Universidad de las Ciencias Informáticas

Facultad 7

Trabajo de Diploma para optar por el título de

Ingeniero en Ciencias Informáticas

"GENERADOR DE CÓDIGO PARA EL SISTEMA ALAS HIS"

Autor: Adrian Amado Hidalgo Chang

Noel Piloto Carbonell

Tutor: Ing. Yoandy González Martínez

Cotutor: Ing. Rosel Silva Pérez

Ing. Yicel Rivera Suárez

La Habana, junio de 2013 "Año 55 de la Revolución

"El Secreto del éxito es la constancia en los propósitos."

Benjamin Disraeli

Datos de contacto

Tutor

Ing. Yoandy González Martínez

Centro: CESIM

Departamento: Sistemas de Gestión Hospitalaria

Universidad de las Ciencias Informáticas, La Habana, Cuba

e-mail: ygonzalezm@uci.cu

Cotutor

Ing. Rosel Silva Pérez

Centro: CESIM

Departamento: Sistemas de Gestión Hospitalaria

Universidad de las Ciencias Informáticas, La Habana, Cuba

e-mail: rsperez@uci.cu

Agradecimientos

Adrian

Agradezco al compañero de tesis Noel Piloto Carbonell por su entrega para sacar adelante este trabajo.

Al tutor Yoandy González Martínez y a su esposa por ayudarnos con el desarrollo del presente trabajo.

Al cotutor Rosel Silva Pérez por contribuir con pequeños proyectos que ayudaron al desarrollo de la herramienta.

A la UCI por proporcionarme una gran fuente inagotable de conocimientos.

Noel

A la Revolución y a nuestro Comandante por crear la UCI y darnos la oportunidad de ser parte de este lindo sueño.

A mis padres por ser las personas más importantes en mi vida, por la confianza que siempre han depositado en mí, por su amor, dedicación, preocupación y consejos constantes, que han sido la causa principal de mis esfuerzos para llegar a convertirme en el profesional que hoy soy.

A mi hermano querido, por guiarme siempre y brindarme su apoyo incondicional.

A mi familia en general que siempre se han preocupado por mi recorrido a lo largo de los años y de los cuales me siento orgulloso.

A mi compañero de tesis Adrian, sin el cual no hubiese sido posible lograr este trabajo, por su constancia y entrega en el desarrollo del mismo.

A mi tutor Yoandy por encontrar la forma de ayudarnos lo mejor posible y al cotutor Rosel que contribuyó siempre en alguna medida.

A las amistades que he cosechado a lo largo de estos años intensos de universitario.

A mi hermanito Yaimel por acompañarme durante mis últimos dos años en las más disímiles situaciones, porque con él encontré un buen amigo, y pasamos juntos momentos inolvidables.

A Koki, Jorgito, Victor, Jose, Edder, Yunier, Michel, Geidar y Miguel con los cuales he aprendido y compartido inolvidables momentos, y siempre los recordaré.

Al Migue por supuesto, mi compañero de mil batallas y amigo por muy lejos que nos lleve el destino.

A todos ustedes muchas gracias.

Dedicatoria

Adrian

A mi abuelo Félix Chang Batista y mi abuela Enma Guevara Tejeda por ser mis padres durante toda mi vida.

A mi mamá Enma Gloria Chang Guevara por luchar durante 5 años para ayudarme a salir adelante con mis estudios.

Noel

Dedico este trabajo especialmente a mis padres por ser siempre sacrificados y consagrados con mi futuro y al de mi hermano, por apoyarme y ayudarme siempre, por mostrarme siempre el camino que debo tomar y aceptar mis decisiones en todo momento.

A mi hermano que fue también fuente de inspiración en que terminara mi carrera y que siempre estuvo en disposición ayudarme sin preguntar qué.

A mis tíos Miriam, Piro, Nora e Iliana por preocuparse siempre por mí y ayudarnos incondicionalmente.

A mi abuela Delia y mis abuelos que seguro estarían muy orgullosos, y a los cuales nunca pude mostrar mis logros.

A la memoria del Comandante Hugo Rafael Chávez Frías por su legado y obra que nos dejó a todos los latinoamericanos.

Resumen

Ante la necesidad de cubrir la mayor cantidad de solicitudes realizadas al Centro de Informática Médica (CESIM) respecto al sistema alas HIS, se plantea como disminuir los tiempos de desarrollo sin alterar los recursos humanos con que se cuenta. A partir de esta problemática se identificó que con la existencia de una solución que permita generar el código base para la arquitectura de esta aplicación, es posible reducir los plazos para los diferentes hitos de desarrollo. El presente trabajo tiene como objetivo principal el desarrollo de una herramienta que proporcione el entorno base para el desarrollo de acuerdo a la arquitectura alas HIS.

El desarrollo está basado en el uso de herramientas y tecnologías libres. Se utilizó para la codificación el lenguaje de programación orientado a objetos Java, mediante el Entorno de Desarrollo Integrado (IDE por sus siglas en inglés) *Netbeans* 7.0.1. Para el modelado visual se usó *Visual Paradigm* 8.0 para UML 2.0 y la metodología definida fue XP (Programación Extrema).

La solución propuesta consiste en una herramienta de escritorio concebida para usuarios que no necesariamente dominen técnicas de programación o administración de bases de datos. Propone funcionalidades en lenguaje declarativo que permiten generar el código base del desarrollo para la arquitectura alas HIS a partir de las características del negocio. Una vez generados los principales artefactos del desarrollo, se obtiene la base de datos actualizada mediante la generación por ingeniería directa que ofrece la herramienta de mapeo objeto-relacional Hibernate definida para esta arquitectura.

Palabras claves: ahorro, alas HIS, generación de código, repetividad.

Índice de contenido

Índice de tablas

Índice de figuras

Introducción

Actualmente el constante y acelerado desarrollo de la informática, exige a las empresas que desarrollan software que ofrezcan productos de alta calidad. La marcada competencia que prima en el mercado, rige la satisfacción de las expectativas y objetivos de los clientes con el empleo óptimo de recursos humanos y en el menor tiempo posible. Para lograrlo se debe tener en cuenta que por lo general las demandas son superiores a las capacidades productivas y que la producción de software de forma industrial solamente puede ser respaldada por la minoría de las entidades.

Las soluciones más viables a este conflicto se han buscado en el perfeccionamiento de los procesos y las metodologías de desarrollo. Además se ha enfatizado en la capacitación de los recursos humanos y en la constante búsqueda de nuevas tecnologías y *frameworks* para agilizar el desarrollo. Basado en lo anterior la generación de código se posiciona como uno de los procesos más codiciados por las entidades dedicadas a la producción de software. Los procesos de generación de código se logran por lo general partiendo de modelos UML o a partir de la estructura de una base de datos.

En la actualidad existen diversas herramientas que permiten la generación de código, sobre todo el correspondiente al acceso a datos, debido a que la codificación manual en esta capa requiere de mucho tiempo al involucrar diversas sentencias frecuentemente repetidas. Aunque inicialmente cualquiera de las herramientas de generación de código existentes brinda facilidades a los desarrolladores, es preciso señalar que una mala elección puede implicar pérdida de tiempo y esfuerzo por parte de los desarrolladores. Además es difícil seleccionar una que se ajuste totalmente a las necesidades específicas de cada entidad.

Las herramientas existentes en la actualidad para estos fines implican altos precios, por ejemplo la licencia por usuario del generador de código *CodeSmith* está sobre los quinientos dólares y el kit de desarrollo de software (SDK por sus siglas en inglés) sobre el cual se ejecuta tres mil dólares.

Cuba se ha encontrado inmersa desde hace años en la informatización de la sociedad incluyendo la inserción de varios centros a la producción de software de exportación. La Universidad de Ciencias Informáticas (UCI) cuenta con varios centros de desarrollo en los cuales se ha estimulado el desarrollo de herramientas que permitan generar código para determinados software de exportación que les proporcione eficacia, eficiencia y una mejor comercialización de estos en el mercado.

Uno de los centros con que cuenta la UCI es el Centro de Informática Médica (CESIM). Dentro de las prioridades del mismo se encuentra contar con la capacidad de respuesta oportuna para considerables solicitudes de utilización de uno de sus principales productos, el sistema alas HIS. La posibilidad de asumir todas las oportunidades se ha visto limitada debido a su complejidad de implementación, a las características particulares de cada entorno de salud y a los numerosos recursos humanos que requiere de acuerdo a la envergadura de la aplicación.

Tras varios años de desarrollo de este sistema se ha identificado que de acuerdo a los estándares de codificación y las pautas de implementación y diseño definidas por el Departamento de Sistemas de Gestión Hospitalaria, existe gran cantidad de código homogéneo y repetitivo. La codificación manual absoluta provoca en ocasiones la introducción involuntaria de errores por el programador. Estos pueden derivar en intensos procesos de liberación de calidad con diversas iteraciones y cientos de No conformidades detectadas por conceptos de pautas de interfaz de usuario, navegación o persistencia. Una muestra de lo anterior lo constituyó el proceso de liberación por CALISOFT entre los años 2009 y 2010 a la primera versión del sistema que duró alrededor de dieciocho meses.

En los últimos dos años países del área como México, Costa Rica, Trinidad y Tobago y Ecuador, han mostrado interés por incorporar en sus sistemas de salud la utilización del sistema alas HIS. Debido a la permanencia del grueso del equipo de desarrollo durante este período en Venezuela por contratos establecidos con la empresa PDVSA y unido a lo anteriormente expuesto, el CESIM se ha visto limitado a tomar decisiones que respondan a las solicitudes vigentes.

A partir de lo expuesto se establece como **problema a resolver:** ¿Cómo reducir los tiempos de desarrollo en la codificación del sistema alas HIS con los recursos humanos actuales?

Teniendo en cuenta lo anterior, el **objeto de estudio** de este trabajo es: “Los procesos de generación de código como vía de simplificación del trabajo de implementación de los desarrolladores” y el **campo de acción:** “Los procesos de generación de código para la arquitectura del sistema alas HIS”.

Así se obtiene que esta investigación tenga como **objetivo general** desarrollar una herramienta generadora de código para el sistema alas HIS. Para el cumplimiento de dicho objetivo se plantean las siguientes **tareas de la investigación**:

- Revisión bibliográfica de las principales herramientas generadoras de código existentes que permita evaluar la factibilidad de su uso.

- Revisión bibliográfica de los tipos de generación de código que permita seleccionar los más adecuados.
- Selección de las tecnologías, herramientas y metodología a utilizar durante el desarrollo.
- Análisis de la arquitectura del sistema alas HIS que determine patrones, buenas prácticas y reutilización, como soporte a la generación de código en la solución propuesta.
- Obtención de los artefactos correspondientes a la metodología seleccionada.
- Aplicación de los métodos de prueba para validar la herramienta.

Para la realización de la presente investigación se utilizaron algunos **métodos teóricos** y **empíricos.**

Como parte de los **métodos teóricos** utilizados se encuentran los siguientes:

- Para el estudio de la evolución y desarrollo del objeto de estudio de la investigación fue utilizado el método histórico lógico, lo que permitió realizar un análisis de las principales herramientas de generación de código existentes de acuerdo a sus características fundamentales.
- Para el análisis de la arquitectura se usó el método analítico-sintético, que permitió la extracción de elementos relevantes del diseño, identificación de patrones, reutilización y buenas prácticas a tomar en cuenta en la solución propuesta.

Se empleó el **método empírico** mediante la aplicación de entrevistas a miembros del equipo de desarrollo, para ello fueron elaborados cuestionarios con elementos particulares de la arquitectura y las tecnologías utilizadas. Esto permitió conocer elementos que limitaron el avance durante el desarrollo de la primera versión e identificar las posibles mejoras al código para futuras versiones.

El documento está estructurado de la siguiente forma: resumen, introducción, tres capítulos, conclusiones, recomendaciones, referencias bibliográficas, bibliografía y glosario de términos.

Capítulo 1: Fundamentación teórica de los procesos de generación de código

Este capítulo presenta un estudio relacionado con los procesos de generación de código y las herramientas existentes para este fin. Además se describen las características y técnicas usadas que posibilitan a la generación de código ser un proceso viable y eficiente., todo esto permitirá comprender mejor la propuesta de solución. Se definen además las tecnologías a utilizar para la construcción del software, así como la metodología a emplear.

Generadores de código

Un generador de código es una herramienta capaz de generar código de forma automática. Son programas utilizados para crear otros programas en un menor tiempo, puesto que generan código que está listo para ser compilado o interpretado. Estas herramientas simplifican el trabajo de los programadores y reducen considerablemente el tiempo de desarrollo.

Los generadores de código toman como entrada un modelo, que pudiera ser un diagrama de clases, un modelo de base de datos, e incluso un lenguaje declarativo previamente definido. El código generado depende en gran medida del modelo inicial, además de los algoritmos y patrones que tenga implementado el generador.

De forma general estos sistemas son usados para construir programas de manera automática y en un tiempo breve, creando código consistente y de alta calidad para contribuir al desarrollo de proyectos y aplicaciones informáticas. (1)

Tipos de generadores de código

Existen fundamentalmente 3 formas de generar código, estas se enuncian a continuación. (2)

- **Plantillas** (*Templating*): Se genera un armazón (o esbozo) de código fuente no funcional para ser editado, con el que se evita tener que escribir la parte más repetitiva del código que suele ser poco compleja. Usualmente es una opción recomendable.
- **Parcial:** Se implementa parcialmente la funcionalidad requerida. El programador modificará o integrará este código en función de sus necesidades. No suele ser recomendable.

- **Total:** Se genera código funcionalmente completo, que no será modificado por el programador, si fuera necesario corregir algo solo debe regenerase. Estos no suelen ser códigos demasiado complejo. Se recomienda.

Según su interacción con el código generado se clasifican en activos y pasivos. (1)

- **Activos:** Permiten regenera varias veces el mismo código variando solo la entrada. Estos generadores definen espacios de código seguros, donde el programador puede hacer los cambios que desee sin que éstos se pierdan en las sucesivas generaciones de código.
- **Pasivos:** Generan el código una sola vez y no permiten interacción alguna. Cuando de corrigen errores del generador o se cambia el diseño y se regenerar el código no permite salvar los cambios realizados manualmente, lo cual constituye una significativa desventaja.

Aspectos a tener en cuenta

Para la creación de generadores de código se deben tener en cuenta la arquitectura de software y el lenguaje de programación para el cual se va a desarrollar y el lenguaje con el que se desarrollará el propio generador. (3)

Funcionamiento de un generador de código

Para poder llevar a cabo el proceso de generación se transita por tres fases fundamentales que se especifican a continuación en orden de ejecución. (1)

Carga: La fase de carga consiste en la lectura desde un repositorio (puede ser un fichero binario, XML, una base de datos, o un diagrama UML) del modelo a traducir y crear una representación de éste en memoria. La representación de este modelo en memoria, no tiene porqué ser completa (cargar toda la información del modelo), ni tampoco seguir la misma estructura del modelo. Al contrario, la carga y las estructuras pueden ser adaptadas para cargar sólo la información necesaria y disponerla del modo que sea más conveniente para la tarea de traducción a realizar.

Inferencia: Si se han definido una serie de mecanismos de inferencia, que completan la información de modelado, éstos se ejecutan. Las estructuras del modelo en memoria son completadas y extendidas. Es necesario llevar a cabo este proceso antes de proceder a la generación propiamente dicha. El proceso es responsable de realizar precálculos útiles y de disponer adecuadamente la información para la fase posterior.

Generación: En función del código a generar se recorren secuencialmente y de modo anidado los elementos del modelo en sucesivas iteraciones. Por ejemplo, para cada clase, para cada servicio y para cada argumento. Como resultado de esta fase se obtiene el código.

Ventajas de la generación de código

La generación de código trae consigo muchas ventajas, pero muchos autores e investigadores del tema en las diferentes bibliografías que publican refieren que entre ellas se destacan cuatro muy relevantes, estas son: productividad, calidad, consistencia y abstracción. (4).

Calidad: La calidad del código generado está en correspondencia con la calidad de las plantillas y del proceso que se usa para la generación. A medida que son detectados errores y es mejorado el código de las plantillas la calidad del código generado aumenta. Se pueden imponer reglas de estilo al código generado para aumentar su homogeneidad y legibilidad. Se puede generar la documentación del código así como comentarios que faciliten su mantenimiento.

Consistencia: El código generado es extremadamente consistente. El nombre de las variables, métodos y clases es creado de la misma forma a lo largo de todo el código. La relación entre el modelo y el código generado permite que también haya consistencia entre la documentación y el código, la cual es muy difícil de mantener en un proceso de desarrollo tradicional.

Productividad: Es fácil reconocer los beneficios de la generación de código respecto a la productividad. Se comienza con un diseño de entrada e instantáneamente se obtiene una implementación de salida que responde al diseño. Por otra parte estos beneficios son mucho más apreciados cuando se regenera el código debido a un cambio en el diseño y no se pierde el trabajo que ya está hecho. Por otra parte libera a los programadores del trabajo tedioso y repetitivo permitiéndoles enfocarse en los aspectos que sí requieren un profundo análisis.

Abstracción: Muchos generadores construyen el código basados en modelos abstractos. Por ejemplo, se puede generar una capa de acceso a datos, a través de un XML que represente las tablas, los atributos y sus relaciones. Entre las ventajas que esto brinda está la portabilidad a distintas plataformas ya que elevando el nivel de abstracción no se crean especificaciones propias de una plataforma, sino que permite centrarse en el modelado de los datos o del negocio. Incluso si surgiera una nueva tecnología sólo habría que cambiar el generador y volver a generar la aplicación para que la implemente.

Desventajas de la generación de código

Mantenimiento: Cuando se usa un generador hay que darle mantenimiento constantemente. Si es desarrollado por terceros se tiene que estar al tanto de las versiones nuevas que salen y de los errores que se le van detectando. En cualquier caso hay que dedicarle esfuerzos a resolver los errores que pueda traer el generador y a agregarle las nuevas funcionalidades que van surgiendo en el mercado. Si es un generador poco usado se corre además el riesgo de que sus desarrolladores dejen de darle soporte y que por tanto en poco tiempo se vuelva obsoleto. Por otra parte el grado de sofisticación de estas herramientas dificulta mucho su mantenimiento (1).

Dominios reducidos: La generación de código tradicionalmente se ha aplicado a dominios muy específicos y bien conocidos donde es posible anticiparse a la variabilidad de problemas que pueden encontrarse en ese dominio. Los dominios o áreas de aplicación demasiado grandes o fuera de ámbito no se benefician de la aplicación de técnicas de generación de código.

Resistencia por parte de los desarrolladores: Hay programadores convencionales que ven la generación de código como una amenaza para su trabajo. Ante lo cual, toman una postura defensiva o de rechazo. Otros afirman que el uso de generadores va contra las buenas prácticas de diseño y critican mucho la idea de copiar y pegar sobre plantillas.

Estudio de las herramientas de generación de códigos existentes

JBoss Seam 2.1

JBoss Seam es un framework desarrollado por JBoss que combina a los frameworks Enterprise Java Beans (EJB por sus siglas en inglés) y Java Server Faces (JSF por sus siglas en inglés). La distribución de Seam 2.1 incluye una aplicación de línea de comandos para crear y mantener aplicaciones Seam. Esta herramienta es capaz de generar un proyecto Seam que contenga la estructura de una aplicación, incluyendo configuración y librerías. Esta es la forma más rápida de empezar un desarrollo con Seam. Genera una aplicación en blanco con *facelets, drools* (seguridad), página de autenticación y bienvenida, configuración de la base de datos, así como scripts para su compilación, empaquetado y despliegue. (5)

Durante 5 años de experiencia en el desarrollo del sistema alas HIS se ha codificado de manera manual en su implementación mediante el empleo del Entorno de Desarrollo Integrado (IDE por sus siglas en inglés) Eclipse 3.4 y el *framework JBoss* Seam 2.1. Este *framework* cuenta con la

herramienta seam-gen, capaz de generar código a partir de una base de datos relacional, además incluye plantillas para la generación de formularios, acciones, entidades y conversaciones. De esta solo se utilizan las clases entidades que genera, así como la clase java asociada a la búsqueda de dicha entidad (*EntityQuery*).

