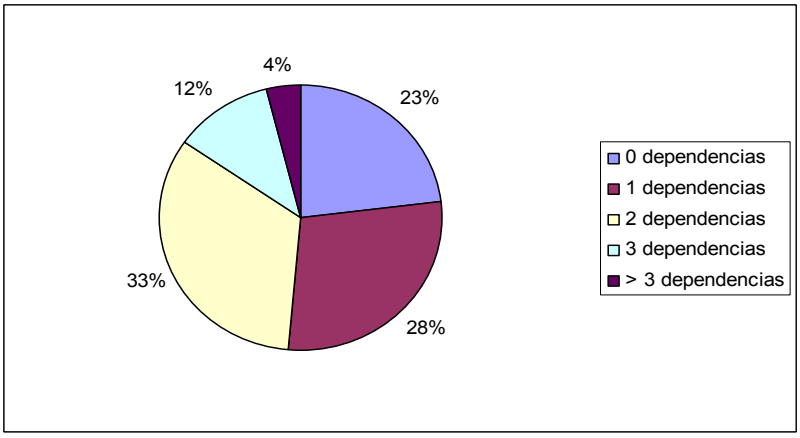

Figura 27 Representación en % de los resultados obtenidos en el instrumento agrupados en los intervalos definidos

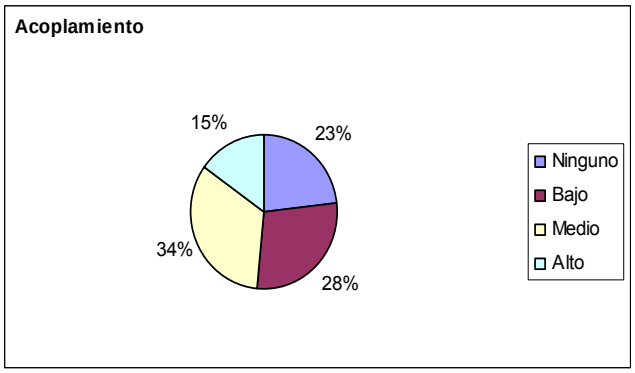

Figura 28 Representación de la incidencia de los resultados de la evaluación de la métrica RC en el atributo Acoplamiento

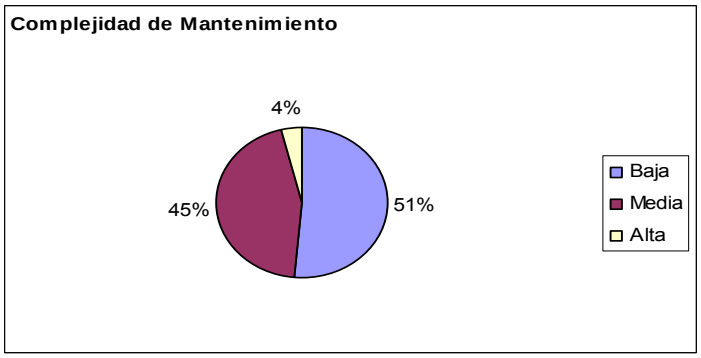

Figura 29 Representación de la incidencia de los resultados de la evaluación de la métrica RC en el atributo Complejidad de Mantenimiento

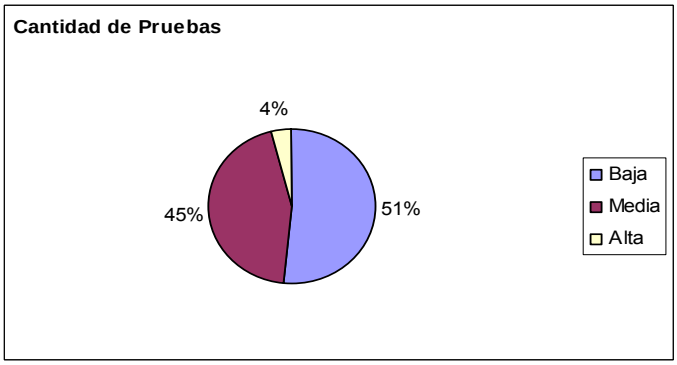

Figura 30 Representación de la incidencia de los resultados de la evaluación de la métrica RC en el atributo Cantidad de Pruebas

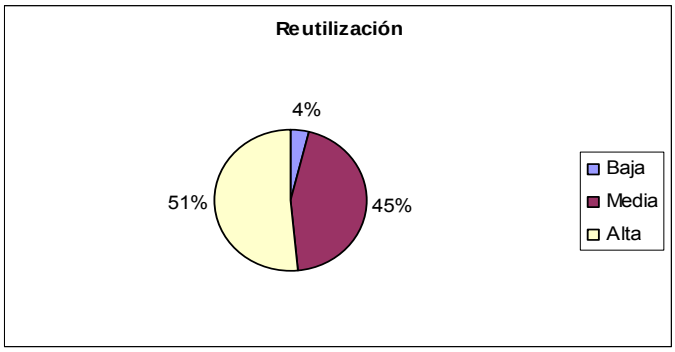

Figura 31 Representación de la incidencia de los resultados de la evaluación de la métrica RC en el atributo Reutilización