La herramienta también proporciona generación de interfaces de usuarios (UI) de las entidades presentes en la base de datos relacional para operaciones CRUD; pero esta aplicación es muy trabajosa y difícil de entender por su complejidad, lo que limita la adaptación a la arquitectura del sistema alas HIS. A continuación se enuncian las desventajas fundamentales que impidieron que el uso de seam-gen fuera la forma más viable de solución para automatizar el desarrollo del sistema alas HIS:

- Oculta gran cantidad de información acerca del código, puesto que proporciona una capa de abstracción que evita conocer cómo se ejecuta realmente, ya que genera para cada entidad una clase que hereda de *EntityHome*, siendo esta última la que brinda los métodos de adicionar, actualizar y eliminar, pero dichos métodos no pueden ser modificados o adaptados a las necesidades del programador.
- Cuando genera los archivos visuales (xhtml) y los de navegación (xml) estos por defecto son generados en el directorio raíz de la aplicación (*WebContent*), elemento que difiere con la distribución de los archivos definida en la arquitectura del sistema alas HIS.
- No genera ficheros de internacionalización (.properties) de acuerdo a los atributos de cada entidad en la base de datos, limitando la internacionalización de sistema alas HIS.
- Cuando existe en la base de datos una tabla que posee como llave primaria los atributos de otras tablas (o sea esta tabla se deriva de una relación muchos a muchos) y no tiene más atributos, la herramienta seam-gen no proporciona un mecanismo para la inserción, modificación y eliminación para esa relación.

Luego de evaluar la factibilidad de la herramienta más adecuada para cumplir los objetivos propuestos, se identificó que esta no cumplía con varías especificidades necesarias para el desarrollo establecido en el sistema alas HIS. Debido a lo anterior se determina la necesidad de continuar la investigación de las herramientas de generación de código más empleadas en el mercado nacional e internacional actual. A continuación se caracterizan y argumentan las potencialidades y deficiencias de algunas de las más importantes.

CodeChargeStudio 5.0

Fig. 1 *Logo del CodeChargeStudio 5.0*

CodeCharge Studio permite generar código de programación en C# y VB.Net (ASP.NET), VBScript (ASP), ColdFusion, Java (Servlets o JSP), PHP, y PERL. Debido a que el modelo del proyecto se almacena en formato XML, el lenguaje de programación de base puede ser cambiado en cualquier momento, de modo que el proyecto puede ser regenerado en múltiples lenguajes de programación. Esta herramienta está orientada principalmente a la creación de aplicaciones web. Entre sus principales características se encuentran: (6)

- Generación de informes Web.
- Construcción de componentes y formularios.
- Compatibilidad con múltiples bases de datos (Microsoft SQL Server, Oracle, DB2, MYSQL, Microsoft Access).
- Construcción visual de consultas.
- Traducción e internacionalización de aplicaciones web.
- Conexión simultánea a varias bases de datos.

CodeSmith Generator 6.5.2

Fig. 2 *Logo del CodeSmith Generator*

CodeSmith Generator es un generador de código basado en plantillas que permite generar códigos para cualquier lenguaje de programación. Se pueden utilizar los lenguajes C#, VB.NET o JScript.NET en sus modelos (plantillas) y los modelos de salida (código generado) puede ser en cualquier lenguaje basado en ASCII (C#, Java, VB, PHP, ASP.Net, SQL.).

Las plantillas o modelos de entrada pueden ser modificadas por el usuario para generar código de la manera deseada, sin embargo, para esto es necesario tener conocimientos avanzados sobre programación en lenguaje de marcas extensible (XML por sus siglas en inglés). La adquisición de este producto es mediante su compra en dólares estadounidenses (USD por sus siglas en inglés), los precios de su licencia se relacionan a continuación: (7)

CodeSmith Generator	**Precio**
Un único desarrollador	499 USD
Equipo de 10 desarrolladores	3 391 USD
SDK CodeSmith	2 999 USD

Tabla 1 Relación de precios de CodeSmith Generator

GeneXus

Fig. 3 *Logo del GeneXus*

GeneXus es una herramienta de desarrollo de software ágil, basada en conocimiento, orientada principalmente a aplicaciones web empresariales. El desarrollador describe sus aplicaciones en alto nivel, de manera mayormente declarativa, a partir de lo cual GeneXus genera código para múltiples plataformas (Windows, iSeries, Web, dispositivos móviles), y para múltiples lenguajes, incluyendo: Cobol, RPG, Visual Basic, Visual FoxPro, Ruby, C#, Java para múltiples plataformas móviles, incluyendo Android o Blackberry, y Objective-C para dispositivos Apple. Los Sistemas de Gestión de Bases de Datos (DBMS por sus siglas en inglés) más populares son soportados, como Microsoft SQL Server, Oracle, IBM DB2, Informix, PostgreSQL y MySQL**.** (8)

GeneXus es muy rápido para el desarrollo de aplicaciones permitiendo en poco tiempo tener resultados a la vista. A su vez para realizar prototipos para un cliente antes de darle el producto final, lo cual no implica que lo generado no pueda ser usado como producto final. Entre sus desventajas se encuentra que no está pensado para aplicaciones donde la lógica cumple un rol muy importante en la aplicación, o aplicaciones muy específicas.

Valoración de las soluciones existentes

Tras la caracterización de las herramientas realizada en el epígrafe anterior se identificó que en la actualidad existen software que son capaces de generar código para diferentes plataformas y lenguajes, utilizando procedimientos fáciles como modelos XML (***CodeCharge Studio***), plantillas (***CodeSmith Generator***), y lenguajes declarativos (***GeneXus***). Estos programas además soportan una serie de lenguajes de programación de uso actual como Java, C# y PHP, e incluyen gestores de bases de datos potentes como PostgresSQL, Oracle, Microsoft SQL Server entre otros.

Estas herramientas son capaces además de generar código para diferentes plataformas como Windows, Web y dispositivos móviles. Sus características hacen que sean programas a ser tomados en cuenta en el desarrollo de software en aras de reducir los tiempos de desarrollo y obtener un producto final con una relación tiempo-calidad adecuada a las exigencias de los clientes. Sin embargo estos programas presentan dificultades que impiden su uso en el desarrollo del sistema alas HIS, entre las que figuran:

- No están orientados a estructuras específicas dentro de una arquitectura determinada.
- En ocasiones no es posible a través de los diagramas especificarle lo que se desea hacer.
- Son sistemas privativos, lo que determina que su código sea cerrado, imposibilitando realizarle las modificaciones necesarias.
- No son herramientas multiplataforma, estas solo se ejecutan bajos sistemas operativos Windows, mientras que el desarrollo del sistema alas HIS está basado en el sistema operativo Ubuntu desde la versión 11.4 en adelante.
- Sus precios por adquisición de licencias son elevados y en el caso de GeneXus que cuenta con una versión de prueba, esta es limitada en cuanto a funcionalidades.
- Solo generan métodos básicos como insertar, eliminar, modificar.

A partir de las dificultades identificadas anteriormente se decidió desarrollar una aplicación de escritorio que esté basada en un lenguaje declarativo para obtener una representación de las entidades del negocio. Por sus características se clasificará como un generador **Pasivo** según la interacción con el código generado y se selecciona el tipo de generador basado en **Plantillas** por sus potencialidades.

Herramientas y tecnologías a utilizar

Tecnologías

Java 6. Java es un lenguaje de programación de propósito general, concurrente, basado en clases, y orientado a objetos, que fue diseñado específicamente para tener tan pocas dependencias de implementación como fuera posible, deriva mucho de su sintaxis de C y C++. Es un lenguaje independiente de la plataforma, esto significa que programas escritos en este lenguaje pueden ejecutarse igualmente en cualquier tipo de Hardware. Es libre y además cuenta en el mundo con una comunidad de más de 10 millones de usuarios que contribuyen con el desarrollo de librerías para facilitar el trabajo a los programadores que lo eligen. (9)

UML 2.0. Lenguaje Unificado de Modelado (**UML**, por sus siglas en inglés, *Unified Modeling Language*) es el lenguaje de modelado de sistemas de software más conocido y utilizado en la actualidad. Es un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema. UML ofrece un estándar para describir un "plano" del sistema (modelo), incluyendo aspectos conceptuales tales como procesos de negocio, funciones del sistema, y aspectos concretos como expresiones de lenguajes de programación, esquemas de bases de datos y compuestos reciclados. (10)

Herramientas

***NetBeans* 7.0.1.** Es un entorno de desarrollo integrado (IDE) una herramienta para que los programadores puedan escribir, compilar, depurar y ejecutar programas. Está escrito en Java pero soporta otros lenguajes de programación como Java, PHP, C, C++, Ruby, JavaScript, Perl, etc. Existe además un número importante de módulos para extender el *Netbeans* IDE. *Netbeans* IDE es un producto libre y gratuito sin restricciones de uso. (11)

***Visual Paradigm* 8.0.** Es una herramienta UML para programadores. Se escogió el *Visual Paradigm* porque es una herramienta de Ingeniería de Software Asistida por Computadora (CASE por sus siglas en inglés) que ofrece un entorno de creación de diagramas para UML; diseño centrado en casos de uso y enfocado al negocio que generan un software de mayor calidad y el uso de un lenguaje estándar común a todo el equipo de desarrollo que facilita la comunicación.

Además posee capacidades de ingeniería directa (versión profesional) e inversa; modelo y código que permanece sincronizado en todo el ciclo de desarrollo; disponibilidad de múltiples versiones,

para cada necesidad; disponibilidad de integrarse en los principales IDEs; disponibilidad en múltiples plataformas. Proporciona modelar procesos de negocio así como un generador de esqueletos de clases para Java, .NET y PHP.

***RSyntaxTextArea*-1.4.1.jar.** RSyntaxTextArea es un componente Java Swing que resalta sintaxis de texto. Se extiende de JTextComponent por lo que se integra completamente con el paquete javax.swing.text estándar. Es rápido y eficiente, y se puede utilizar en cualquier aplicación que lo necesite para editar o ver el código fuente. RSyntaxTextArea era originalmente parte del editor de texto para programadores RText, pero se está haciendo actualmente en un componente separado, reutilizable en cualquier aplicación Swing. (12)

Gtranslateapi-1.0.jar. Es una librería que utiliza la Interfaz de Programación de Aplicaciones (API por sus siglas en inglés) de traducción de Google, con el objetivo de crear aplicaciones multilenguaje, como por ejemplo traducir texto en páginas web o aplicaciones de escritorio.

Metodologías de Desarrollo

Para el desarrollo de un software con la calidad requerida y en el tiempo propuesto se requiere del trabajo en equipo y entendimiento mutuo en el desarrollo de las tareas, es por eso que la utilización de metodologías es de vital importancia para lograr los objetivos trazados. A continuación se describe la metodología aplicada en el desarrollo del sistema y la fundamentación de por qué su uso, así como la descripción de los principales aspectos que esta contiene.

Para seleccionar la metodología se debe tomar en cuenta que estas se clasifican en dos grupos:

Metodologías ágiles: Las metodologías ágiles o ligeras se encargan de valorar al individuo y las iteraciones del equipo más que a las herramientas o los procesos utilizados, resulta más importante crear un producto de software que funcione que escribir mucha documentación. El cliente está en todo momento colaborando en el proyecto. Es más importante la capacidad para responder a un cambio realizado que el seguimiento estricto de un plan.

Metodologías robustas: Las metodologías robustas o pesadas están orientadas al control de los procesos, detallan rigurosamente tanto las tareas y actividades del equipo de desarrollo como los artefactos de software. Establecen rigurosamente las actividades a desarrollar, herramientas a utilizar y notaciones que se usarán. Son las más tradicionales y altamente recomendadas para proyectos grandes y complejos, donde se requiere de una gran organización.

Visto lo anterior las metodologías robustas no serían las más recomendables para el desarrollo del sistema ya que demandan de un numeroso equipo de proyecto y generan una gran cantidad de documentación, que en ocasiones resulta inconveniente para proyectos sencillos y con poco tiempo para el desarrollo. Es por ello que teniendo en cuenta que el equipo de desarrollo es pequeño se enfoca el análisis en las metodologías ágiles, descartando así metodologías como RUP.

Dentro de las principales metodologías ágiles, Scrum y XP tienen varias similitudes; Scrum se enfoca en la gestión del proyecto siendo difícil hacer cambios una vez comenzada una iteración, generalmente hay que esperar a que concluya para poder realizarlos; mientras que XP es más enfocada a la técnica de desarrollo y es más flexible al cambio durante las iteraciones.

Scrum

Scrum más que una metodología de desarrollo de software, es una forma de auto-gestión de los equipos de desarrollo de software. Cada equipo de desarrollo decide cómo hacer sus tareas y cuánto van a tardar en ello, además ayuda a que trabajen todos juntos en la misma dirección, con un objetivo claro. (13)

Básicamente es un framework iterativo e incremental para desarrollar cualquier producto o manejar cualquier trabajo. Permite que los equipos entreguen un conjunto de funcionalidades del sistema en cada iteración, proporcionando la agilidad necesaria para responder rápidamente a los cambios de requisitos.

Desafía constantemente a sus usuarios a centrarse en la mejora, y sus sprints proporcionan la estabilidad para tratar las necesidades evolutivas que ocurren en cualquier proyecto

SCRUM define un marco para la gestión de proyectos. Es un proceso en el que se aplican de manera regular un conjunto de mejores prácticas para trabajar en equipo y obtener el mejor resultado posible de un proyecto. Está especialmente indicado para proyectos en entornos complejos, donde se necesita obtener resultados pronto, donde los requisitos son cambiantes o poco definidos, donde la innovación, la competitividad, la flexibilidad y la productividad son fundamentales.

Características

- Equipos auto dirigidos.
- Utiliza reglas para crear un entorno ágil de administración de proyectos. No prescribe prácticas específicas de ingeniería.

- Los requerimientos se capturan como ítems de la lista reserva del producto.
- El producto se construye en una serie de sprint (iteraciones) de un mes de duración.
- Usado para proyectos complejos con requerimientos cambiantes.

Xtreme Programing

Es una metodología ágil centrada en potenciar las relaciones interpersonales como clave para el éxito en desarrollo de software, promoviendo el trabajo en equipo, preocupándose por el aprendizaje de los desarrolladores, y propiciando un buen clima de trabajo. XP se basa en realimentación continua entre el cliente y el equipo de desarrollo, comunicación fluida entre todos los participantes, simplicidad en las soluciones implementadas y coraje para enfrentar los cambios. XP se define como especialmente adecuada para proyectos con requisitos imprecisos y muy cambiantes, y donde existe un alto riesgo técnico. (13)

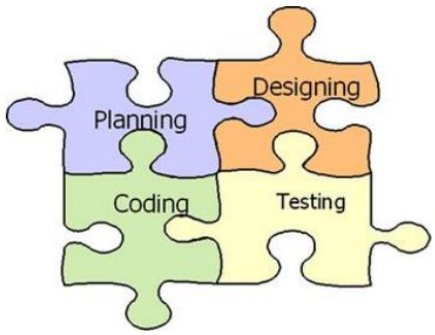

Fig. 4 *Metodología Xtreme Programing (XP)*

La metodología se basa en: (14)

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

¿Qué es lo que propone XP? (14)

Xtreme Programing (XP) propone empezar en pequeño y añadir funcionalidades con retroalimentación continua, donde el control del cambio se convierte en una parte sustantiva del proceso y el costo a estos cambios no depende de la fase o etapa. No introduce funcionalidades antes que sean necesarias. Además, el cliente o el usuario se convierte en miembro del equipo.

Lo fundamental de esta metodología es:

- ✓ La comunicación, entre los usuarios y los desarrolladores.
- ✓ La simplicidad, al desarrollar y codificar los módulos del sistema.
- ✓ La retroalimentación, concreta y frecuente del equipo de desarrollo, el cliente y los usuarios finales.

Derechos del cliente

El cliente es uno de los miembros fundamentales en el desarrollo de un software según XP, por lo que posee un conjunto de derechos dentro de los que se encuentran la capacidad de decidir qué es lo que se debe implementar. También debe conocer luego de cada iteración el estado real y el progreso del proyecto. Puede añadir, cambiar o quitar requerimientos en cualquier momento y obtener lo máximo de cada semana de trabajo, o sea, obtener un sistema funcionando cada 3 o 4 meses.

Derechos del desarrollador

Así mismo, los desarrolladores cuentan también con una serie de derechos, algunos similares a los del cliente, ya que estos pueden actuar en algunos casos como clientes. Estos pueden decidir cómo se implementarán los procesos, pedir al cliente en cualquier momento aclaraciones de los requerimientos y crear el sistema con la mejor calidad posible, estimando el esfuerzo para implementar el mismo. También pueden cambiar los requerimientos sobre la base de nuevos descubrimientos.

Ventajas de XP

Esta metodología da lugar a una programación sumamente organizada, además cuenta con una traza de errores muy pequeña que propicia la satisfacción del programador. XP facilita los cambios

permitiendo ahorrar mucho tiempo y dinero pudiendo ser aplicada a cualquier lenguaje de programación. Se hacen pruebas continuas mediante el proyecto.

Desventajas de XP

Es recomendable emplear XP solo en proyectos a corto plazo y en caso de fallar las comisiones son altas.

Para el desarrollo del sistema se cuenta con un equipo pequeño y la programación se realiza en parejas, el cliente y el desarrollador podrán añadir funcionalidades que entiendan necesarias durante toda su implementación, por lo que se precisa flexibilidad en la gestión de cambios. Por lo anterior se selecciona la metodología XP para guiar el desarrollo del sistema.

Conclusiones del capítulo

En este capítulo se realizó un estudio de los principales generadores de código utilizados en la actualidad para el proceso de desarrollo de software, con el objetivo de tomar sus ventajas y deficiencias. Se fundamentaron las herramientas y tecnologías a utilizar, seleccionando las que más se adecuen a la solución expuesta y se obtuvo como metodología de desarrollo más adecuada la *Xtreme Programing* (XP). Conocidas estas herramientas y los conceptos a utilizar se puede comenzar con el desarrollo del sistema propuesto.

Capítulo 2: Características del sistema generador de código para la arquitectura alas HIS

En este capítulo se hace un análisis objetivo de la propuesta de solución al problema de la investigación planteado, estableciendo así, las funcionalidades más importantes con las que contará el generador de código para el sistema alas HIS. Además se muestran explícitamente las historias de usuarios y los requisitos no funcionales, también serán generados los artefactos que propone la metodología XP correspondientes a la etapa de diseño y los diferentes patrones de arquitectura y de diseño que serán utilizados, así como las interfaces de la aplicación desarrollada.

Identificación del problema

Los procesos actualmente se ejecutan de manera correcta, donde el jefe del proyecto alas HIS le asigna a cada jefe de módulo las tareas a desarrollar. Luego el jefe de módulo le orienta a cada programador la tarea que debe desarrollar con su diagrama de casos de uso junto a una descripción detallada del mismo, así como el diagrama de clases y el diagrama de interacción. Una vez que el programador termina le entrega la tarea resuelta al Jefe de Módulo y este la revisa, probando su calidad, funcionalidad y nivel de terminación.

La problemática de este proceso se encuentra en la ejecución de dichas tareas, ya que en la codificación, el programador debe repetir las mismas sentencias de código múltiples veces, variando detalles como el nombre de la clase, la estructura de código embebido en lenguaje de consultas de Hibernate (HQL por sus siglas en inglés) y otra serie de parámetros. En ocasiones estas leves variaciones traen como consecuencia que se introduzcan errores cuya corrección implica tiempo al detectar el problema. Además es usual el incumplimiento de elementos como las pautas de diseño de interfaz gráfica de usuario, el estándar de codificación y las pautas de implementación, que atentan contra la organización establecida para el desarrollo.

Para dar solución al problema de la investigación antes mostrado, se determinó desarrollar una herramienta que genere automáticamente el código redundante que se encuentre en el sistema alas HIS. Se identifica además la necesidad de crear desde la herramienta nuevos módulos con sus entidades y relaciones, además de generar la estructura correspondiente en la base de datos y las interfaces gráficas de usuario de la aplicación, a partir de un lenguaje declarativo sencillo previamente establecido por el equipo de desarrollo.

Para una mejor comprensión de la estructura del sistema alas HIS, se muestra a continuación una breve descripción de la arquitectura del mismo:

El Sistema de Información Hospitalaria alas HIS es un sistema web que está compuesto por diferentes módulos, cada uno con un espacio de trabajo (*workspace*) que está compuesto por la siguiente estructura:

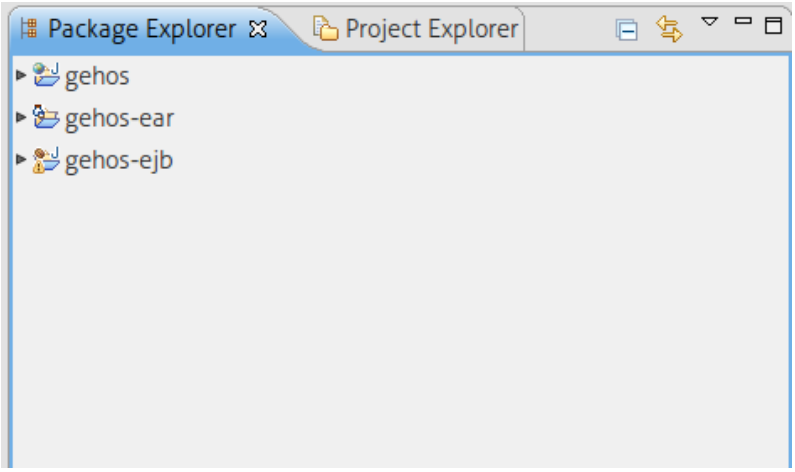

Fig. 5 *Estructura de un espacio de trabajo (workspace) de un módulo*

Dentro del directorio *gehos* se encuentra el directorio *WebContent* que es el contenedor de las páginas (xhtml) que interactúan con el usuario, además de los ficheros de internacionalización (properties) y los de navegación (xml). La distribución de los ficheros será establecida de acuerdo al módulo correspondiente, por ejemplo para el módulo “Consulta externa”, la funcionalidad de gestionar historia clínica de pediatría tendría la siguiente ruta:

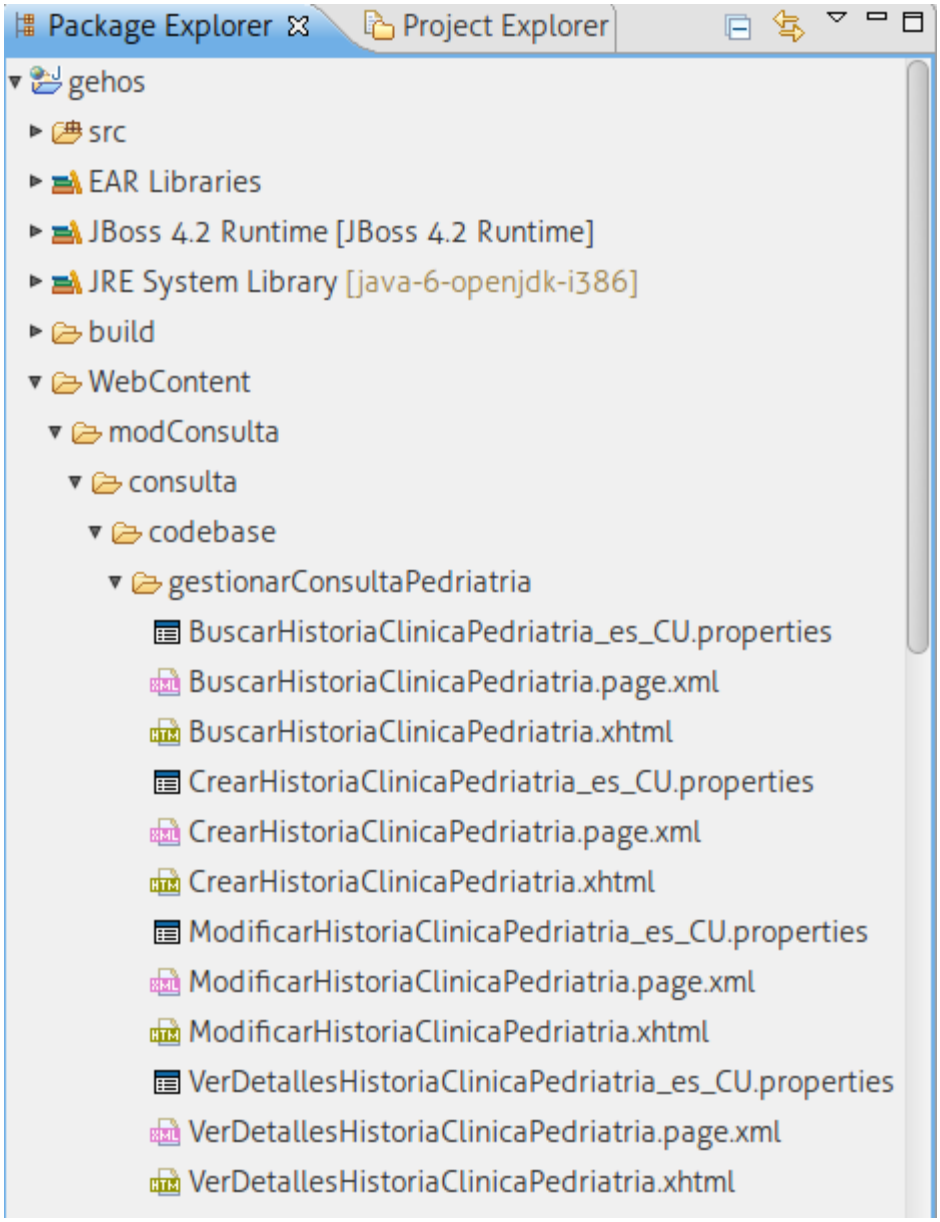

Fig. 6 *Ubicación de los ficheros de internacionalización, navegación e interfaces visuales*

Dentro del paquete *gehos-ejb* se ubican las clases controladoras y entidades que corresponden a la capa de la lógica del negocio, estas también se ubican según el módulo en cuestión. Para el caso del módulo “Consulta externa” las clases entidades que utiliza el módulo se ubican en la dirección *gehos-ejb/ejbModule/gehos/consulta/entity* y para las clases controladoras correspondientes a la funcionalidad de gestionar historia clínica de pediatría la ubicación sería *gehos-ejb/ejbModule/gehos/consulta/session/gestionarHistoriaClinicaPediatria*.

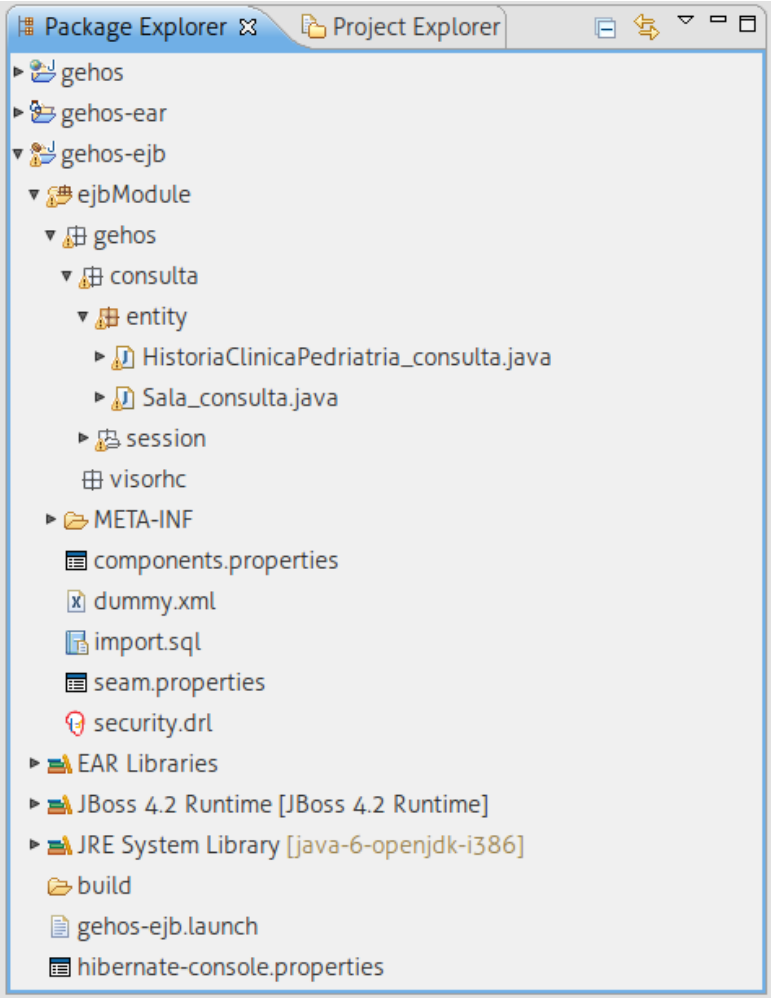

Fig. 7 *Ubicación de las entidades del negocio*

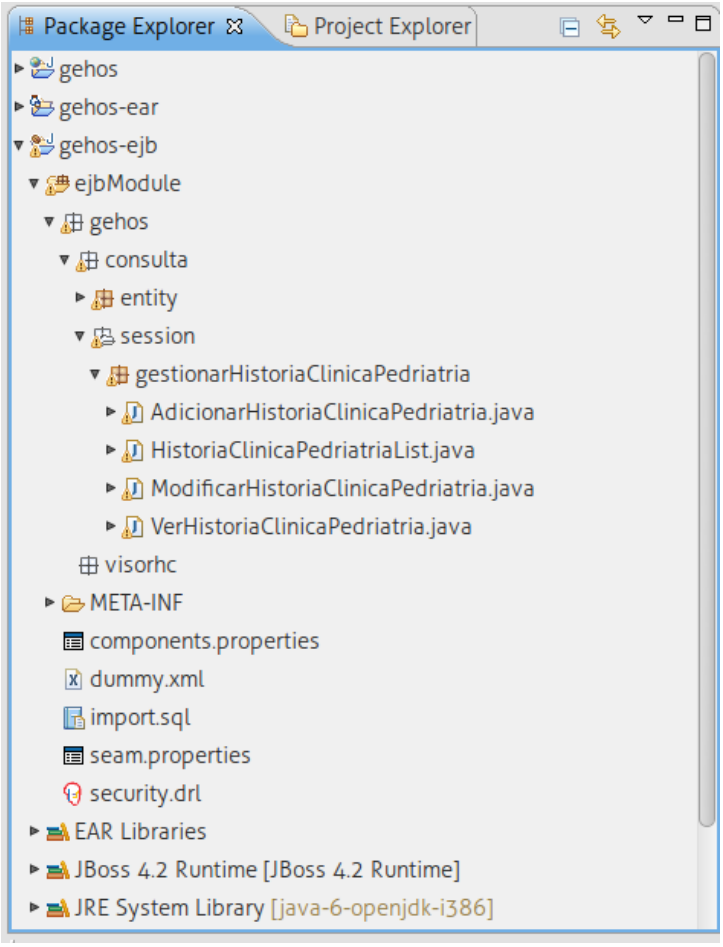

Fig. 8 *Ubicación de las clases controladoras del negocio correspondiente a gestionar historia clínica de pediatría*

Modelo de dominio

Para proporcionar mayor comprensión de los principales conceptos asociados al dominio del problema se decidió realizar el modelo de dominio. Esta selección se basa en que no existe una diferencia evidente entre los trabajadores y los actores del negocio ya que el cliente está dentro del equipo de desarrollo (solapamiento de roles), no se consigue precisar el proceso del negocio donde se pueda evidenciar claramente quiénes son las personas que desarrollan las actividades, quiénes son las que la inician y quiénes son los beneficiados con cada uno de estos procesos. Además la herramienta constituye una solución genérica que puede desplegarse en cualquier entidad que la necesite.

La metodología utilizada XP no define un modo específico para representar el negocio lo cual robustece la selección del modelo de dominio teniendo en cuenta los conceptos existentes vinculados al sistema a desarrollar, estableciendo las posibles relaciones que existen entre ellos. A continuación se muestra el modelo de dominio conformado:

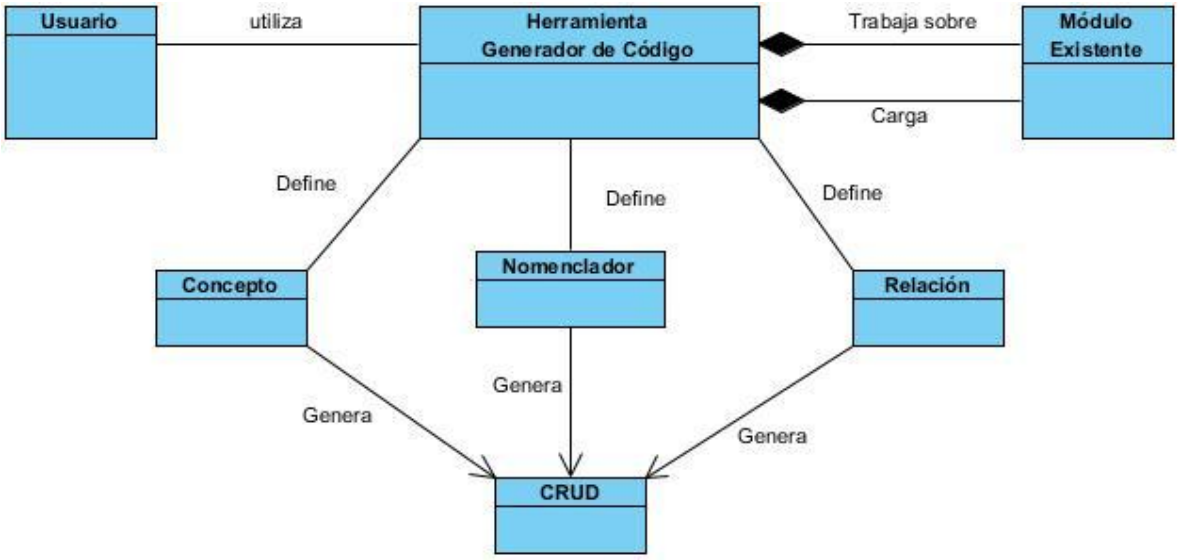

Fig. 9 *Modelo de dominio*

- **Usuario:** Persona que interactúa con el Generador de Código.
- **Herramienta generador de código:** Generador de código creado para la generación de ficheros de interfaces visuales (xhtml), de navegación (xml), y de internacionalización (properties) así como las clases controladoras y entidades para la arquitectura de alas HIS.
- **Módulo existente:** Módulo existente de un *workspace* que se selecciona de un directorio antes de iniciar la aplicación. Después de iniciada la herramienta se carga este módulo para trabajar sobre las entidades y nomencladores que este posee, y además crear nuevos conceptos o nomencladores sobre este con sus relaciones.
- **Concepto:** Define conceptos o denominaciones a las futuras entidades.
- **Nomenclador:** Es un tipo de concepto que debe tener obligatoriamente un atributo de nombre valor y de tipo texto corto.
- **Relación:** Es la relación que puede existir entre conceptos, ya sea los nuevos creados o los existentes en los módulos cargados.

- **CRUD:** Contiene las funcionalidades de Crear, Obtener, Actualizar y Borrar (**C**reate, **R**ead, **U**pdate and **D**elete) que se generan a partir de los conceptos, nomencladores y sus relaciones, para ser integradas a la arquitectura del sistema alas HIS.

Descripción del sistema propuesto

La herramienta permitirá modificar los módulos o entidades ya existentes de un workspace previamente seleccionado. Para esto, cuenta con un interfaz sencilla, donde permite la creación de conceptos y nomencladores vinculados a las entidades a crear, lo cuales son definidos y relacionados empleando un lenguaje declarativo entendible para el uso de cualquier usuario. Una vez identificados los conceptos, nomencladores y relaciones, se generan las clases de manera automática, ya sean las controladoras y entidades, visuales, XML y ficheros de internacionalización.

Historias de Usuario

Las Historias de Usuarios (HU) representan una breve descripción del comportamiento del sistema. Las mismas emplean terminologías del cliente sin lenguaje técnico, se realiza una por cada característica principal del sistema. Se emplean para hacer estimaciones de tiempo y para el plan de lanzamientos; reemplazan documentos de requisitos y presiden la creación de las pruebas de aceptación. Las HU difieren de los casos de uso porque son escritas por el cliente, no por los programadores, empleando terminologías del cliente. Las historias de usuario son más "amigables" que los casos de uso formales. (15)

En la metodología XP las historias de usuarios sustituyen a los documentos de especificación funcional y a los casos de usos. Se usan para elaborar estimaciones de tiempo de desarrollo de la aplicación. Tienen la facilidad de permitir respuestas de forma rápida a los requisitos cambiantes que puedan surgir durante el desarrollo del software, gracias a esto no se derrocha tiempo en la gestión de los mismos y se pueden administrar cómodamente los requisitos de los usuarios. (15)

Las historias de usuarios deben tener un tiempo de desarrollo de programación estimado entre una y tres semanas, si esta estimación es mayor de tres semanas, esta historia debe ser dividida en dos o más, en el caso de que sea menos de una semana debe ser combinada con otra historia. (15)

Se identificaron y redactaron 15 historias de usuarios donde se describen las funcionalidades con que deberá contar la aplicación en su versión final. A continuación se mostrará como ejemplo la

historia de usuario Generar funcionalidad “Añadir concepto”, las demás se encuentran en el expediente de proyecto.

Historia de Usuario	
Número: 1	**Nombre de la Historia de Usuario:** Generar funcionalidad Añadir concepto.
Cantidad de modificaciones a la Historia de Usuario: Ninguna	
Usuario: Yoandy González Martínez	**Iteración asignada:** 1
Prioridad en negocio: Muy Alta	**Puntos estimados:** 1 semana
Riesgo en desarrollo: Alto	**Puntos reales:** 1
Descripción: Añade un concepto que a través de un lenguaje declarativo sencillo será creado y posteriormente será tratado como una entidad. En este se establece un nombre, y el esquema al que pertenecerá en la base de datos.	
Observaciones: El concepto creado debe representar un objeto del negocio que se va a trabajar.	

Prototipo de interfaces:

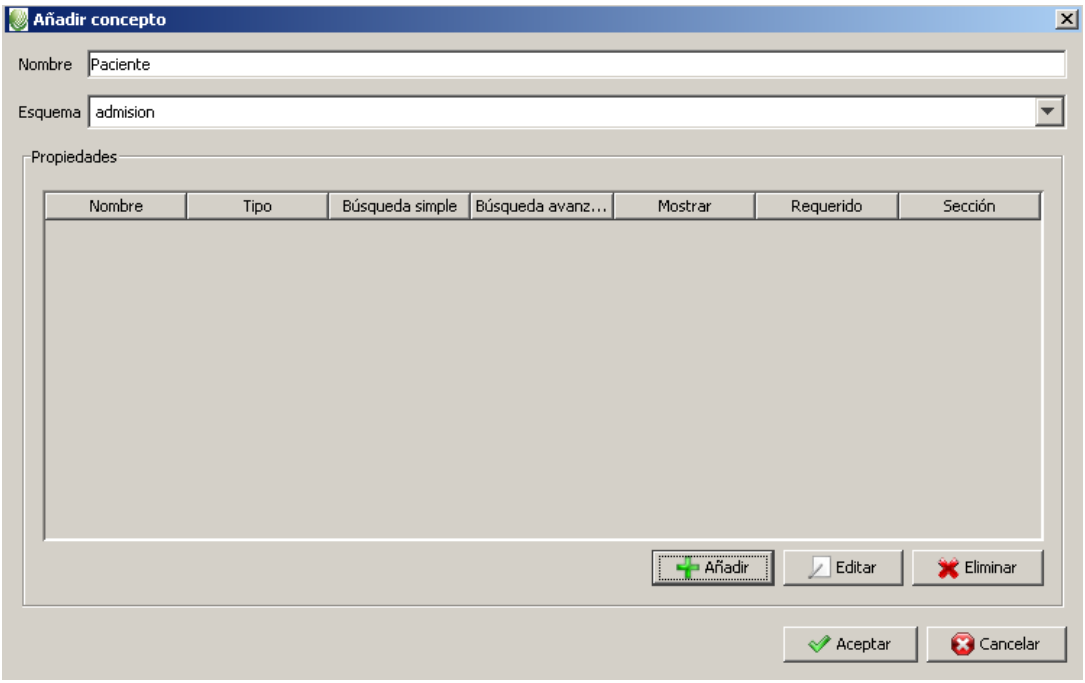

Tabla 2 HU Generar funcionalidad “Añadir concepto”

Lista de reserva del producto

La Lista de Reserva del Producto (LRP) es una tabla que contiene el listado de las historias de usuarios, ordenadas por su prioridad de implementación. Estas se clasifican en Muy Alta, Alta, Media y Baja, en esta última aparecen los requisitos de menor complejidad, además de los requisitos no funcionales del sistema a desarrollar. También aparece la estimación del tiempo de cada uno de ellos y su implementación por semanas, además del rol que hizo dicha estimación.