Haciendo un análisis de los resultados obtenidos en la evaluación del instrumento de medición de la métrica RC, se puede concluir que el diseño de los subsistemas Tratamiento Farmacológico, Configuración y Comunes tienen una calidad aceptable teniendo en cuenta que el 84 % de las clases incluidas en estos subsistemas posee menos de 3 dependencias de otras clases. Además el 23% de las clases no poseen acoplamiento con otras y el 85% posee índices aceptables en cuanto a Acoplamiento. Así mismo los atributos de calidad Complejidad de Mantenimiento, Cantidad de Pruebas y Reutilización se comportan satisfactoriamente en un 96 % de las clases.

Resultados del instrumento de evaluación de la métrica Profundidad de Herencia (PH)

Ver instrumentos y tabla de resultados en (Anexo 13 Instrumento de medición de la métrica Profundidad de Herencia (PH)).

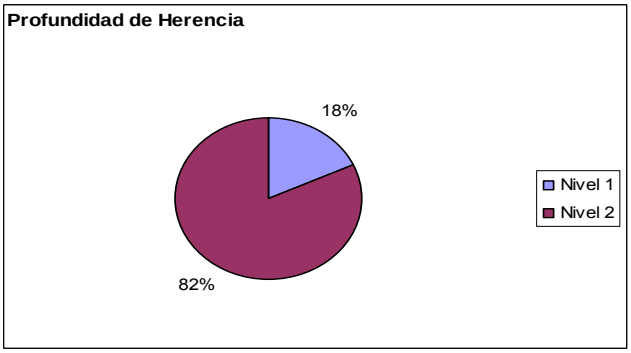

Figura 32 Representación en % de los resultados obtenidos en el instrumento agrupados por nivel

Haciendo un análisis de los resultados obtenidos en la evaluación del instrumento de medición de la métrica PH, se puede concluir que el diseño de los subsistemas Tratamiento Farmacológico, Configuración y Comunes tienen una calidad aceptable teniendo en cuenta que la profundidad de herencia presentes en los subsistemas desarrollados es siempre de 2, el cual es el valor mínimo posible. Analizando los atributos de calidad Complejidad de Mantenimiento, Complejidad de Diseño se puede decir que sin duda se tienen buenos índices debido a que solo se cuenta como máximo con un nivel 2 de profundidad en la herencia. Sin embargo el atributo Reutilización no posee indicadores positivos debido al bajo nivel de profundidad de herencia.

Resultados del instrumento de evaluación de la métrica Número de Descendientes (ND)

Ver instrumentos y tabla de resultados en (Anexo 14 Instrumento de medición de la métrica Número de Descendientes (ND)).

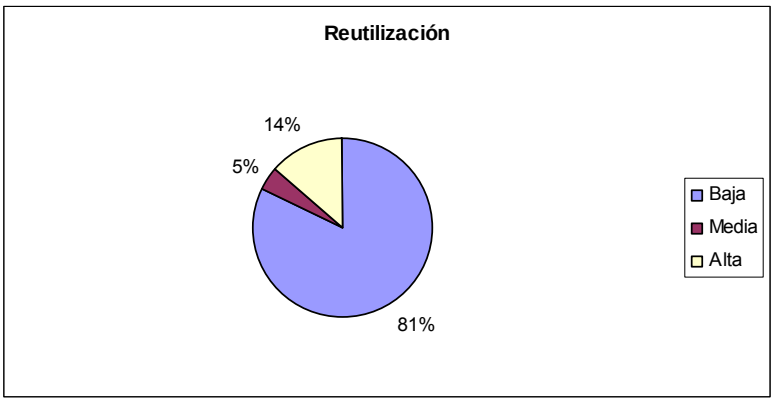

Figura 33 Representación de la incidencia de los resultados de la evaluación de la métrica ND en el atributo Reutilización

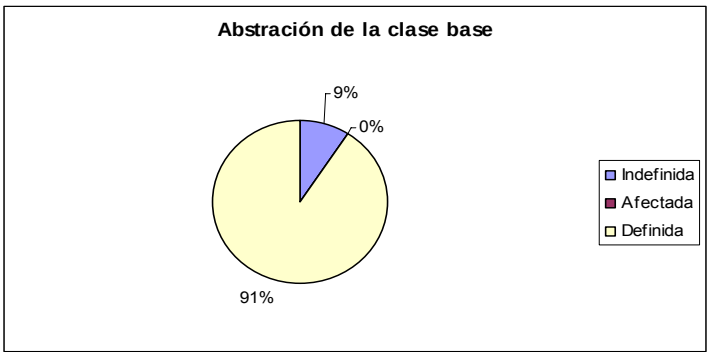

Figura 34 Representación de la incidencia de los resultados de la evaluación de la métrica ND en el atributo Abstracción de la clase base