La LRP recoge los requisitos correspondientes al generador de código a implementar, los cuales demuestran la capacidad del mismo para ser empleado en el desarrollo del sistema alas HIS.

No.	Descripción	Estimación	Estimado por
Prioridad: Muy alta			
1	Generar funcionalidad “Cargar módulo”	2	Noel Piloto Carbonell Adrian A. Hidalgo Chang

2	Generar funcionalidad “Añadir concepto”	1	Noel Piloto Carbonell Adrian A. Hidalgo Chang
3	Generar funcionalidad “Editar concepto”	0.5	Noel Piloto Carbonell Adrian A. Hidalgo Chang
4	Generar funcionalidad “Eliminar concepto”	0.5	Noel Piloto Carbonell Adrian A. Hidalgo Chang
5	Generar funcionalidad “Añadir nomenclador”	1	Noel Piloto Carbonell Adrian A. Hidalgo Chang
6	Generar funcionalidad “Editar nomenclador”	0.5	Noel Piloto Carbonell Adrian A. Hidalgo Chang
7	Generar funcionalidad “Eliminar nomenclador”	0.5	Noel Piloto Carbonell Adrian A. Hidalgo Chang
8	Generar funcionalidad “Añadir propiedades en conceptos y nomencladores”	1	Noel Piloto Carbonell Adrian A. Hidalgo Chang
9	Generar funcionalidad “Editar propiedades en conceptos y nomencladores”	0.5	Noel Piloto Carbonell Adrian A. Hidalgo Chang
10	Generar funcionalidad “Eliminar propiedades en conceptos y nomencladores”	0.5	Noel Piloto Carbonell Adrian A. Hidalgo Chang
Prioridad: Alta			
11	Generar funcionalidad “Definir relaciones entre conceptos y/o nomencladores”	2	Noel Piloto Carbonell Adrian A. Hidalgo Chang
12	Generar funcionalidad “Eliminar relaciones entre conceptos y/o nomencladores”	1	Noel Piloto Carbonell Adrian A. Hidalgo Chang
13	Mostrar listado de conceptos y nomencladores.	1.5	Noel Piloto Carbonell Adrian A. Hidalgo Chang
14	Obtener funcionalidad “Generar código”	8	Noel Piloto Carbonell

			Adrian A. Hidalgo Chang
Prioridad: Media			
15	Generar funcionalidad “Preferencias de idiomas”.	2	Noel Piloto Carbonell Adrian A. Hidalgo Chang
Prioridad: Baja			
Requisitos no funcionales			
1	**Requisitos de rendimiento:** El tiempo de respuesta dependerá de la cantidad de entidades con las que se trabaje.		Noel Piloto Carbonell Adrian A. Hidalgo Chang
2	**Requisitos de portabilidad:** Será un sistema multiplataforma, lo que permitirá poder disponer del mismo en sistemas operativos Windows (XP o superior) y Linux (Versión del kernel 2.5.0 o superior).		Noel Piloto Carbonell Adrian A. Hidalgo Chang
3	**Requisitos de integridad:** El generador de código no sobrescribirá las entidades cargadas, solo modificará lo que el usuario necesite.		Noel Piloto Carbonell Adrian A. Hidalgo Chang
4	**Requisitos de disponibilidad:** Se podrá hacer uso de la aplicación siempre que esté instalada la máquina virtual de Java en su versión 1.6.0.		Noel Piloto Carbonell Adrian A. Hidalgo Chang
5	**Restricciones del diseño y la implementación:** Para el diseño e implementación del generador de código se utiliza la metodología XP, haciendo uso del Visual Paradigm 8.0 para UML 2.0 Edición Empresa para el modelado. También se utilizará como lenguaje de programación Java y como IDE de desarrollo Netbeans en su versión 7.0.1.		Noel Piloto Carbonell Adrian A. Hidalgo Chang
6	**Requisitos de apariencia o interfaz externa:** Interfaz intuitiva, amigable, organizada, sencilla y de fácil comprensión.		Noel Piloto Carbonell Adrian A. Hidalgo Chang
7	**Requisitos de facilidad de uso:** Para utilizar el sistema solo es necesario tener conocimientos sobre		Noel Piloto Carbonell Adrian A. Hidalgo Chang

	el negocio asociado al sistema alas HIS, no es necesario tener conocimientos de programación, ni de base de datos relacionales.		
8	**Requisitos de software:** Es multiplataforma y necesita la máquina virtual de Java 6.		Noel Piloto Carbonell Adrian A. Hidalgo Chang
9	**Requisitos de hardware:** Se necesita un mínimo de 256 MB de memoria RAM, 100 MB de espacio libre en el disco duro para almacenamiento y una velocidad de la pila del microprocesador de 300 MHz.		Noel Piloto Carbonell Adrian A. Hidalgo Chang

Tabla 3 *Lista de Reserva del Producto*

Tareas de la ingeniería

Las Tareas de la Ingeniería (TI) se usan para describir las tareas que se realizan sobre el proyecto. Estas pueden ser de: desarrollo, corrección, mejora, etc. Las TI tienen relación con una historia de usuario, aunque de una historia de usuario se puede derivar más de una tarea de la ingeniería. En estas se especifica la fecha de inicio y fin de la tarea, se nombra al programador responsable de cumplirla, también aparece el tiempo estimado que se demorará en realizar las tareas, además se realiza un pequeña descripción de lo que se tratará de hacer en la tarea.

A continuación se mostrará un ejemplo de la tarea de ingeniería, esta es correspondiente con la historia de usuario mostrada anteriormente, el resto se encuentra en el expediente de proyecto.

Tarea de Ingeniería	
Número Tarea: 1	**Número Historia de Usuario:** 1
Nombre Tarea: Crear función “Añadir concepto”.	
Tipo de Tarea : Desarrollo	**Puntos Estimados:** 1
Fecha Inicio: 1-1-2013	**Fecha Fin:** 8-1-2013
Programador Responsable: Adrian A. Hidalgo Chang	

Descripción: El sistema permitirá añadir un concepto a través de un lenguaje declarativo sencillo, dicho concepto será creado y posteriormente tratado como una entidad.

Tabla 4 TI Generar funcionalidad “Añadir concepto”

Plan de iteraciones

Las historias de usuarios seleccionadas para cada entrega son desarrolladas y probadas en un ciclo de iteración, de acuerdo al orden preestablecido. Al comienzo de cada ciclo, se realiza una reunión de planificación de la iteración. Cada historia de usuario se traduce en tareas específicas de programación. De esta misma forma, para cada historia de usuario se establecen las pruebas de aceptación.

Estas pruebas se realizan al final del ciclo en el que se desarrollan, para verificar que subsiguientes iteraciones no han afectado a las anteriores. Las pruebas de aceptación que hayan fallado en el ciclo anterior son analizadas para evaluar su corrección, así como para prever que no vuelvan a ocurrir. El sistema estará listo para su utilización al completarse la última iteración. (16)

Cada programador es responsable del número de tareas de programación que le sean asignadas, aunque todas son llevadas a cabo por parejas de programadores.

Ya identificadas las HU se establecieron tres iteraciones para el desarrollo de la aplicación. Para realizar la selección de las HU a implementar se tuvo en cuenta la prioridad que estas representan para el negocio y la relación de sus funcionalidades.

Iteración	Descripción de la iteración	Orden de la HU a implementar	Duración total
1	En esta iteración se cargará el módulo y se realizarán las HU de prioridad muy alta, estas HU se encargarán del proceso de realizar las funcionalidades principales de la aplicación, tales como agregar los conceptos y nomencladores con sus propiedades.	1-2-3-4-5-6-7-8-9-10	8 semanas

2	En esta iteración se mostrarán los listados de los conceptos y nomencladores creados y además se podrán relacionar los conceptos creados y los cargados.	11-12-13	4.5 semanas
3	El objetivo de esta iteración es que el generador de código pueda generar todas las clases controladoras y entidades, visuales, ficheros de navegación y de internacionalización, ya que todas las HU son de prioridad muy alta. Con esto se da por cumplido el objetivo final. Además se podrá cambiar el idioma en que se generarán los ficheros de internacionalización en las preferencias de idiomas.	14-15	10 semanas

Tabla 5 *Plan de iteraciones*

Diseño del sistema

En la fase de diseño de la metodología XP se aconseja el empleo de diseños sencillos para conseguir un mejor entendimiento e implementación consiguiendo reducir el tiempo y esfuerzo a la hora de desarrollar. Esta fase incluye las tarjetas clase-responsabilidades-colaboración (CRC) y diagrama de clases.

Diagrama de clases

Los diagramas de clases se utilizan para representar la estructura estática de un sistema incluyendo una colección de elementos, tales como, clases y relaciones. (17) Se dice que los diagramas de clases son “diagramas *estáticos”* porque muestran las clases, junto con sus métodos y atributos, así como las relaciones estáticas entre ellas: qué clases “*conocen”* a qué otras clases o qué clases “*son parte”* de otras clases, pero no muestran los métodos mediante los que se invocan entre ellas. (18)

Fig. 10: *Diagrama de clases*

- ***CodeGeneratorMain***: Clase que contiene las funcionalidades principales de la aplicación. Contiene un objeto de tipo *ControlCodeGenerator*.
- ***ControlCodeGenerator***: Clase controladora principal que ejecuta todas las funcionalidades fundamentales tales como la generación de clases, interfaces gráficas de usuario y ficheros de navegación XML.

- **Concepto:** Clase que contiene las propiedades del objeto que posteriormente será una entidad del negocio y estas propiedades sus atributos.
- ***ConceptoNomenclador*:** Clase que hereda de concepto y que poseerá un conjunto de valores que este tomará, así como los códigos asociados a estos.
- ***ConceptoCargado*:** Clase que hereda de concepto y que contiene los datos y dirección de un concepto que se encuentra en el módulo donde se está trabajando.
- ***Propiedad*:** Clase que contiene los valores que pasarán a ser atributos de determinado concepto.
- ***Relacion*:** Clase que relaciona dos conceptos.
- ***Esquema*:** Tipo enumerativo que contiene los esquemas que los conceptos pueden tener en la Base de Datos.
- ***Modulos*:** Muestra los nombres de los módulos en el selector.
- ***TipoPropiedad*:** Tipo enumerativo que contiene los tipos de propiedades (fecha, numérico, decisión, texto_largo, texto_corto, entero, real)
- ***TipoRelacion*:** Tipo enumerativo que contiene los tipos de relaciones entre conceptos que se pueden establecer (tiene_un, tiene_muchos).
- ***Validaciones*:** Clase que se invoca en casi toda la aplicación para poner en práctica la aplicación del estándar *CamelCase* mediante validaciones.
- ***GenerateEntity*:** Clase donde se genera las entidades con la estructura de la arquitectura del sistema alas HIS.
- ***GenerateCustomList*:** Clase donde se generan las clases controladoras del tipo CustomList con la estructura de la arquitectura del sistema alas HIS.
- ***GenerateListXHTML*:** Clase donde se generan las interfaces gráficas de usuario para listar una entidad. Son obtenidas en formato de ficheros XHTML de acuerdo con la estructura de la arquitectura del sistema alas HIS.
- ***GenerateAdd*:** Clase donde se genera la funcionalidad adicionar del CRUD correspondiente a la entidad respecto a la estructura de la arquitectura del sistema alas HIS.
- ***GenerateEdit*:** Clase donde se genera la funcionalidad Modificar del CRUD correspondiente a la entidad respecto a la estructura de la arquitectura del sistema alas HIS.
- ***GenerateView*:** Clase donde se genera la funcionalidad Ver detalles del CRUD correspondiente a la entidad respecto a la estructura de la arquitectura del sistema alas HIS.

Tarjeta CRC

El uso de las tarjetas C.R.C permite al programador centrarse y apreciar el desarrollo orientado a objetos. Estas representan objetos; la clase a la que pertenece el objeto se puede escribir en la porción superior de la tarjeta, en una columna a la izquierda se pueden escribir las responsabilidades u objetivos que debe cumplir el objeto y a la derecha, las clases que colaboran con cada responsabilidad. (19)

Las tarjetas CRC están divididas en tres partes:

- **Clase**: Representa una colección de objetos similares.
- **Responsabilidades**: Es aquello que la clase sabe o hace. En ocasiones la misma no tiene toda la información para hacerlo. Esto hace que deba interactuar con otras clases.
- **Colaboración**: Toman dos formas, un pedido de información o un pedido a que se realice una acción.

A continuación se mostrará la tarjeta CRC de la clase ***Concepto***, la cual como todas las demás desarrolladas se obtuvo del agrupamiento de varias HU. El resto de las tarjetas se encuentran descritas en el expediente de proyecto.

Tarjeta CRC	
Clase: Concepto	
Responsabilidades	**Colaboraciones**
- Crear un concepto nuevo al módulo seleccionado. - Editar un concepto nuevo al módulo seleccionado. - Añadir propiedades a los conceptos creados. - Relacionar con otros conceptos y/o nomencladores. - Mostrar la lista con los conceptos creados y/o cargados. - Eliminar un concepto creado de la lista.	

Tabla 6 *Tarjeta CRC "Concepto"*

Patrones de arquitectura

Las técnicas desarrolladas con el fin de facilitar la programación se engloban dentro de la llamada arquitectura de software o arquitectura lógica. Se refiere a un grupo de abstracciones y patrones que brindan un esquema de referencia útil para la guía en el desarrollo de software dentro de un sistema informático.

Estas arquitecturas están definidas muchas veces por el tipo de tecnología a la cual se enfrenta un programador o grupo de programadores, por lo cual algunos tipos de arquitectura son más recomendables que otras para ciertas tecnologías. (20)

Actualmente se suele usar la arquitectura *programación en capas* la cual permite distribuir el trabajo de una aplicación por niveles. El estilo arquitectural en capas se basa en una distribución ordenada de los roles y las responsabilidades para lograr brindar una división efectiva de los problemas a resolver. Los roles indican el tipo y la forma de la interacción con otras capas y las responsabilidades y funcionalidades que implementan. (21)

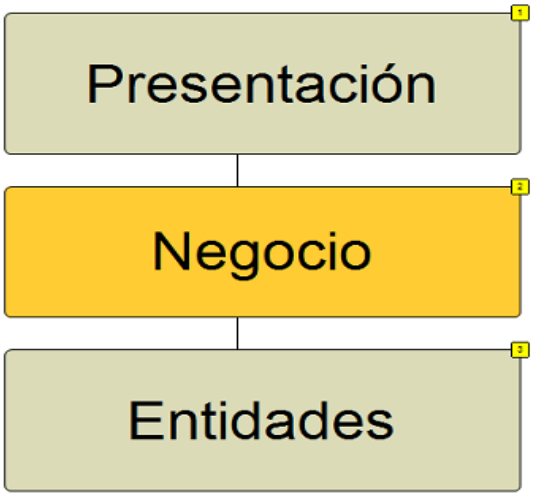

Fig. 11 *Patrón de arquitectura programación en capas*

Para esta herramienta, se hace uso del patrón *programación en capas* logrando que el sistema quede perfectamente organizado para de esta forma tener un orden lógico en la programación del mismo. A continuación se hace una breve explicación de cada una de las capas del sistema.

Capa de presentación (o interfaz del usuario): Esta capa contiene todas las clases visuales creadas para la aplicación, teniendo como clase principal *CodeGeneratorMain.java*. La capa de presentación se comunica únicamente con la capa de negocio. Es una interfaz gráfica que es amigable ante los ojos del usuario.

Capa de negocio: En esta capa se encuentra la lógica del negocio, funciona como puente entre la primera capa y la tercera, aquí es donde se reciben las peticiones del usuario y se envían las respuestas tras el proceso. En esta capa se establecen todas las reglas que deben cumplirse. La clase fundamental para controlar los procesos es *ControlCodeGenerator.java*.

Capa de información: En esta capa se evidencian todas las clases que poseen alguna información necesaria, pero que por sí solas no funcionan. Estas clases contienen toda la información que la capa de negocio necesita para realizar las operaciones. Esas clases son fundamentalmente: *Concepto.java* y *Relacion.java*.

Ahora se muestra como se encuentran distribuidas las clases según el patrón seleccionado:

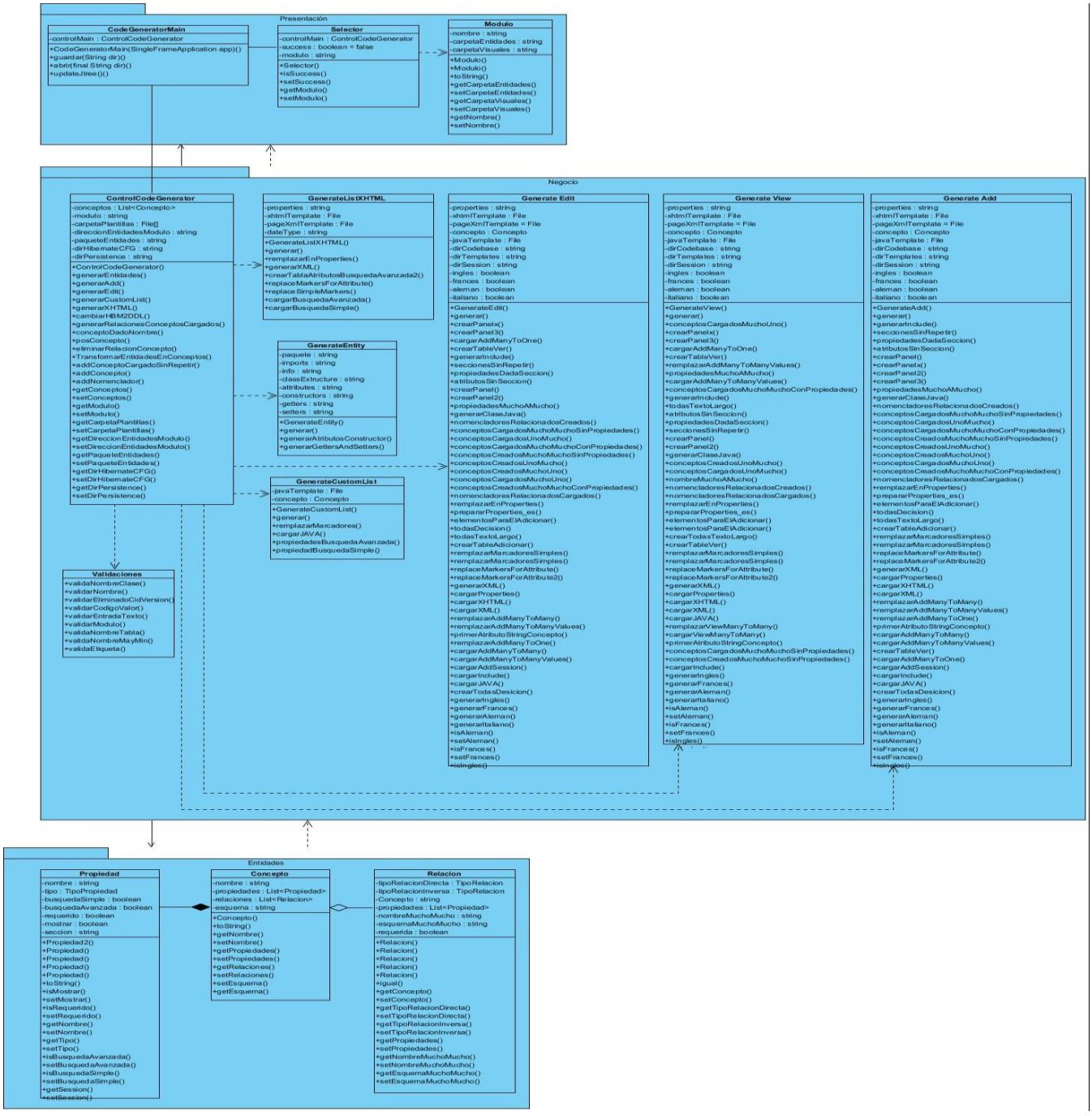

Fig. 12 *Diagrama del patrón programación en capas*

Patrones de diseño

Los patrones de diseño son el soporte de las soluciones a problemas comunes en el desarrollo de un software y de otros contornos referentes al diseño de interacción o interfaces. A su vez brindan una solución ya aprobada y documentada de problemas de desarrollo del software que están sometidos a contextos equivalentes. (22)

Un patrón de diseño es una solución a un problema de diseño. Para que una solución sea considerada un patrón debe poseer ciertas características. Una de ellas es que debe haber comprobado su efectividad resolviendo problemas similares en ocasiones anteriores. Otra es que debe ser reusable, lo que significa que es aplicable a diferentes problemas de diseño en distintas circunstancias. (22)

Existen varias familias de patrones de diseño una de ellas son los GRASP (*General Responsibility Assignment Software Patterns*). Estos son utilizados para la asignación de responsabilidades. Esta familia de patrones son una serie de buenas prácticas de aplicación recomendable en el diseño de software. Ejemplos de patrones son: experto, creador, bajo acoplamiento, alta cohesión y controlador. Para el desarrollo de esta aplicación se utilizan los siguientes patrones de diseño pertenecientes a la familia GRASP:

- **Patrón experto**: Plantea asignar una responsabilidad al experto en información, es decir, la clase que tiene la información necesaria para cumplir con la responsabilidad. El problema que resuelve el patrón experto está referido al principio más básico mediante el cual las responsabilidades son asignadas en el diseño orientado a objetos. (23)
- **Patrón creador**: Ayuda a identificar quién debe ser el responsable de la creación de nuevos objetos o clases. La nueva instancia deberá ser creada por la clase que tiene la información necesaria para realizar la creación del objeto, o usa directamente las instancias creadas del objeto. (23)
- **Alta cohesión**: Asigna una responsabilidad de modo que la cohesión sea alta. La cohesión es una medida de cuán relacionadas y enfocadas están las responsabilidades de una clase, además una alta cohesión garantiza que clases con responsabilidades estrechamente relacionadas no realicen un trabajo enorme. (23)
- **Bajo acoplamiento:** Es un principio que asigna la responsabilidad de controlar el flujo de eventos del sistema, a clases específicas; facilitando la centralización de actividades. Otros de

los beneficios es que no afectan por cambios de otros componentes, son fáciles de entender por separado y fáciles de reutilizar. (23)

- **Controlador:** Es el encargado de asignar la responsabilidad de recibir o manejar un mensaje de evento del sistema a una clase. (23)

Estándares de codificación

Para asegurar la calidad de un software uno de los instrumentos que se utilizan es la adopción de estándares de código. Los mismos tienen un sin números de ventajas como:

- Asegurar la legibilidad del código entre distintos programadores facilitando el traceo del mismo.
- Proveer una guía para el encargado de mantenimiento/actualización del sistema, con código claro y bien documentado.
- Facilitar la portabilidad entre plataformas y aplicaciones.

Es por ello que se definen para la presente aplicación varios requisitos que a continuación serán descritos:

Indentación

La unidad de indentado es de cuatro espacios. El uso de la tabulación debe ser evitado porque no existe un estándar que determine con precisión el ancho que va a producir la tabulación.

Longitud de la línea

Se debe evitar líneas con más de 80 caracteres. Cuando una sentencia sobrepase una línea simple del editor, esta deber ser separada en otras. La separación después de un operador, debe realizarse preferentemente luego de una coma. Realizar la ruptura después de un operador disminuye la probabilidad de que al realizar un copiado del código se cometa un error por olvido de la inserción del punto y coma. La siguiente línea debe ser indentada con 5 espacios.

Comentarios

Es conveniente dejar información que pueda ser leída tiempo después por personas que necesitan entender que fue lo que se hizo en el fragmento de código. Los comentarios deben ser escritos correctamente y claros. Generalmente deben usarse comentarios de una sola línea.

Declaración de variables

Cada variable debe de ser declarada en una línea y comentada. También deben aparecer ordenadas alfabéticamente: El nombre de las variables debe de comenzar con letras minúsculas y cada palabra relevante por la que esté compuesta debe ser con letra mayúscula.

Declaración de funciones

No debe haber espacio entre el nombre de la función y el paréntesis izquierdo, ni entre este y la lista de parámetros. Debe haber un espacio entre el paréntesis derecho y la llave de comienzo del cuerpo de la función. Las sentencias del cuerpo deben estar en la línea siguiente. La llave que cierra debe estar alineada con la línea de declaración de la función. Los nombres de las funciones se rigen por las mismas características que el de las variables.

Identificadores

Los identificadores pueden estar formados por cualesquiera de las 26 letras minúsculas o mayúsculas (A- Z, a - z), los 10 dígitos (0 - 9) y el carácter subrayado “_”. Debe evitarse el uso de caracteres internacionales (ej, ñ, ü) porque no siempre pueden ser leídos o entendidos correctamente en todos los lugares. No se debe usar el símbolo dólar “$” o la barra invertida “\” en los identificadores.

Sentencias Simples

Cada línea debe contener como máximo una sentencia. Debe poner un punto y coma “;” al final de cada sentencia simple. Se debe tener en cuenta que una sentencia de asignación puede resultar en la asignación de una función o de un objeto como literal y en todos los casos como sentencia de asignación debe estar finalizada con un punto y coma.

Sentencias Compuestas

Las sentencias compuestas son aquellas sentencias que contienen una lista de sentencias encerradas entre llaves:

- Las sentencias encerradas deben ser indentadas a 4 espacios.
- La llave que inicia la lista de sentencias debe estar al final de línea de la sentencia compuesta.
- La llave que termina la lista de sentencias debe estar al comienzo de una línea y guardar la misma indentación que la sentencia compuesta en correspondencia con la llave que inicia.

- Las llaves siempre serán usadas para listar todas las sentencias, aunque se trate de una sola, cuando son parte de una estructura de control como *if* o *for*. Ello facilita agregar nuevas sentencias sin la introducción accidental de errores.

Etiquetas

Las sentencias etiquetadas son opcionales. Solo estas sentencias deben ser etiquetadas: *while, do, for, foreach, switch*.

Sentencia *return*

Una sentencia *return* no debe utilizar paréntesis "()" alrededor del valor que se retorna. La expresión cuyo valor se retorna debe comenzar en la misma línea que la palabra reservada *return*, terminada con un punto y coma.

Sentencia *if*

La sentencia *if* debe ser escrita de esta manera:

- *if (condition)*

  {

  statements

  }

- *if (condition)*

  {

  statements

  } else

  {

  statements

  }

- *if (condition)*

  {

  statements

}

else if (condition)

{

statements

}

else

{

statements

}

Estructuras repetitivas

Las estructuras repetitivas deben ser escritas de esta manera:

- *for (initialization; condition; update)*

{

statements

}

- *while (condition)*

{

statements

}

foreach (valor1, valor2)

{

statements

}

Sentencia *switch*

La sentencia *switch* debe ser escrita de la siguiente forma*:*

- *switch (expression)*

  {

  case expression:

  statements

  case expression:

  statements

  default:

  statements

  }

Espacios en blanco

Las líneas en blanco facilitan la lectura determinando secciones de código lógicamente relacionada.

Los espacios en blanco pueden ser (o no deben ser) utilizados en las siguientes circunstancias:

- No se debe utilizar un espacio en blanco entre el identificador de una función y el paréntesis que abre a la lista de parámetros. Ello ayuda a distinguir entre palabras reservadas y llamadas a funciones.
- Para cualquier operador binario excepto el punto ".", el paréntesis que abre "(" y el corchete que abre "[" todos deben ser separados por un espacio entre operandos y operador.
- No se debe utilizar el espacio para separar un operador unario de su operando excepto cuando ese operador es una palabra como *typeof*.
- Cada punto y coma ";" en una sentencia de control debe ser seguido por un espacio.
- Debe dejarse un espacio luego de cada coma ",".

Declaraciones de clases

Solo debe existir un fichero con más de una clase declarada.

CamelCase

Además de estos estándares básicos que se emplean en la aplicación, también vale resaltar en empleo del tipo de escritura *CamelCase*, el cual se caracteriza porque las palabras van unidas entre sí sin espacios; con la peculiaridad de que la primera letras de cada término se encuentra en mayúscula para hacer más legible el conjunto. Tradicionalmente, se había utilizado para la formulación química; pasando a ser empleado, en la actualidad, como un lenguaje de catalogación o clasificación, utilizado exclusivamente en la Web como lenguaje de programación o simplemente para llamar la atención con fines publicitarios.

El *CamelCase* admite dos posibles combinaciones entre mayúsculas y minúsculas: la primera letra de la palabra en mayúscula y el resto también en mayúscula (*UpperCamelCase*) o, por el contrario, cuando la primera está en minúscula y las demás están en mayúscula (*lowerCamelCase*). (24)

Es decir:

- ***UpperCamelCase***, cuando la primera letra de cada una de las palabras es mayúscula. Ejemplo: *EjemploDeUpperCamelCase*.
- ***lowerCamelCase***, igual que la anterior con la excepción de que la primera palabra es con minúscula. Ejemplo: *ejemploDeLowerCamelCase*.

Interfaces principales de la aplicación

A continuación se relacionan diferentes interfaces de la aplicación que muestran las principales funcionalidades que presenta el generador de código para la arquitectura del sistema alas HIS.

Fig. 13 *Presentación de la aplicación*

En la figura 13 se muestra la presentación del Generador de Código para el sistema alas HIS, tras el cual aparecerá un selector (Figura 14) con los módulos configurados en una plantilla existente en la aplicación (Figura 15).

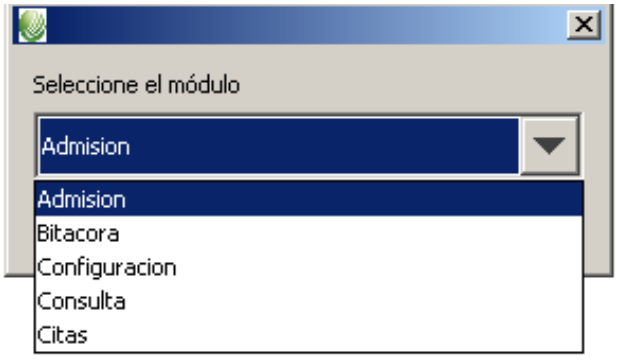

Fig. 14 *Selector de módulos existentes*

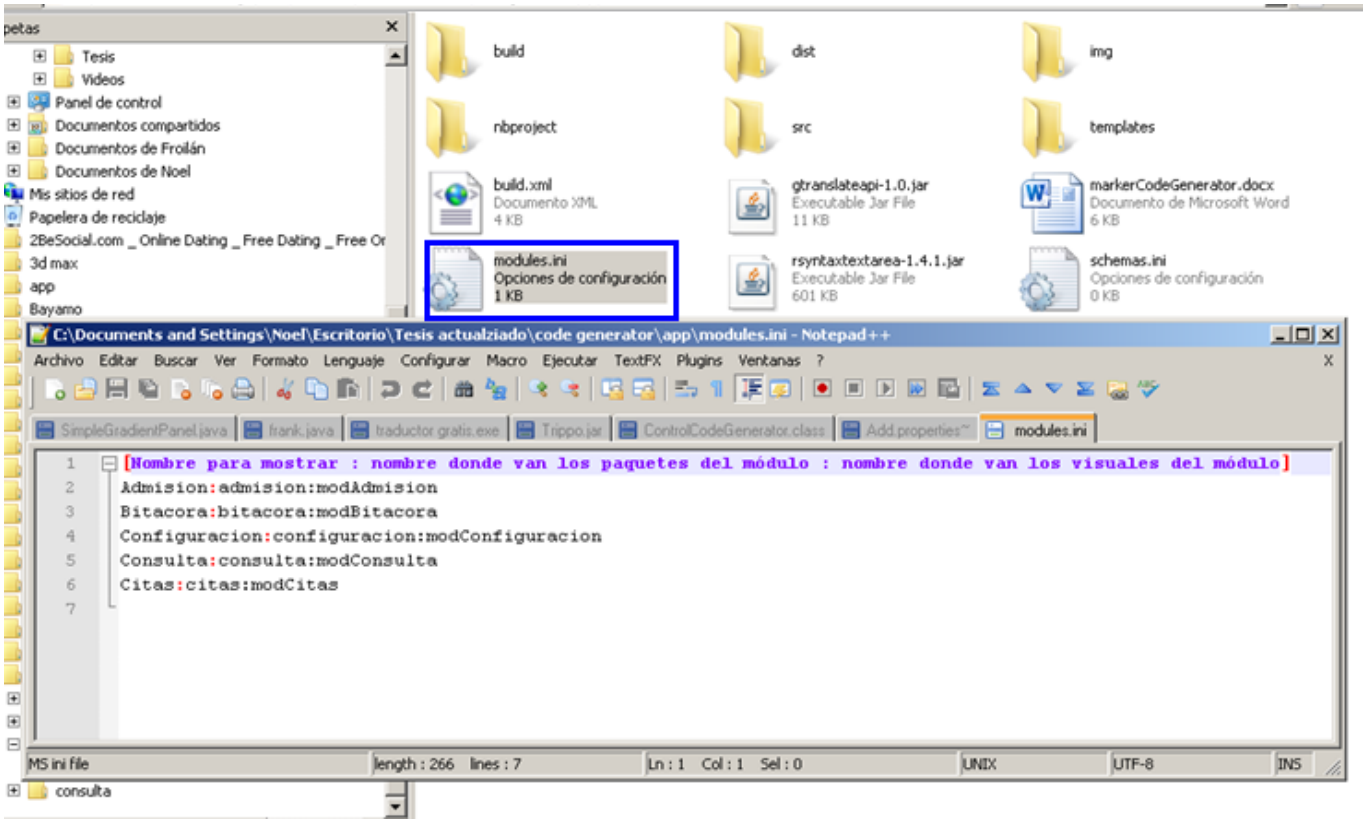

Fig. 15 *Ubicación del fichero de configuración de los módulos*

Después que se escoge el módulo sobre el cual se va a trabajar, se muestra la pantalla principal de la herramienta (figura 16) donde posee tres menús con todas las funcionalidades y tres ventanas internas, la No.1 muestra la lista de conceptos creados, nomencladores creados, conceptos cargados y nomencladores cargados; la No.2 con los datos del concepto o nomenclador marcado y el No.3 con el código del mismo.

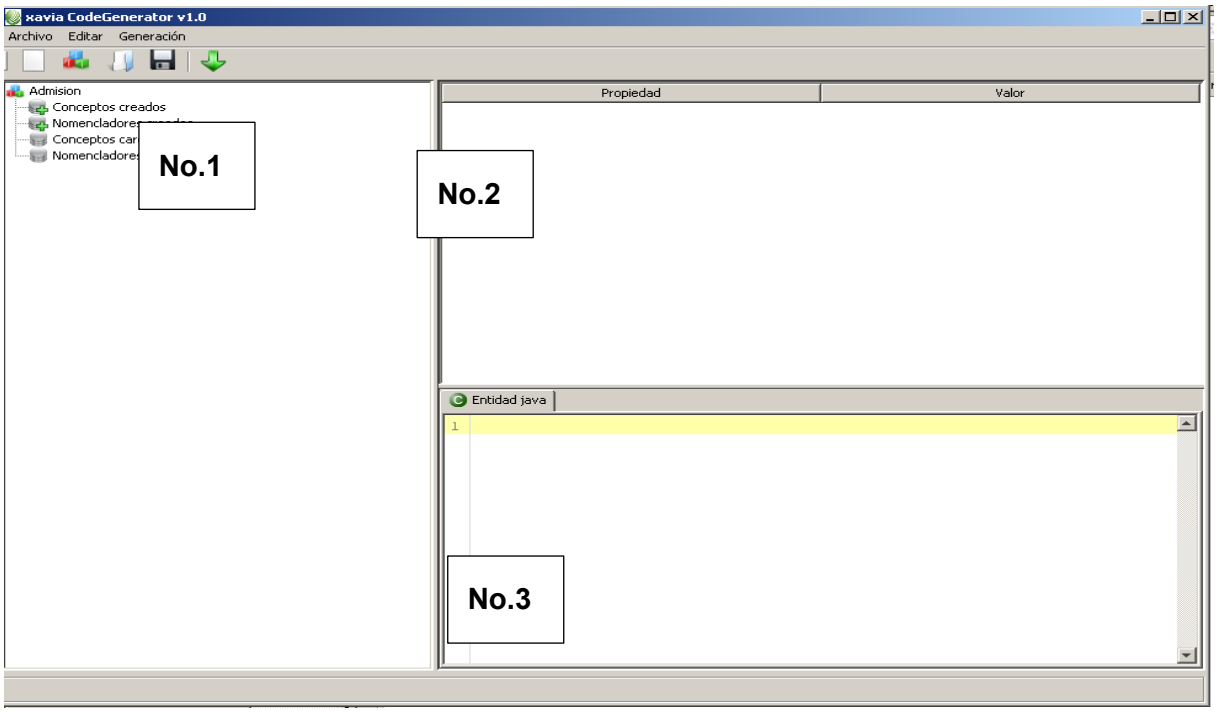

Fig. 16 *Pantalla principal del generador de código*

Antes de comenzar el trabajo con las diferentes funcionalidades se debe cargar el módulo (figura 17) seleccionado previamente que se debe encontrar en una carpeta con la estructura del módulo en cuestión, para comenzar a añadir conceptos, nomencladores y establecer las relaciones entre ellos.