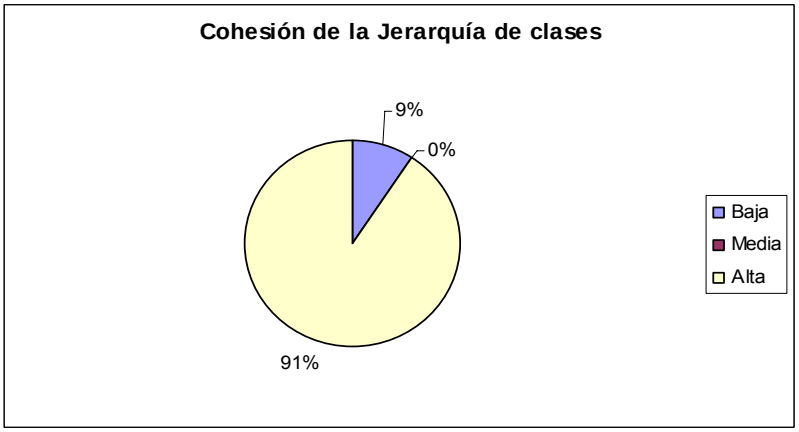

Figura 35 Representación de la incidencia de los resultados de la evaluación de la métrica ND en el atributo Cohesión de la Jerarquía de clases

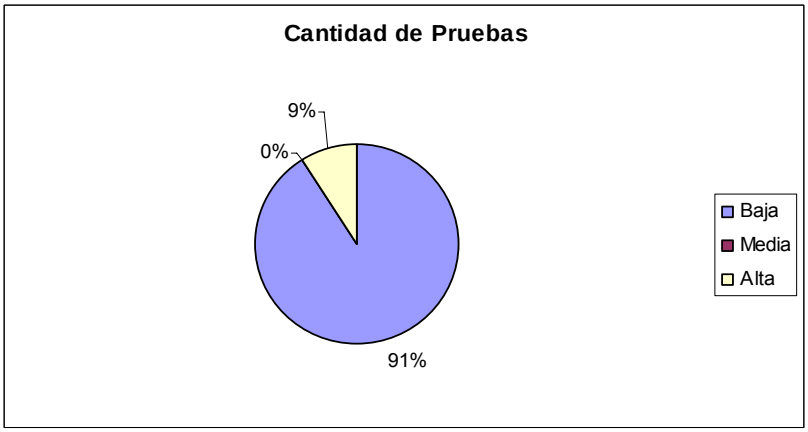

Figura 36 Representación de la incidencia de los resultados de la evaluación de la métrica ND en el atributo Cantidad de Pruebas

Haciendo un análisis de los resultados obtenidos en la evaluación del instrumento de medición de la métrica ND, se puede concluir que el diseño de los subsistemas Tratamiento Farmacológico, Configuración y Comunes tienen una calidad aceptable teniendo en cuenta que solo se emplea la herencia en casos bien identificados a partir de las necesidades del negocio o del diseño. Solo 4 clases poseen descendientes y solo en dos de estos casos la cantidad de descendientes supera la cantidad de 5.

Valorando el atributo de calidad Reutilización se puede decir que por lo antes explicado los índices de Reutilización se mantienen bajos representados por un 81%. En cuanto al atributo Abstracción de la clase base se muestra como existe una tendencia a la conservación de la abstracción. Tanto para la Cohesión de la jerarquía de clases como para el atributo Cantidad de Pruebas los índices son positivos favoreciendo esto al diseño.

Resultados del instrumento de evaluación de la métrica Número de Operaciones Redefinidas (NOR)

Ver instrumentos y tabla de resultados en (Anexo 15 Instrumento de medición de la métrica Número de Operaciones Redefinidas(NOR)).