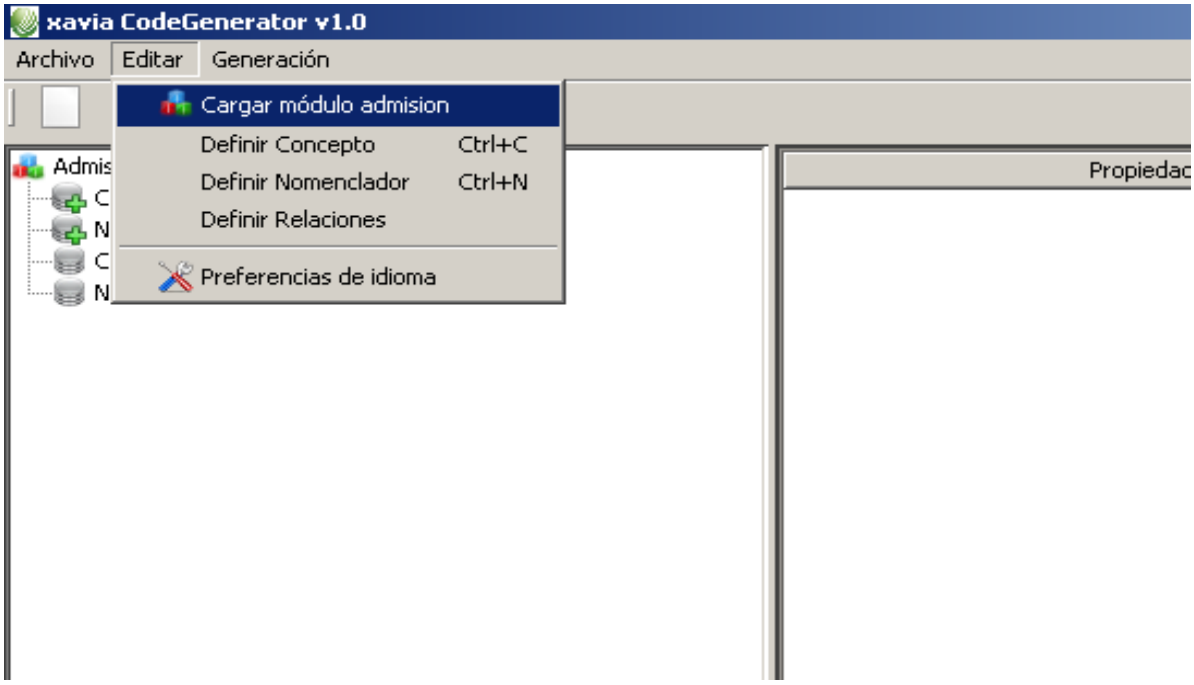

Fig. 17 *Funcionalidad "Cargar módulo"*

Después de cargado el módulo se define un nuevo concepto en el menú Editar (figura 18) con sus propiedades (figura 19). El proceso para definir un nomenclador es similar.

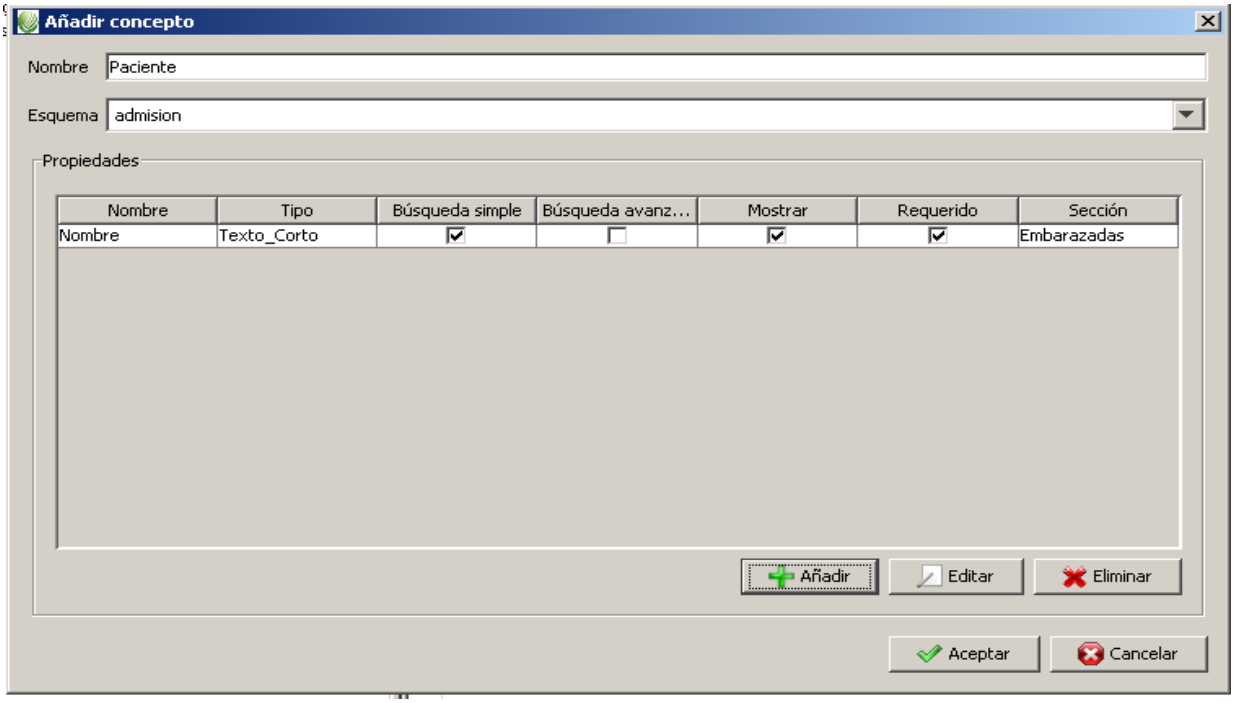

Fig. 18 *Funcionalidad "Añadir concepto"*

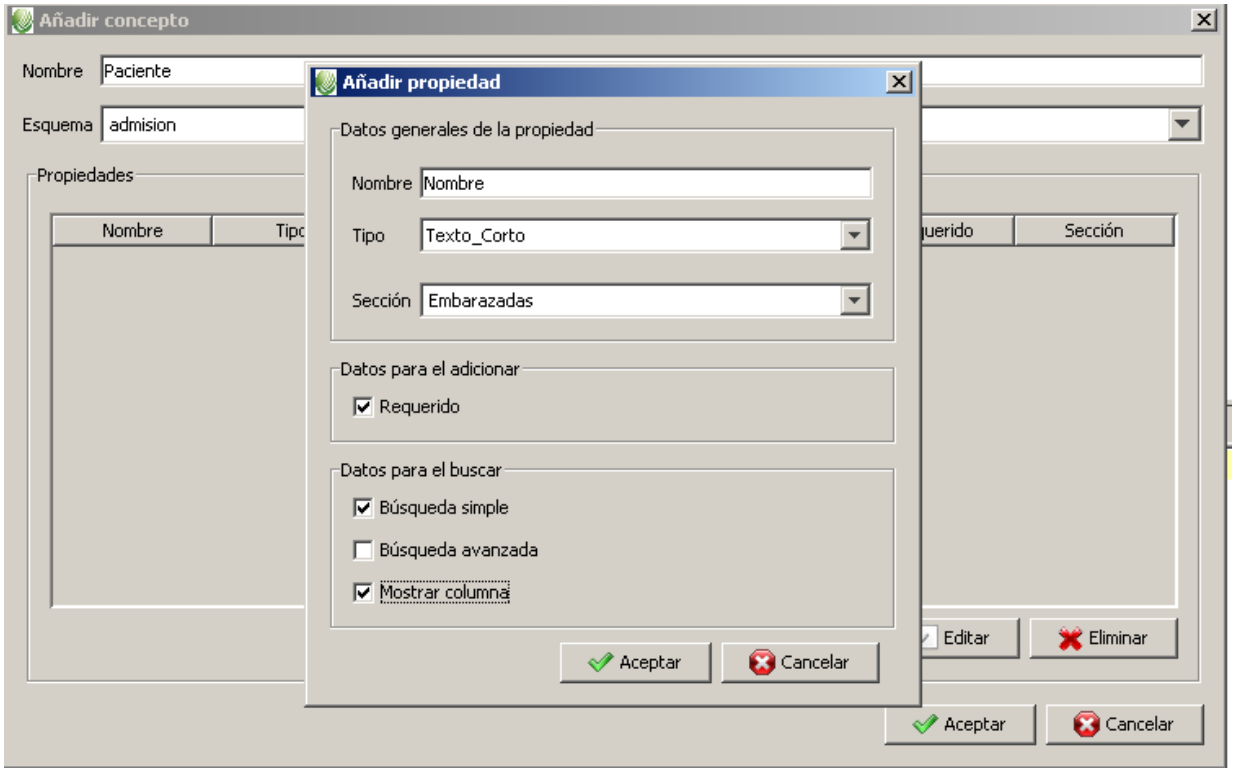

Fig. 19 *Funcionalidad "Añadir propiedad"*

Mientras se van creando los conceptos y nomencladores necesarios, se van mostrando en la pantalla principal (figura 20), al igual que los cargados inicialmente. Esta herramienta también permite relacionar un concepto o nomenclador con el resto (figura 21).

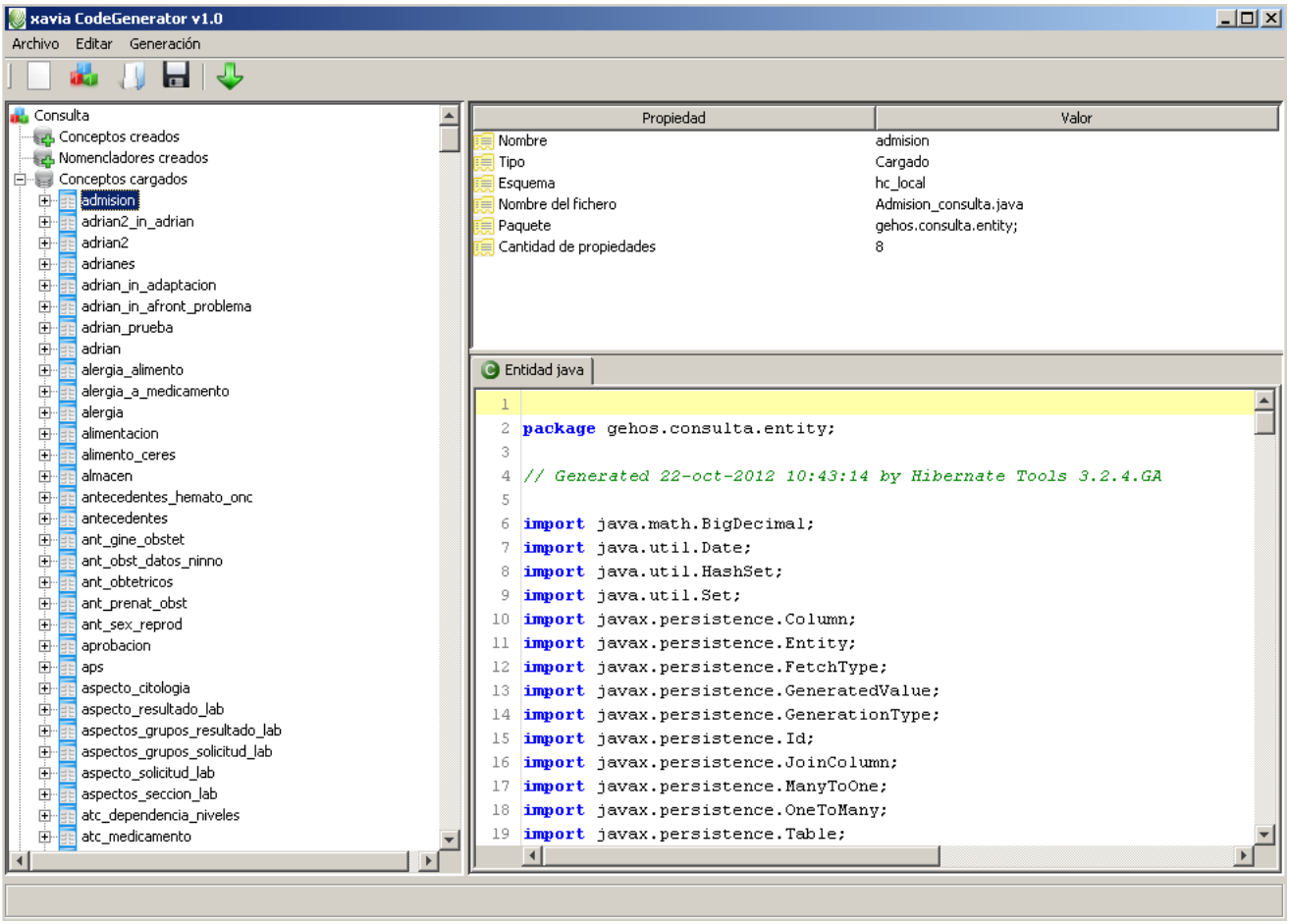

Fig. 20 *Funcionalidad "Mostrar conceptos y nomencladores"*

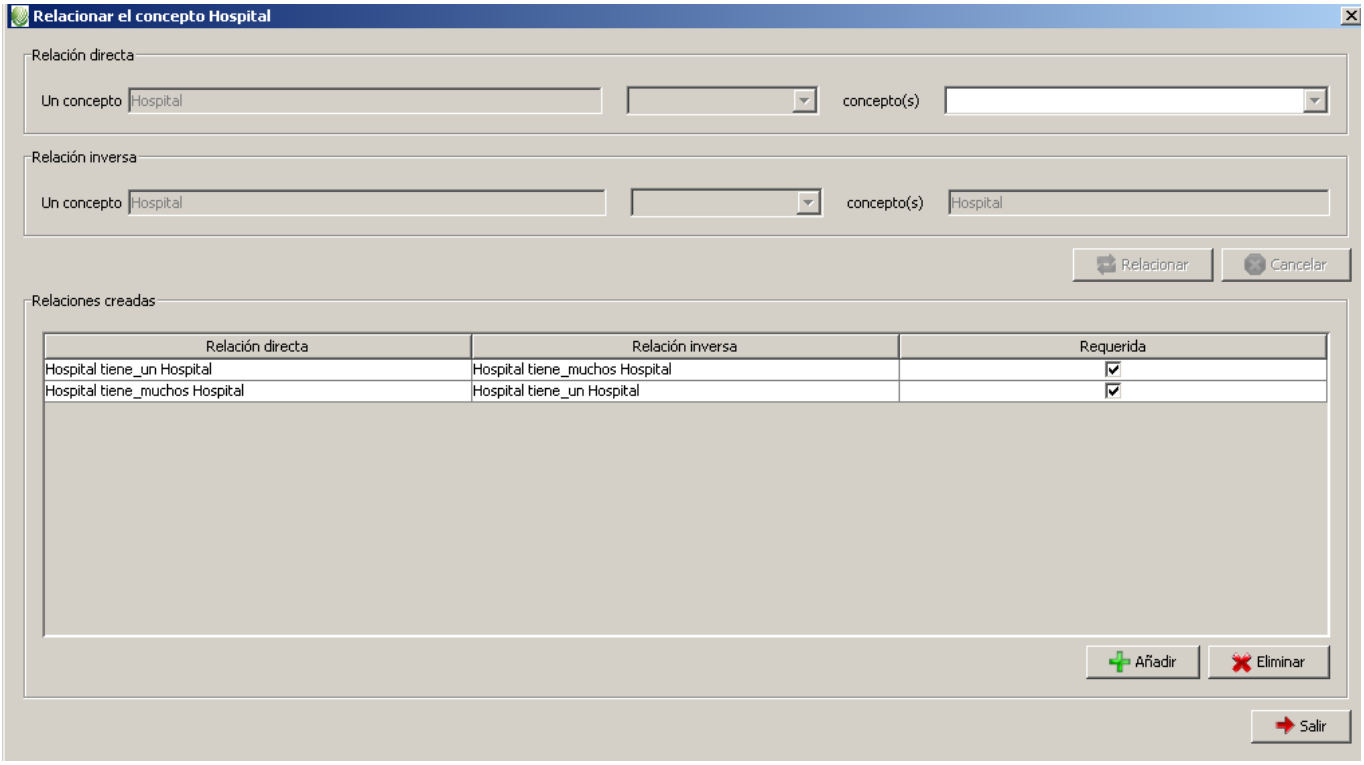

Fig. 21 *Funcionalidad "Relacionar concepto"*

Después de definidos todos los conceptos y nomencladores con sus relaciones, se puede proceder a generar el CRUD para cada entidad (concepto y/o nomenclador creados). El Generador de Código para el sistema alas HIS permite generar opcionalmente ficheros de internacionalización (properties) en varios idiomas, los cuales ya definidos pueden ser seleccionados en la funcionalidad Preferencia de idiomas del menú Editar. Esta funcionalidad requiere de conexión a internet, por lo que el usuario debe de especificar las credenciales para la autenticación en el dominio correspondiente (figura 22).

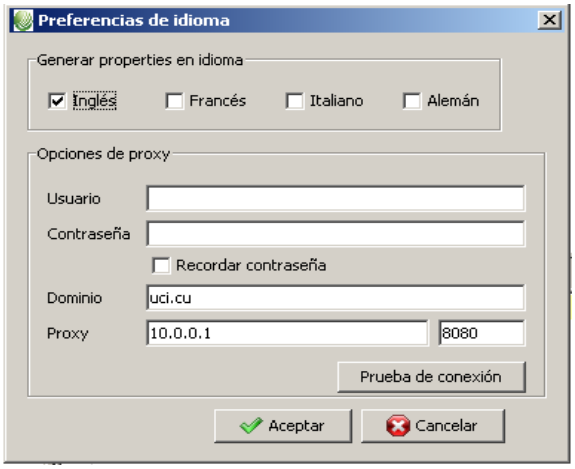

Fig. 22 *Funcionalidad "Preferencia de idiomas"*

Ahora se puede pasar al proceso de generación, antes del cual la herramienta permite seleccionar los ficheros que se van a generar, ya sea los correspondientes a las funcionalidades de Adicionar, Buscar, Modificar y Ver detalles (figura 23).

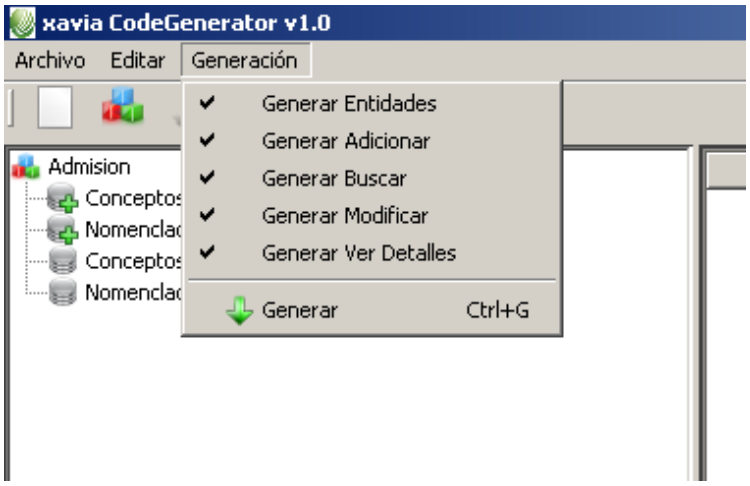

Fig. 23 *Funcionalidad "Generar"*

Al finalizar la generación se distribuirán los ficheros respetando la arquitectura establecida en el sistema alas HIS (figura 24 y 25).

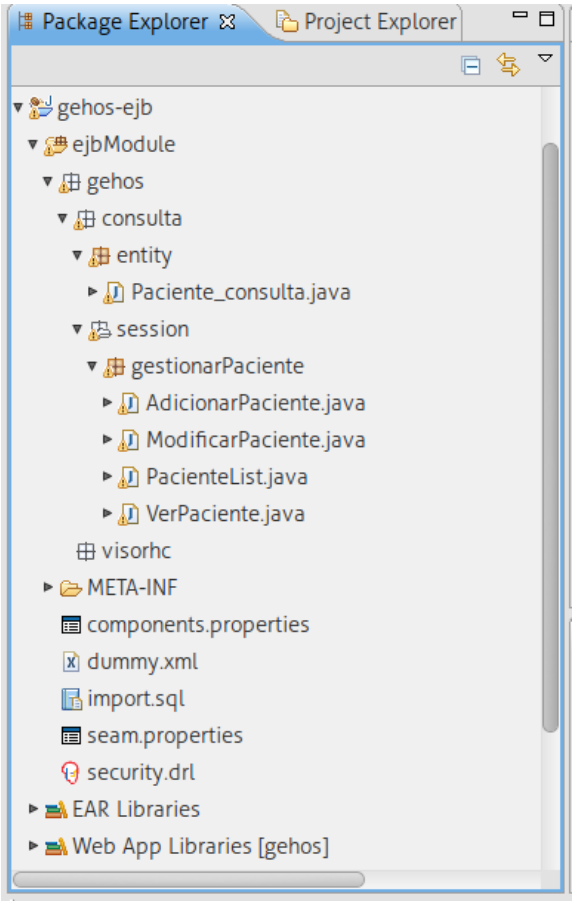

Fig. 24 *Distribución de las clases controladoras para el CRUD "Paciente generado"*

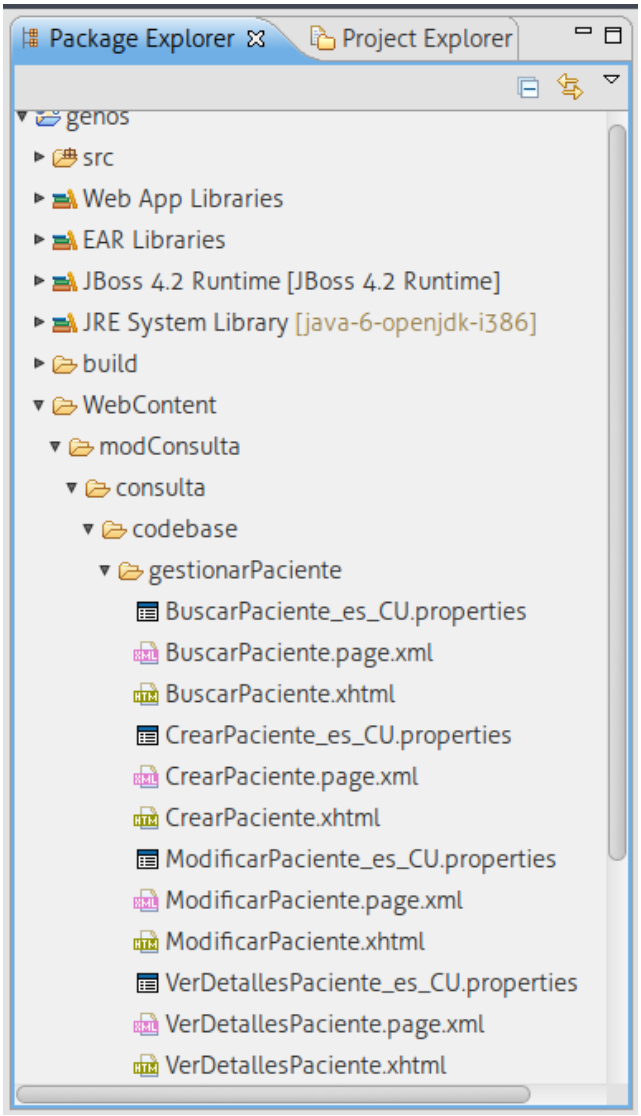

Fig. 25 *Distribución de los ficheros visuales del CRUD "Paciente generado"*

Además el generador modifica dentro del *workspace* los ficheros *hibernate.cfg.xml* y *persistence.xml* ubicados en la dirección *gehos-ear/EarContent* y *gehos-ejb/ejbModule/META-INF* respectivamente con el objetivo de cambiar la propiedad *hbm2ddl.auto* de *validate* a *update*, esto permite obtener una vez iniciado el servidor, la base de datos actualizada según las clases entidades generadas.

Conclusiones del capítulo

En el presente capítulo se describió la solución propuesta para el problema de la investigación, se identificaron los requisitos no funcionales y las historias de usuario. Además se generaron los

artefactos que admite la metodología XP como son: la lista de reserva del producto, las tareas de la ingeniería y el plan de iteraciones, los cuales son muy importantes ya que establecerán el avance de la aplicación.

También se creó el modelo de dominio, para así lograr un mejor entendimiento del desarrollo de la aplicación. Se nombraron los diferentes patrones de diseño y arquitectónicos que se utilizarán en la implementación del software.

Fueron generadas las tarjetas CRC donde en ellas se reflejan las diferentes clases que intervendrán en la aplicación con sus responsabilidades y se logró alcanzar un prototipo cercano a la aplicación final.

Capítulo 3: Validación y pruebas del sistema generador de código para la arquitectura alas HIS

Para determinar si el resultado de un producto es el deseado, se hace necesario realizarle una serie de pruebas que determinarán la calidad con la que este saldrá. El siguiente capítulo muestra las técnicas a emplear de validación y pruebas al sistema, para garantizar que las tareas realizadas en el diseño y los requisitos cumplen con las normativas planteadas por los clientes. Se determina además las acciones a realizar respecto a las técnicas seleccionadas para las pruebas con el fin de verificar que el producto funcione de acuerdo al diseño realizado y a los requisitos establecidos por el cliente.

Pruebas

La fase de pruebas es una de las más importantes en el desarrollo de una aplicación. El objetivo de cada una de ellas no es prevenir errores sino detectarlos basándose en una serie de técnicas y estrategias empleadas para cada una de las pruebas. Mediante esta filosofía se reduce el número de errores no detectados permitiendo una mejor calidad en los productos desarrollados y la seguridad de los programadores para interactuar satisfactoriamente con la introducción de algún cambio o modificación en el sistema. (25)

La metodología XP divide las pruebas en dos grupos:

- Estrategia de pruebas unitarias.
- Estrategia de pruebas de aceptación.

Como ya se ha mencionado anteriormente, XP divide las pruebas del sistema en dos grupos: pruebas unitarias, encargadas de verificar el código y diseñada por los programadores, y pruebas de aceptación o pruebas funcionales destinadas a evaluar si al final de una iteración se consiguió la funcionalidad requerida diseñadas por el cliente final. (26)

Pruebas unitarias

Una prueba unitaria es la comprobación de un módulo determinado dentro de un sistema. Son llevadas a cabo por los programadores encargados de cada módulo, los cuales aseguran que un

determinado módulo cumpla con un comportamiento esperado en forma aislada antes de ser integrado al sistema.

Los programadores deben realizar estas pruebas cuando la interfaz de un método de la aplicación no es clara, la implementación es complicada, para probar entradas y condiciones inusuales, luego de modificar algo. Éstas deben contemplar cada módulo del sistema que pueda generar fallas. Que todo código liberado pase correctamente las pruebas unitarias habilita que funcione la propiedad colectiva del código. (27)

Las pruebas unitarias no descubrirán todos los errores del código, por definición, sólo examinan porciones de código por separado. Esto quiere decir que los errores de integración, problemas de rendimiento y otros que afectan al sistema en su conjunto, no serán detectados.

Pruebas de aceptación

Las pruebas de aceptación, al igual que las de sistema, se realizan sobre el producto terminado e integrado; pero a diferencia de aquellas, están concebidas para que sea un usuario final quien detecte los posibles errores.

Estas pruebas son definidas por el cliente para cada historia de usuario, y tienen como objetivo asegurar que las funcionalidades del sistema cumplen con lo que se espera de ellas. El cliente es el rol fundamental en este tipo de prueba, ya que es el responsable de que los resultados de las pruebas sean correctos, en caso de que fallen algunas, es el encargado de indicar el orden de prioridad de resolución. (28)

Las pruebas de aceptación corresponden a una especie de documento de requerimientos en XP, ya que marcan el camino a seguir en cada iteración, indicándole al equipo de desarrollo hacia donde tiene que ir y en qué puntos o funcionalidades debe poner el mayor esfuerzo y atención. (28) Se emplean dos técnicas para las pruebas de aceptación, la prueba alfa y la prueba beta.

La prueba alfa se lleva a cabo por el cliente en el lugar de desarrollo; donde la aplicación se usa de forma natural con el desarrollador de espectador, es decir, se lleva a cabo en un entorno controlado. Se debe crear un ambiente con las mismas condiciones que se encontrará el sistema en las instalaciones del usuario. Luego se realizan las pruebas y se documentan los resultados.

La prueba beta la realiza el usuario final en el lugar de trabajo de los primeros clientes. Esta se aplica en un entorno no controlado por el desarrollador. El cliente registra todos los errores

encontrados y los informa. Como resultado de estos problemas durante la prueba beta, se realizan modificaciones, preparando así una versión de la aplicación.

Estas pruebas no se realizan durante el desarrollo, pues sería impresentable al cliente; sino que se realizan sobre el producto terminado o pudiera ser una versión del producto o una iteración pactada previamente con el cliente.

Técnica de prueba seleccionada

Cualquier proyecto que se trace una estrategia de prueba debe contar con métodos y técnicas para la aplicación de cada una de estas pruebas. A continuación se realiza una breve caracterización de dos técnicas importantes para de esta forma seleccionar una de estas técnicas para la correcta realización de las pruebas. Las pruebas unitarias suelen ser pruebas de caja blanca, mientras que las de aceptación son de caja negra.

Pruebas estructurales o de caja blanca

La prueba de la caja blanca es una técnica de diseño de casos de prueba que realiza las pruebas al código de la aplicación y que usa la estructura de control del diseño procedimental para derivar los casos de prueba. Ellas garantizan que el programador pueda ejecutar los caminos independientes de cada módulo al menos una vez y que utilice todas las estructuras de datos internas. (29)

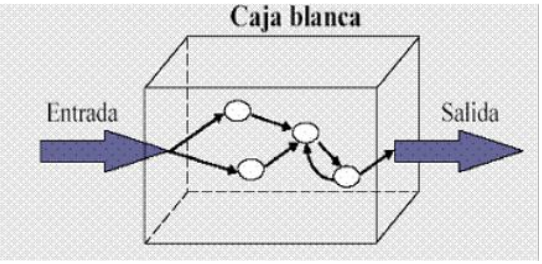

Fig. 26 *Método de Caja blanca*

En las pruebas estructurales estas se seleccionan en función del conocimiento que se tiene de la implementación del módulo. Se suelen aplicar a módulos pequeños. El probador analiza el código y deduce cuántos y qué conjuntos de valores de entrada han de probarse para que al menos se ejecute una vez cada sentencia del código. Se pueden refinar los casos de prueba que se identifican con pruebas de caja negra.

Pruebas de funcionalidad o de caja negra

También conocidas como pruebas de comportamiento, estas pruebas se basan en la especificación del programa o componente a ser probado para elaborar los casos de prueba. El componente se ve como una “caja negra” cuyo comportamiento sólo puede ser determinado estudiando sus entradas y las salidas obtenidas a partir de ellas. (30)

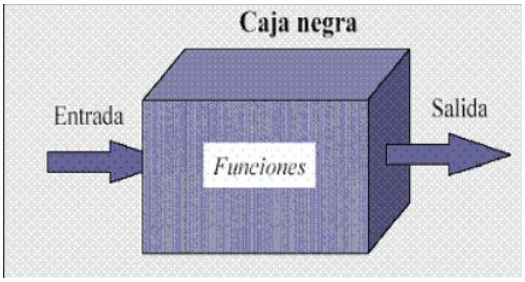

Fig. 27 *Método de Caja Negra*

Las pruebas de caja negra buscan mostrar las funcionalidades del software en estados óptimos y operativos; aceptando de forma adecuada la entrada de datos y produciendo una salida correcta. Este tipo de prueba permite demostrarle al cliente que la aplicación puede satisfacer las necesidades del mismo.

A continuación se muestran algunas variantes de pruebas de caja negra:

- **Métodos de prueba basados en gráficos:** en este método se debe entender los objetos (objetos de datos, objetos de programa tales como módulos o colecciones de sentencias del lenguaje de programación) que se modelan en el software y las relaciones que conectan a estos objetos. Una vez que se ha llevado a cabo esto, el siguiente paso es definir una serie de pruebas que verifiquen que todos los objetos tienen entre ellos las relaciones esperadas. (31)
- **Partición equivalente:** se presenta la partición equivalente como un método de prueba de caja negra que divide el campo de entrada de un programa en clases de datos de los que se pueden derivar casos de prueba. Un caso de prueba ideal descubre de forma inmediata una clase de errores que, de otro modo, requerirían la ejecución de muchos casos antes de detectar el error genérico. La partición equivalente se dirige a la definición de casos de prueba que descubran clases de errores, reduciendo así el número total de casos de prueba que hay que desarrollar. El objetivo de partición equivalente es reducir el posible conjunto de casos de prueba en uno más pequeño, un conjunto manejable que evalúe bien el software. (31)

- **Análisis de valores límite:** los errores tienden a darse más en los límites del campo de entrada que en el centro. Por ello, se ha desarrollado el análisis de valores límites (AVL) como técnica de prueba. El análisis de valores límite lleva a una elección de casos de prueba que ejerciten los valores límite. El análisis de valores límite es una técnica de diseño de casos de prueba que completa a la partición equivalente. En lugar de seleccionar cualquier elemento de una clase de equivalencia, el AVL lleva a la elección de casos de prueba en los extremos de la clase. En lugar de centrarse solamente en las condiciones de entrada, el AVL obtiene casos de prueba también para el campo de salida. (31)

Casos de pruebas basados en historias de usuarios

Luego del estudio realizado se estimó llevar a cabo la estrategia de pruebas de aceptación de tipo alfa utilizando la técnica de partición de equivalencia en los métodos de prueba de caja negra. Con esta estrategia la presentación de los resultados es importante, en cambio para las pruebas unitarias no tiene mucho sentido ya que siempre se requiere un total de efectividad en los módulos más críticos. Es más efectivo que el cliente de una aceptación del producto que desea, a que el producto lo pruebe el programador.

La técnica de partición equivalente intenta dividir el dominio de entrada de un programa en un número finito de clases de datos o clases de equivalencia de los que se pueden derivar los casos de prueba; de tal modo que se pueda asumir razonablemente que una prueba realizada con un valor representativo de cada clase es equivalente a una prueba realizada con cualquier otro valor de dicha clase. (32)

Esto quiere decir que si el caso de prueba correspondiente a una clase de equivalencia detecta un error, el resto de los casos de prueba de dicha clase de equivalencia deben detectar el mismo error. En contraposición, si un caso de prueba no ha detectado ningún error, es de esperar que ninguno de los casos de prueba correspondientes a la misma clase de equivalencia encuentre ningún error.

El diseño de casos de prueba según esta técnica consta de dos pasos:

1. Identificar las clases de equivalencia.
2. Identificar los casos de prueba.

Dado que las pruebas de aceptación son basadas en HU se comprueba si realmente la implementación de la HU es satisfactoria y si el sistema cumple con el funcionamiento esperado por el cliente. Para llevar a cabo la prueba, antes deben ser diseñados los casos de pruebas.

En la actividad Diseño de los casos de prueba, usando técnicas de caja negra se desarrollan los casos de prueba. Esta actividad incluye diseñar las pruebas e identificar los datos de prueba. Para cada funcionalidad a probar, se crea una matriz que muestra su correspondencia con los casos de prueba, conociéndose de esta forma qué elementos cubren los casos de pruebas. (34)

A través de los casos de pruebas se puede probar si la aplicación realmente funciona, si es así, en ellos se describen los diferentes escenarios y se indica la respuesta correcta que el sistema debe mostrar en cada escenario. Con el propósito de comprobar que todas las HU de una aplicación son revisadas, debe haber al menos un caso de prueba para cada una de ellas. A continuación se muestra un ejemplo del Caso de Prueba 1 Gestionar Concepto basado en las HU Añadir concepto, Editar concepto y Eliminar concepto:

Descripción general de la HU

Este caso de prueba permitirá describir las historias de usuario referidas a añadir, modificar y eliminar un concepto nuevo al módulo en el cual se está trabajando.

Condiciones de ejecución

Debe haberse cargado uno de los módulos existentes.

SC <Añadir concepto>					
Escenario	**Descripción**	**V1_Nombre**	**V2_Esquema**	**Respuesta del sistema**	**Flujo central**
EC 1.1 Añadir concepto con datos correctos	El usuario deberá especificar el nombre del concepto empleando letras y/o números, elegir el tipo de esquema en la base de datos que este tendrá.	V	N/A	Mostrar el concepto en una lista con todos los conceptos creados.	El usuario hace clic en la opción Definir Concepto en el menú Editar. Ya dentro introduce el nombre del concepto, elige el tipo de esquema que tendrá en la BD. Después añade las propiedades **(Ver CP Añadir Propiedades)** y si desea añadir el concepto acciona la opción Aceptar, sino Cancelar. Si acciona Aceptar, el sistema muestra el concepto en una lista con todos los conceptos creados.
		Paciente	Admisión		
EC 1.2	El usuario	I	N/A	El sistema	El usuario hace clic en la opción

Añadir concepto con datos incorrectos	deberá especificar el nombre del concepto empleando caracteres especiales o diferentes de letras y números.			muestra un mensaje el mensaje “El nombre del concepto tiene caracteres no válidos.”	Definir Concepto en el menú Editar. Ya dentro introduce el nombre del concepto, elige el tipo de esquema que tendrá en la BD. Después añade las propiedades (**Ver CP Añadir Propiedades**) y si desea añadir el concepto acciona la opción Aceptar, sino Cancelar. Si acciona Aceptar, el sistema muestra el concepto en una lista con todos los conceptos creados.
		Pa*iente	Admisión		
EC 1.3 Añadir concepto con campos vacíos	El usuario deberá dejar los campos vacíos	V	N/A	El sistema muestra un mensaje el mensaje “Debe especificar el nombre del concepto.”	El usuario hace clic en la opción Definir Concepto en el menú Editar. Ya dentro introduce el nombre del concepto, elige el tipo de esquema que tendrá en la BD. Después añade las propiedades (**Ver CP Añadir Propiedades**) y si desea añadir el concepto acciona la opción Aceptar, sino Cancelar. Si acciona Aceptar, el sistema muestra un mensaje el mensaje “Debe especificar el nombre del concepto.”
		()	Admisión		
EC 1.4 Añadir concepto sin propiedades	El usuario deberá definir un concepto sin añadir ninguna propiedad.	V	N/A	El sistema muestra un mensaje el mensaje “El concepto debe tener propiedades.”	El usuario hace clic en la opción Definir Concepto en el menú Editar. Ya dentro introduce el nombre del concepto, elige el tipo de esquema que tendrá en la BD. Después añade las propiedades (**Ver CP Añadir Propiedades**) y si desea añadir el concepto acciona la opción Aceptar, sino Cancelar. Si acciona Aceptar, el sistema muestra un mensaje el mensaje “El concepto debe tener propiedades.”
		Paciente	Admisión		
EC 1.5	El usuario	N/A	N/A	El sistema	El usuario hace clic en la opción

Añadir concepto con ninguna propiedad con búsqueda simple	deberá tener el campo búsqueda simple de todas las propiedades sin marcar.			muestra un mensaje el mensaje “Debe marcar al menos una propiedad para búsqueda simple.”	Definir Concepto en el menú Editar. Ya dentro introduce el nombre del concepto, elige el tipo de esquema que tendrá en la BD. Después añade las propiedades (**Ver CP Añadir Propiedades**) y si desea añadir el concepto acciona la opción Aceptar, sino Cancelar. Si acciona Aceptar, el sistema muestra un mensaje el mensaje “Debe marcar al menos una propiedad para búsqueda simple.”
		()	Admisión		
EC 1.6 Añadir concepto con esquema incorrecto.	El usuario deberá especificar el nombre del esquema empleando un nombre distinto a los ya definidos.	N/A	N/A	El sistema muestra un mensaje el mensaje “El esquema no es válido.”	El usuario hace clic en la opción Definir Concepto en el menú Editar. Ya dentro introduce el nombre del concepto, elige el tipo de esquema que tendrá en la BD. Después añade las propiedades (**Ver CP Añadir Propiedades**) y si desea añadir el concepto acciona la opción Aceptar, sino Cancelar. Si acciona Aceptar, el sistema muestra un mensaje el mensaje “El esquema no es válido.”
		Paciente	Admasd/65		