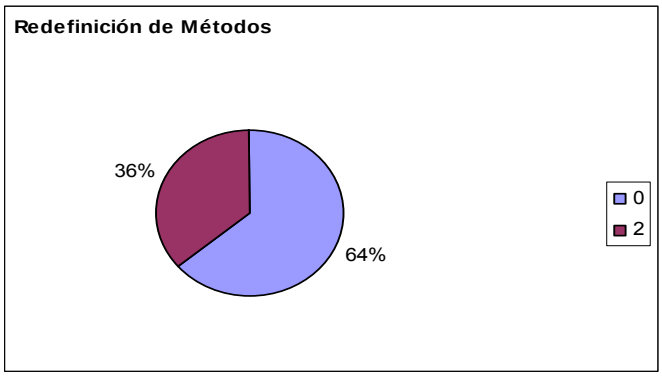

Figura 37 Representación en % de los resultados obtenidos en el instrumento agrupados en los valores existentes

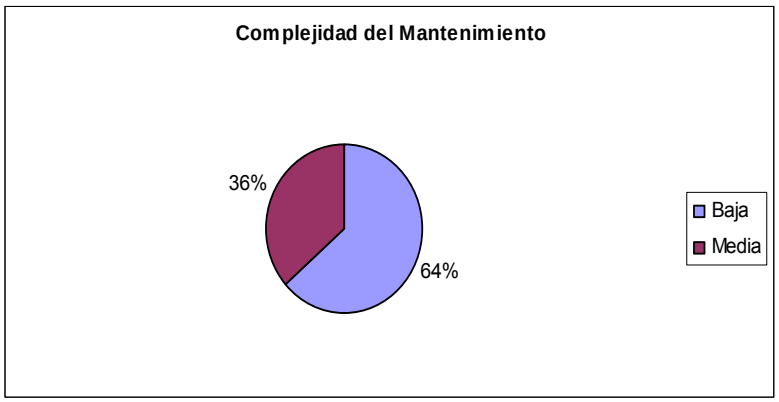

Figura 38 Representación de la incidencia de los resultados de la evaluación de la métrica NOR en el atributo Complejidad del Mantenimiento

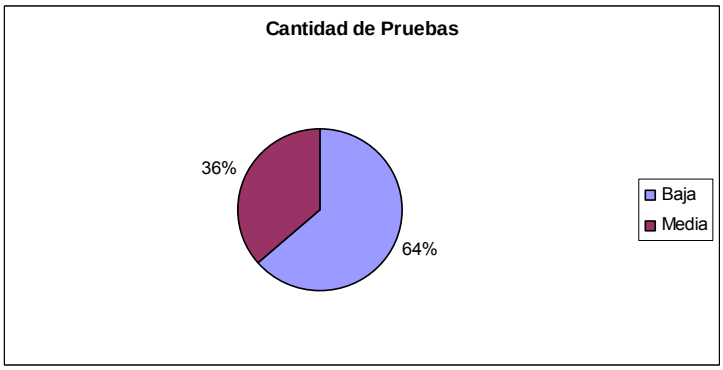

Figura 39 Representación de la incidencia de los resultados de la evaluación de la métrica NOR en el atributo Cantidad de Pruebas

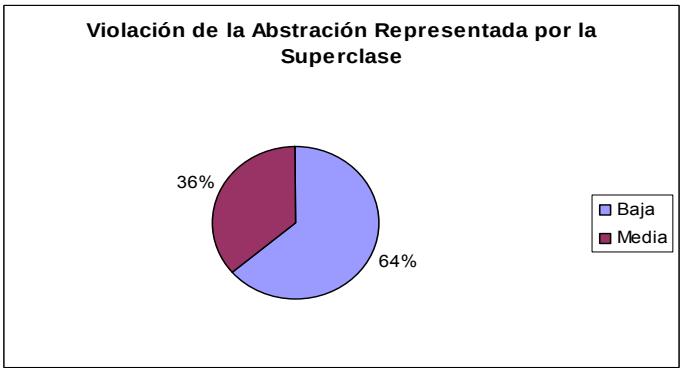

Figura 40 Representación de la incidencia de los resultados de la evaluación de la métrica NOR en el atributo Violación de la Abstracción representada por la superclase

Haciendo un análisis de los resultados obtenidos en la evaluación del instrumento de medición de la métrica NOR, se puede concluir que el diseño de los subsistemas Tratamiento Farmacológico, Configuración y Comunes tienen una calidad aceptable teniendo en cuenta que solo el 36 % de las clases incluidas en el análisis (solo relacionadas con la herencia) redefinen algún método heredado. Además los indicadores se comportan de forma adecuada para los atributos de calidad Complejidad del Mantenimiento, Cantidad de Pruebas, Violación de la Astricción representada por la superclase.

A manera de resumen se han tabulado los resultados obtenidos en la siguiente tabla.

Tabla 9 Resumen de los resultados

	TOC %	RC %	ND %	NOR %	PH %
Responsabilidad	88 B				
Complejidad del diseño					100 B
Complejidad de implementación	88 B				
Reutilización	88 B	96 B	81 M (Padre)		100 M (Padre)
Acoplamiento		85 B			
Complejidad del mantenimiento		96 B		36 R y 64 B	100 B
Cantidad de pruebas		96 B	91 B	100 B	
Nivel de Cohesión			91 B		
Abstracción del diseño			91 B	100 B	

Conclusiones

A manera de conclusión se pudiese decir que este capítulo abarcó los aspectos más significativos de la solución propuesta, posibilitando que a partir de los artefactos generados se pueda desarrollar la fase de Implementación de la aplicación. Se escogen artefactos como diagramas de subsistemas, de clases, de secuencia que tributan a la realización de los casos de uso incluidos en la solución. También se recogen el modelo de datos y el diagrama de despliegue. La claridad que se logró en la modelación realizada lleva a tener facilidad posteriormente no solo de implementación sino de reutilización, de mantenimiento y lo más significativo, se obtiene mayor calidad total. Finalmente se elaboraron instrumentos inspirados en métricas para calidad del diseño que mediante su empleo permitió afirmar que el diseño realizado se puede valorar de aceptable.

CONCLUSIONES GENERALES

A manera de conclusión se pudiese plantear que los estudios realizados alrededor del proceso de indicación y control del tratamiento farmacológico de un paciente arrojaron que este se ha convertido en un problema complejo de solucionar y con un alcance mundial. Esta complejidad ha propiciado que informáticamente se aborde el tema, sin embargo, las herramientas analizadas solo abarcan hasta el proceso de evaluación clínica del paciente y procesos como la ayuda a la toma de decisiones en la indicación y control del tratamiento farmacológico no se tratan con la importancia necesaria.

A raíz de lo antes planteado y complementando lo recogido en el primer objetivo específico propuesto para esta tesis, se realiza un estudio de un conjunto de conceptos y aspectos dirigidos al diseño de software, resaltando la incidencia de un mal diseño y los factores de un correcto diseño de una aplicación informática, de forma tal que se ganase en una completa claridad de cuales eran los aspectos que debían estar presentes en la solución propuesta. Es decir, se identificó la necesidad de la utilización de patrones y estándares de diseño que propiciarían resultados positivos en materia de cohesión, acoplamiento, reutilización, calidad y facilidad de mantenimiento.

Buscando solidez en los algoritmos elaborados para el tratamiento farmacológico se estudian un conjunto de técnicas de Inteligencia Artificial que propician un apoyo a la ayuda a la toma de decisiones, aspecto importante de acuerdo a lo que se deseaba lograr en materia de indicación y control de tratamiento farmacológico.

Ya a la altura del segundo capítulo de la tesis se comienza realizando un estudio de las metodologías, herramientas y patrones que potencialmente se hubiesen podido aplicar en el desarrollo de la solución propuesta, se concluye con la explicación de la elección realizada en cada uno de los casos.

En función de darle cumplimiento al segundo y tercer objetivo específico trazado, se desarrolló la modelación del diseño de sistema de los módulos Tratamiento Farmacológico y Configuración de acuerdo a lo planteado por la metodología seleccionada, en este caso RUP. Esta metodología enmarca el desarrollo del diseño de sistema en la Fase de Análisis y Diseño, y para ello determina un conjunto de actividades que devienen en artefactos que se exponen y explican durante el tercer capítulo del siguiente trabajo. Se escogen artefactos como diagramas de subsistemas, de clases, de secuencia que tributan a la realización de los casos de uso incluidos en la solución. También se recogen el modelo de datos y el diagrama de despliegue. La claridad que se logró en la modelación realizada lleva a tener facilidad

posteriormente no solo de implementación sino de reutilización, de mantenimiento y lo más significativo, se obtiene mayor calidad total.

Para darle cumplimiento al cuarto objetivo específico y base a la afirmación anterior se realiza una validación del diseño mediante la utilización de instrumentos de medición que se inspiraron en métricas para la calidad del diseño. Los resultados arrojados permitieron concluir que el diseño presentaba valores positivos en indicadores de calidad tales como Reutilización, Facilidad de Mantenimiento, Complejidad del Diseño, Complejidad de Implementación, Cohesión, Acoplamiento, Cantidad de pruebas entre otros. Esto apoya la afirmación de que el diseño desarrollado se puede considerar como aceptable.

RECOMENDACIONES

Como recomendación de este trabajo se considera importante incluir los siguientes aspectos:

- Desarrollar el diseño de un subsistema que modele un mecanismo auditable, para el control de las decisiones tomadas por los médicos, durante el flujo de trabajo de tratamiento farmacológico, que no estén en correspondencia con las sugerencias del sistema. Esto permitiría garantizar un mecanismo de protección del sistema, ante las incorrectas decisiones tomadas por los médicos usuarios del mismo.
- Desarrollar el diseño de un mecanismo de replica de información, que permita la migración de los datos de un sistema a otro aunque estén en situaciones geográficas bien distantes. De esta forma se podría continuar el ciclo de consultas que el paciente ha estado realizando.
- Estudiar y desarrollar una estrategia para la migración de este sistema hacia software libre, de manera que se pueda reutilizar gran parte del modelo ya diseñado.
- Incluir en versiones posteriores de este trabajo, la utilización de estándares médicos como el HL7, que permitan la integración de este sistema a entonos más genéricos e internacionales.
- Desarrollar un estudio a partir de las experiencias obtenidas en este trabajo, que permitan desarrollar un sistema genérico, para la construcción de sistemas expertos aplicados a los problemas de diagnóstico médico.

Como recomendación general, se considera que se hace necesario el desarrollo y uso de sistemas que ayuden a mejorar la toma de decisiones de los médicos en las diversas especialidades. Lo cual pudiera contribuir a expandir la experticidad y estandarizar métodos de efectividad comprobada para el tratamiento de las diversas enfermedades, elevándose de esta manera los indicadores de salud de la población.

BIBLIOGRAFÍA

1. ADESFA. Definición de mensajería entre aplicación para la validación en línea de prescripciones médicas, http://www.foureware.com.ar/adesfa/A000001R01.html, 2007.
2. AHA, D. W. Case-Based Learning Algorithm, http://www.aic.nrl.navy.mil/~aha/papers/aha-cbr91.ps, 1991.
3. ALAMEDA, A. El viejo EDI, se alía con Internet, http://www.verticalia.com/index.php?option=com_content&task=view&id=91&Itemid=36, 2006.
4. ALBIN, S. The Art of Software Architecture: Design methods and techniques. Nueva York, Adison Wiley, 2003. p.
5. ALFREDO MORALES OLIVA, E. M. D. SISTEMA PARA LA AYUDA A LA TOMA DE DECISIONES EN EL DIAGNÓSTICO, CONTROL Y TRATAMIENTO DE LOS PACIENTES CON HIPERTENSIÓN ARTERIAL. Módulos de Evaluación Clínica y Tratamiento No Farmacológico., Universidad de Ciencias Informáticas, 2006. p.
6. ANSI. Health Level Seven, http://www.hl7.org/, 2006.
7. ARREGUI, J. J. O. Revisión Sistemática de Métricas de Diseño Orientado a Objetos. Facultad de Informática. Madrid. España., Universidad Politécnica de Madrid, Facultad de Informática, Septiembre 2005. p.
8. ÁVILA, S. J. V. Introducción a Microsoft Solutions Framework, http://www.mentores.net/articulos/intro_microsoft_sol_frame.htm, 2005.
9. BAKER, F. T. Structured programming in a production programming environment. ACM Press, 1975. 172-185 p.
10. BELLO, R. Métodos de Solución de Problemas para la Inteligencia Artificial, Universidad Central de Las Villas. CUBA, 1998. p.
11. BIOCOM. Estandarización en la transmición de la información Informática Médica HL7, http://biocom.com/informatica_medica/HL7_introduccion.html, 2006a.
12. ---. La Historia Clínica Informatizada y de los HIS, http://www.biocom.com.mx/informatica_medica/codificacion.html, 2006b.
13. ---. LOINC - Logical Observation Identifiers Names and Codes, http://biocom.com/sistema/documentacion/standares_codificaciones/loinc.html, 2006c.

14. BOOCH, G. Object-Oriented Design with Applications. The Benjamin/Cummings Publishing Company. 1991. p.
15. BROWN, W. J. AntiPatterns: refactoring software, architectures, and projects in crisis. John Wiley & Sons, 1998. p. 0-471-19713-0
16. BUSCHMANN, F. Pattern-Oriented Software Architecture. John Wiley & Sons, 1996. p.
17. CABALLERO, M. D. P. Guia cubana para la prevención, diagnóstico y tratamiento de la hipertensión arterial 2006. 2006. p.
18. CARRIÓN PINZÓN MARIO ISRAEL, M. A. H. G., M.C. JOSE EVARISTO PACHECO VELASCO* MonoUML. Herramienta CASE http://ewh.ieee.org/sb/mexico/itveracruz/techpapers/monouml.pdf, 2007.
19. CHOBANIAN, A. V.; G. L. BAKRIS, et al. SEVENTH REPORT OF THE JOINT NATIONAL COMMITTEE ON PREVENTION, DETECTION, EVALUATION, AND TREATMENT OF HIGH BLOOD PRESSURE. Boston, Chicago,Michigan, New York, Buffalo, Mississippi, Miami, Alabama, Cleveland, Ohio, National High Blood Pressure Education Program Coordinating Committee, 2003. 47.
20. CUETO, M. H. Incidencia y prevalencia de la hipertensión arterial por edad, sexo y provincias se incluyen otras afecciones relacionadas 2005, http://www.sld.cu/servicios/hta/, 2006.
21. DAEDALUS. Diseño de Sistemas, http://www.daedalus.es/AreasISDiseno-E.php, 2006.
22. DAVIS, A. Principles of Software Development. McGraw-Hill, 1995. p.
23. DENNIS, J. B. Modularity. Advanced Course: Software Engineering. 1972. 128-182 p.
24. EMBARCADERO_TECHNOLOGIES. Home Page, http://www.embarcadero.com/, 2007.
25. ERICH GAMMA, R. H., RALPH JOHNSON Y JOHN VLISSIDES. Design Patterns: Elements of reusable object-oriented software. Addison-Wesley, 1995. p.
26. FOWLER, M. La nueva metodología, http://www.programacionextrema.org/articulos/newMethodology.es.html, 2003.
27. GARCÍA, Z.; I. BONET, et al. Sistemas basados en el conocimiento usando Prolog. X Convención Internacional y Feria Informática 2004, Ciudad de la Habana, 2004. 8 p. 959-237-117-2
28. GARLAN, M. S. Y. D. Software Architecture: Perspectives on an emerging discipline. Upper Saddle River, Prentice Hall. 1996. p.

29. GARLAN, R. A. Y. D. A formal approach to software architectures Proceedings of the IFIP Congress '92, Setiembre de 1992.
30. GONZÁLEZ RODRÍGUEZ, E. F., DR.; L. M. E. M. PÉREZ, et al. TENSOFT II Programa para la detección del riesgo, diagnóstico, control y terapéutica de la Hipertensión Arterial. II Simposio de Hipertensión Arterial HTA 2004, 2004a: 7.
31. --- TENSOFT II Programa para la detección del riesgo, diagnóstico, control y terapéutica de la Hipetensión Arterial. II Simposio de Hipertensión Arterial HTA 2004, 2004b: 7.
32. HALGAMUGE, S. K. and M. GLESNER Neural Networks in Designing Fuzzy Systems for real World Applications Fuzzy Sets and Systems, 1994, 65: 1-12.
33. IBM_RATIONAL_SOFTWARE. Rational Rose Enterprise, http://www-306.ibm.com/software/awdtools/developer/rose/enterprise/, 2007.
34. JACKSON, M. A. Principles of Program Design. Academic Press, 1975. p.
35. JACOBSON, I.; G. BOOCH, et al. El Proceso Unificado de Desarrollo de Software. Madrid, Pearson Education SA, 2000. 458 p. Addison Wesley. 84-7829-036-2
36. JOSÉ H. CANOS, P. L., CARMEN PENADES. Métodologías Agiles en el Desarrollo de Software, http://issi.dsic.upv.es/tallerma/actas.pdf 2003.
37. KRUCHTEN, P. The 4+1 View Model of Architecture., Noviembre de 1995. pp. 42-50 p. IEEE Software
38. LARMAN, C. UML y Patrones. Introducción al analisis y diseño orientado a objetos. España, 1998. 195 a 250 p. 0-13-748880-7
39. MARCH, A. D. Estandarización en Informática aplicada a la Salud:La teoría y la práctica, www.minsal.cl/ici/2_encuentro_tic/7_Expo_AlanM-Aregentino.ppt, 2006.
40. MARTÍNEZ, I. G. Un Modelo para la Toma de Decisiones usando Razonamiento Basado en Casos en condiciones de Incertidumbre. Departamento de Ciencias de la Computación, Facultad de Matemática-Física y Computación. Santa Clara, Universidad Central Marta Abreu de las Villas, 2003. 116. p.
41. MATEOS, J. G.; A. HUERTA, et al. Modelo con RNA para Diagnosticar Hipertensión Arterial. II Simposio de Hipertensión Arterial HTA 2004. Guadalajara, Universidad de Guadalajara, División de Electrónica y Computación, Departamento de Electrónica, 2003. 52.

42. MICROSOFT. Diseño de software, http://www.microsoft.com/spanish/MSDN/estudiantes/ingsoft/ingenieria/diseno.asp., 2007a.
43. ---. Microsoft Solutions Framework, http://www.microsoft.com/technet/solutionaccelerators/msf/default.mspx, 2006.
44. ---. MSF for Agile Software Development, http://msdn2.microsoft.com/en-us/teamsystem/aa718801.aspx, 2005a.
45. ---. MSF for CMMI Process Improvement, http://msdn2.microsoft.com/en-us/teamsystem/aa718802.aspx, 2005b.
46. ---. Three-layered services application, http://msdn.microsoft.com/architecture/patterns/default.aspx?pull=/library/en-us/dnpatterns/html/ArcThreeLayeredSvcsApp.asp, 2004.
47. ---. Visual Studio Team System http://msdn2.microsoft.com/en-us/teamsystem/default.aspx, 2007b.
48. MILLS, H. D. Mathematical foundations for structured programming. IBM Report FSC 72- 6012, 1972. p.
49. MOLPECERES, A. Procesos de desarrollo: RUP, XP y FDD, www.willydev.net/descargas/articulos/general/cualxpfddrup.PDF, 2003.
50. MUÑIZ LODOS, E. Manual de HIPERTENCID. Ciudad de la Habana, Instituto Central de Investigaciones Digitales, 1994.
51. NIELSEN, J. Top Ten Web Design Mistakes of 2005, http://www.useit.com/alertbox/designmistakes.html, 2005.
52. NII, H. P. Blackboard Systems, parts 1 & 2. AI Magazine, 1 y 2.
53. PALACIO, J. Gestión y procesos en empresas de software, www.navegapolis.net/files/articulos/gestion_y_procesos.pdf 2005.
54. PÉREZ CABALLERO, D.; L. CORDIÉS JACKSON, et al. Programa Nacional de Prevención, Diagnóstico, Evaluación y Control de la Hipertensión Arterial. HIPERTENSIÓN ARTERIAL. Ciudad de la Habana, Ministerio de Salud Pública, 2004. 29.
55. PERRY, D. Software Architecture and its relevance for Software Engineering. Coord, 1997. p.
56. PRESSMAN, R. S. Ingenieria de software. Un enfoque practico. Mc Graw Hill, 1998. 614 p.

57. REYNOSO, C. B. Estilos y Patrones en la Estrategia de Arquitectura de Microsoft, http://www.microsoft.com/spanish/msdn/arquitectura/roadmap_arq/arquitectura_soft.asp, Junio de 2004.
58. ---. Introducción a la Arquitectura de Software, http://www.microsoft.com/spanish/msdn/arquitectura/roadmap_arq/intro.asp marzo 2004.
59. RICHARD TAYLOR, N. M., KENNET ANDERSON, JAMES WHITEHEAD JR, JASON ROBBINS, KARI NIES, PEYMAN OREIZY Y DEBORAH DUBROW. A component- and message-based architectural style for GUI software, Reporte para el proyecto F30602-94-C-0218.
60. RISING, L. Pattern Almanac 2000. Addison-Wesley, 2000. p.
61. RUGGERI, E. HyperTA Calc, http://www.bd-access.com.ar/HyperTACalc.htm, 2004a.
62. ---. HyperTA Control, http://www.bd-access.com.ar/HyperTA.htm, 2004b.
63. RUMBAUGH, J.; I. JACOBSON, et al. The Unified Modeling Language. Reference Manual. Massachusetts, ADDISON-WESLEY, 1998. 568 p. 0-201-30998-X
64. S. RUSELL, P. N. Artificial intelligence: a modern approach. Prentice Hall, 1995. p.
65. SEMG I Foro Intermédico, 2006.
66. SHAW, D. G. Y. M. An introduction to software architecture. CMU Software Engineering Institute Technical Report, 1994.
67. SHAW, M. Abstraction Techniques in Modern Programming Languages. IEEE Software, 1984. pp. 10-26 p.
68. STAPLETON, J. DSDM Business Focused Development. DSDM Consortium, 2003. p.
69. SYSTEMS, S. Enterprise Architect - UML Design Tools, http://www.sparxsystems.com.au/products/ea.html, 1996 - 2007. [2007]. Disponible en:
70. SZYPERSKI, C. A. Component Oriented Programming: A refined variation of Object-Oriented Programming The Oberon Tribune, Diciembre de 1995.
71. ---. Component software: Beyond Object-Oriented programming. Addison-Wesley, 2002. p.
72. TEJADA, D. H. GUIA DE PATRONES DE DISEÑO., www.Teleprogramadores.com, 2002.
73. TOLLS, I. C. CASE tools by category, http://www.cs.queensu.ca/Software-Engineering/toolcat.html, 2005.
74. TORREGROSA, F. A. Programa Corresponde. http://www.noticias.com/articulo/21-02-2006/francisco-acedo-torregrosa/se-presento-programa-corresponde-536j.html, 2006. p.

75. VISUAL_PARADIGM_INTERNATIONAL_COMPANY. Build Quality Applications Faster, Better and Cheaper, http://www.visual-paradigm.com/, 2007.
76. W. STEVENS, G. M., L. CONSTANTINE Structured Design IBM Systems Journal, 1974, 13: 115-139.
77. WIKIPEDIA. DICOM, http://es.wikipedia.org/wiki/DICOM, 2007a.
78. ---. Intercambio electrónico de datos(EDI), http://es.wikipedia.org/wiki/EDI, 2007b.
79. ---. XML, http://es.wikipedia.org/wiki/XML, 2007c.