SC <Editar concepto>					
Escenario	**Descripción**	**V1_Nombre**	**V2_Esquema**	**Respuesta del sistema**	**Flujo central**
EC 1.1 Editar concepto con datos correctos	El usuario deberá modificar el nombre del concepto empleando letras y/o números, elegir el tipo de esquema en la base de datos que este tendrá.	V	N/A	Mostrar el concepto en una lista con todos los conceptos creados.	El usuario hace clic derecho encima del concepto creado y da en la opción Editar. Ya dentro modifica los campos que desee. Si desea añadir el concepto acciona la opción Aceptar, sino Cancelar. Si acciona Aceptar, el sistema muestra el concepto
		Consulta	Admisión		

					en una lista con todos los conceptos creados.
EC 1.2 Editar concepto con datos incorrectos	El usuario deberá modificar el nombre del concepto empleando caracteres especiales o diferentes de letras y números.	I	N/A	El sistema muestra un mensaje el mensaje “El nombre del concepto tiene caracteres no válidos.”	El usuario da clic derecho encima del concepto creado y da en la opción Editar. Ya dentro modifica los campos que desee. Si desea añadir el concepto acciona la opción Aceptar, sino Cancelar. Si acciona Aceptar, el sistema muestra un mensaje el mensaje “El nombre del concepto tiene caracteres no válidos.”
		Consu*ta	Admisión		
EC 1.3 Editar concepto con campos vacíos	El usuario deberá dejar los campos vacíos	V	N/A	El sistema muestra un mensaje el mensaje “Debe especificar el nombre del concepto.”	El usuario hace clic derecho encima del concepto creado y da en la opción Editar Concepto. Ya dentro modifica los campos que desee. S desea añadir el concepto acciona la opción Aceptar, sino Cancelar. Si acciona Aceptar, el sistema muestra un mensaje el mensaje “Debe especificar el nombre del concepto.”
		()	Admisión		
EC 1.4 Editar concepto sin propiedades	El usuario deberá modificar un concepto sin añadir ninguna	V	N/A	El sistema muestra un mensaje el mensaje “El concepto debe	El usuario hace clic derecho encima del concepto creado y da en la opción Editar Concepto. Ya dentro

	propiedad.	Paciente	Admisión	tener propiedades.”	modifica los campos que desee. Si desea añadir el concepto acciona la opción Aceptar, sino Cancelar. Si acciona Aceptar, el sistema muestra un mensaje el mensaje “El concepto debe tener propiedades.”
EC 1.5 Editar concepto con ninguna propiedad con búsqueda simple	El usuario deberá tener el campo búsqueda simple de todas las propiedades sin marcar.	N/A	N/A	El sistema muestra un mensaje el mensaje “Debe marcar al menos una propiedad para búsqueda simple.”	El usuario hace clic derecho encima del concepto creado y da en la opción Editar. Ya dentro modifica los campos que desee. Si desea añadir el concepto acciona la opción Aceptar, sino Cancelar. Si acciona Aceptar, el sistema muestra un mensaje el mensaje “Debe marcar al menos una propiedad para búsqueda simple.”
		()	Admisión		
EC 1.6 Editar concepto con esquema incorrecto.	El usuario deberá modificar el nombre del esquema empleando un nombre distinto a los ya definidos.	N/A	N/A	El sistema muestra un mensaje el mensaje “El esquema no es válido.”	El usuario hace clic derecho encima del concepto creado y da en la opción Editar. Ya dentro modifica los campos que desee. Si desea añadir el concepto acciona la opción Aceptar, sino Cancelar. Si acciona Aceptar, el sistema muestra un mensaje el mensaje “El esquema no es válido.”
		Paciente	Admasd/65		

EC 1.7 Editar concepto cargado.	El usuario deberá tratar de editar un concepto cargado.	N/A	N/A	El sistema muestra un mensaje el mensaje “Los conceptos cargados no pueden ser editados”	El usuario hace clic derecho encima del concepto cargado y da en la opción Editar. El sistema muestra un mensaje el mensaje “Los conceptos cargados no pueden ser editados”
		()	Admasd/65		

SC <Eliminar concepto>			
Escenario	**Descripción**	**Respuesta del sistema**	**Flujo central**
EC 1.1 Eliminar concepto creado	El usuario deberá eliminar un concepto creado.	El sistema muestra una ventana de advertencia: ¨Está seguro que desea eliminar este concepto¨	El usuario hace clic derecho encima del concepto creado y da en la opción Eliminar. El sistema muestra una ventana de advertencia: ¨Está seguro que desea eliminar este concepto¨
EC 1.2 Eliminar concepto cargado	El usuario deberá eliminar un concepto cargado.	El sistema muestra una ventana de advertencia: ¨Los conceptos cargados no pueden ser eliminados¨	El usuario hace clic derecho encima del concepto cargado y da en la opción Eliminar. El sistema muestra una ventana de advertencia: ¨Los conceptos cargados no pueden ser eliminados¨

Tabla 7 Caso de prueba “Añadir concepto”

Cada una de estas variables son descritas en la siguiente tabla, donde se enumeran cada uno de los campos que se definieron en los escenarios de pruebas, especificando su nombre, clasificación según el componente de diseño utilizado, valor nulo y una breve descripción de las reglas que tiene que cumplir el campo.

No	Nombre de campo	Clasificación	Valor Nulo	Descripción

V1	Nombre	Campo de entrada (jTextField)	NO	Los datos que se introduzcan deben ser letras, sin dejar espacios entre las letras.
V2	Esquema	Campo de selección (jComboBox)	NO	Se escoge el tipo de esquema que desea aparezca en la BD.

Tabla 8 *Descripción de variables de "CP Añadir concepto"*

Resultados de las pruebas

Las pruebas de caja negra fueron realizadas de manera estricta a partir de los casos de prueba elaborados. Realizar las pruebas de caja negra permitió detectar, documentar y solucionar los errores existentes en el sistema implementado. Para llevar a cabo estas pruebas se utilizó la técnica partición de equivalencia que permitió realizar los diseños de casos de prueba. Se validó que la herramienta creada satisface los requisitos identificados. También se identificaron deficiencias en la funcionalidad relacionar conceptos y nomencladores ya que solo permite efectuar relaciones uno a muchos y muchos a muchos para la mayoría de los casos. Esta funcionalidad no permite establecer la relación recursiva y la de uno a uno.

El período de pruebas permitió obtener una aplicación que se corresponde con los objetivos planteados para la arquitectura del sistema alas HIS. Esta genera el código repetitivo identificado, posibilitando prescindir de su codificación manual logrando un ahorro de tiempo considerable.

Conclusiones del capítulo

En el presente capítulo se diseñaron casos de pruebas basados en las HU que se obtuvieron en la fase de exploración de la aplicación. Se realizaron las pruebas de aceptación utilizando el método de las pruebas de caja negra mediante la técnica de partición equivalente, permitiendo detectar, documentar y solucionar los errores existentes en el sistema implementado.

Conclusiones

Investigar los principales generadores de código utilizados en la actualidad permitió identificar ventajas e incorporar buenas prácticas asociadas a sus características.

El análisis realizado de la arquitectura alas HIS permitió identificar en el código fuente patrones, buenas prácticas y reutilización, útiles para definir las plantillas como tipo de generación de código seleccionado.

La utilización del tipo de generador de código basado en plantillas garantiza que el código generado cumpla las pautas de interfaz de usuario y el estándar de codificación definido en el desarrollo del sistema alas HIS.

La metodología XP contribuyó a la obtención de una herramienta que satisface las exigencias del cliente de acuerdo a los artefactos que lo involucran.

El proceso de pruebas realizado a la herramienta arrojó la existencia de errores y permitió su corrección, mejorando la calidad de la aplicación.

Recomendaciones

Luego de desarrollado el generador de código para el sistema alas HIS se identificaron elementos que se recomienda incluir en futuras versiones:

- Agregar la capacidad de validar los campos de entrada de datos para los prototipos funcionales generados, para que solo permita la entrada de caracteres que el usuario desee.
- Permitir que el usuario pueda editar las plantillas desde la propia aplicación.

Bibliografía

1. **Herrington, Jack.** *Code Generation in Action.* s.l. : Manning, 2006.

2. Automated Code Generation. [En línea] [Citado el: 4 de 2 de 2013.] http://c2.com/cgi/wiki?AutomatedCodeGeneration.

3. **Balanta, Vieravictor.** Emagister. [En línea] [Citado el: 4 de 2 de 2013.] http://www.emagister.com/web/cursogratis/frame?idCentro=585710700521515767525568696545 66&idCurso=504653960547481421752004252308 39.

4. **Moreno, P.J.M.** *Especificación de interfaz de usuario: De los requisitos a la generación de código.* Valencia, Universidad de Valencia : s.n., 2003.

5. **Domínguez, Orestes.** Seam City. [En línea] 11 de 2 de 2008. http://seamcity.madeinxpain.com/archives/seam-generator-seam-gen.

6. **YesSoftware Inc.** YesSoftware. [En línea] [Citado el: 1 de 2 de 2013.] http://www.yessoftware.com/products/product.php?product_id=1.

7. **CodeSmith Tools, LLC.** CodeSmith Tools. [En línea] [Citado el: 1 de 2 de 2013.] http://www.codesmithtools.com/product/generator.

8. **Artech Consultores SRL.** GeneXus. [En línea] [Citado el: 4 de 2 de 2013.] http://www.genexus.com/.

9. **Oracle Academy.** Java y Tú. [En línea] ORACLE. [Citado el: 4 de 2 de 2013.] http://www.java.com/es/download/whatis_java.jsp.

10. **Object Management Group, Inc.** UML® Resource Page. [En línea] [Citado el: 11 de 2 de 2013.] http://www.omg.org/gettingstarted/what_is_uml.htm.

11. **Oracle Corporation.** NetBeans. [En línea] [Citado el: 4 de 2 de 2013.] http://netbeans.org/index_es.html.

12. **Fifesoft.** Fifesoft. [En línea] 2012. http://fifesoft.com/rsyntaxtextarea/.

13. **Alarcos.** Metodologías de Desarrollo de software. [En línea] [Citado el: 14 de 4 de 2013.] http://alarcos.inf-cr.uclm.es/doc/ISOFTWAREI/Tema04.pdf.

14. **BAUTISTA, JOSE M.** *Programación Extrema.* s.l. : UNIVERSIDAD UNION BOLIVARIANA.

15. **Fowler, M.** La nueva metodología. *Programación Extrema.* [En línea] 2003. [Citado el: 2 de 3 de 2013.] http://www.programacionextrema.org/articulos/newMethodology.es.html.

16. **Hernández, Pedro Veloso.** *Uso de patrones de arquitectura.*

17. **Joskowicz, Ing. José.** Tareas (Online). [En línea] 10 de 2 de 2008. http://iie.fing.edu.uy/~josej/docs/XP%20-%20Jose%20Joskowicz.pdf..

18. **Zapata, Mª Antonia.** [En línea] 2006. http://ocw.unizar.es/ciencias-experimentales/modelos-matematicos-en-bases-de-datos/uml/03UML_DiagramaClases.pdf..

19. **clases, Diagrama de.** [En línea] http://docs.kde.org/stable/es/kdesdk/umbrello/uml-elements.html..

20. **Tripod.com.** Fases de la Programación Extrema. [En línea] http://programacionextrema.tripod.com/fases.htm#segundaFase..

21. **Guglielmetti, Marcos.** Arquitectura. [En línea] http://www.mastermagazine.info/termino/3916.php.

22. **Cesar de la Torre Llorente, y otros.** *Guía de Arquitectura N-Capas orientada al dominio con .* 2010. 978-84-936696-8.

23. **Eduardo Fernández, Medina Patón.** Alarcos. [En línea] http://alarcos.inf-cr.uclm.es/doc/ISOFTWAREI/Tema09.pdf ..

24. **Aponte, Erick Salazar y Anaís.** Patrones. [En línea] http://ldc.usb.ve/~teruel/ci3711/patron3a/index.html#experto..

25. Manuales.com. [En línea] 3 de 4 de 2013. http://www.manuales.com/manual-de/que-es-el-camelcase.

26. **Visconti, Marcello Astudillo y Hernán.** *Fundamentos de Ingeniería de Software.* s.l. : Universidad Técnica Federico Santa María.

27. **J.J Gutíerrez, M.J Escalona ,M.Mejías ,J.Torres.** *Pruebas del sistema en Programación Extrema.* s.l. : Sevilla : s.n.

28. **Malforá, Dayvis, y otros.** *Testing en eXtreme Programing.* 2006.

29. **Pablo Suearez, Carlos Fontela.** *Documentación y pruebas ante el paradigma de objetos.* 2003.

30. **José M .Drake, Patricia López.** *Verificación y Validación.* 2009.

31. **Juristo, Natalia, M.Moreno, Ana y Vegas, Sira.** *Técnicas de evaluación de software.* 2006.

32. **Johanna Rojas, Emilio Barrios.** [En línea] 13 de 4 de 2011. http://gemini.udistrital.edu.co/comunidad/grupos/arquisoft/fileadmin/Estudiantes/Pruebas/HTML%20-%20Pruebas%20de%20software/node28.html..

33. **Espinoza, Humberto.** PostgreSQL, Una Alternativa de DBMS Open Source. [En línea] 2005. http://www.lgs.com.ve/pres/PresentacionES_PSQL.pdf.

34. **Pérez, Beatriz.** *ProTest-Proceso de Testing Funcional.*

35. **Mauricio.** Updateando. [En línea] 2013. http://updateando.blogspot.com/2012/04/seam-framework.html.

36. **Oswaldo Castillo, Daniel Figueroa , Hector Sevilla.** TRIPOD. [En línea] http://programacionextrema.tripod.com/index.htm..

37. **Michael Juntao Yuan, Jacob Orshalick & Thomas Heute.** *Seam Framework.*

38. **Domínguez, Orestes.** Seam City. [En línea] 11 de 2 de 2008. http://seamcity.madeinxpain.com/archives/seam-generator-seam-gen.

39. **Monografias.com S.A.** monografias.com. [En línea] http://www.monografias.com/trabajos57/programacion-web-avanzada/programacion-web-avanzada.shtml.

40. **KING, CHRISTIAN BAUER & GAVIN.** *Java Persistence with Hibernate.*

41. **Artech Consultores SRL.** GeneXus. [En línea] [Citado el: 4 de 2 de 2013.] http://www.genexus.com/.

42. **Rojas, María del Mar Aguilera Sierra y Sergio Gálvez.** Generación de código intermedio. [En línea] http://www.lcc.uma.es/~galvez/ftp/tci/tictema7.pdf.

43. **Freelance Soft.** FreelanceSoft. *FreelanceSoft.* [En línea] 2013. http://www.freelance-soft.com/productos/framework-de-generacion-de-codigo/.

44. **Fifesoft.** Fifesoft. [En línea] 2012. http://fifesoft.com/rsyntaxtextarea/.

45. **Tripod.com.** Fases de la Programación Extrema. [En línea] http://programacionextrema.tripod.com/fases.htm#segundaFase..

46. **CodeSmith Tools, LLC.** CodeSmith Tools. [En línea] [Citado el: 1 de 2 de 2013.] http://www.codesmithtools.com/product/generator.

47. **Mastermagazine.** Arquitectura. [En línea] http://www.mastermagazine.info/termino/3916.php.

Referencias bibliográficas

1. **Herrington, Jack.** *Code Generation in Action.* s.l. : Manning, 2006.

2. Automated Code Generation. [Online] [Cited: 2 4, 2013.] http://c2.com/cgi/wiki?AutomatedCodeGeneration.

3. **Balanta, Vieravictor.** Emagister. [Online] [Cited: 2 4, 2013.] http://www.emagister.com/web/cursogratis/frame?idCentro=585710700521515767525568696545 66&idCurso=504653960547481421752004252308 39.

4. **Moreno, P.J.M.** *Especificación de interfaz de usuario: De los requisitos a la generación de código.* Valencia, Universidad de Valencia : s.n., 2003.

5. **Domínguez, Orestes.** Seam City. [Online] 2 11, 2008. http://seamcity.madeinxpain.com/archives/seam-generator-seam-gen.

6. **YesSoftware Inc.** YesSoftware. [Online] [Cited: 2 1, 2013.] http://www.yessoftware.com/products/product.php?product_id=1.

7. **CodeSmith Tools, LLC.** CodeSmith Tools. [Online] [Cited: 2 1, 2013.] http://www.codesmithtools.com/product/generator.

8. **Artech Consultores SRL.** GeneXus. [Online] [Cited: 2 4, 2013.] http://www.genexus.com/.

9. **Oracle Academy.** Java y Tú. [Online] ORACLE. [Cited: 2 4, 2013.] http://www.java.com/es/download/whatis_java.jsp.

10. **Object Management Group, Inc.** UML® Resource Page. [Online] [Cited: 2 11, 2013.] http://www.omg.org/gettingstarted/what_is_uml.htm.

11. **Oracle Corporation.** NetBeans. [Online] [Cited: 2 4, 2013.] http://netbeans.org/index_es.html.

12. **Fifesoft.** Fifesoft. [Online] 2012. http://fifesoft.com/rsyntaxtextarea/.

13. **Alarcos.** Metodologías de Desarrollo de software. [Online] [Cited: 4 14, 2013.] http://alarcos.inf-cr.uclm.es/doc/ISOFTWAREI/Tema04.pdf.

14. **BAUTISTA, JOSE M.** *Programación Extrema.* s.l. : UNIVERSIDAD UNION BOLIVARIANA.

15. **Fowler, M.** La nueva metodología. *Programación Extrema.* [Online] 2003. [Cited: 3 2, 2013.] http://www.programacionextrema.org/articulos/newMethodology.es.html.

16. **Hernández, Pedro Veloso.** *Uso de patrones de arquitectura.*

17. **Joskowicz, Ing. José.** Tareas (Online). [Online] 2 10, 2008. http://iie.fing.edu.uy/~josej/docs/XP%20-%20Jose%20Joskowicz.pdf..

18. **Zapata, Mª Antonia.** [Online] 2006. http://ocw.unizar.es/ciencias-experimentales/modelos-matematicos-en-bases-de-datos/uml/03UML_DiagramaClases.pdf..

19. **clases, Diagrama de.** [Online] http://docs.kde.org/stable/es/kdesdk/umbrello/uml-elements.html..

20. **Tripod.com.** Fases de la Programación Extrema. [Online] http://programacionextrema.tripod.com/fases.htm#segundaFase..

21. **Guglielmetti, Marcos.** Arquitectura. [Online] http://www.mastermagazine.info/termino/3916.php.

22. **Cesar de la Torre Llorente, y otros.** *Guía de Arquitectura N-Capas orientada al dominio con .* 2010. 978-84-936696-8.

23. **Eduardo Fernández, Medina Patón.** Alarcos. [Online] http://alarcos.inf-cr.uclm.es/doc/ISOFTWAREI/Tema09.pdf ..

24. **Aponte, Erick Salazar y Anaís.** Patrones. [Online] http://ldc.usb.ve/~teruel/ci3711/patron3a/index.html#experto..

25. Manuales.com. [Online] 4 3, 2013. http://www.manuales.com/manual-de/que-es-el-camelcase.

26. **Visconti, Marcello Astudillo y Hernán.** *Fundamentos de Ingeniería de Software.* s.l. : Universidad Técnica Federico Santa María.

27. **J.J Gutíerrez, M.J Escalona ,M.Mejías ,J.Torres.** *Pruebas del sistema en Programación Extrema.* s.l. : Sevilla : s.n.

28. **Malforá, Dayvis, y otros.** *Testing en eXtreme Programing.* 2006.

29. **Pablo Suearez, Carlos Fontela.** *Documentación y pruebas ante el paradigma de objetos.* 2003.

30. **José M .Drake, Patricia López.** *Verificación y Validación.* 2009.

31. **Juristo, Natalia, M.Moreno, Ana y Vegas, Sira.** *Técnicas de evaluación de software.* 2006.

32. **Johanna Rojas, Emilio Barrios.** [Online] 4 13, 2011. http://gemini.udistrital.edu.co/comunidad/grupos/arquisoft/fileadmin/Estudiantes/Pruebas/HTML%20-%20Pruebas%20de%20software/node28.html..

33. **Espinoza, Humberto.** PostgreSQL, Una Alternativa de DBMS Open Source. [En línea] 2005. http://www.lgs.com.ve/pres/PresentacionES_PSQL.pdf.

34. **Pérez, Beatriz.** *ProTest-Proceso de Testing Funcional.*

Glosario de términos

A

API: Del inglés *Application Programming Interface* (Interfaz de Programación de Aplicaciones) es del conjunto de funciones y procedimientos (o métodos si se refiere a programación orientada a objetos) que ofrece cierta librería para ser utilizado por otro software como una capa de abstracción.

Arquitectura: Consiste en un conjunto de patrones y abstracciones coherentes que proporcionan el marco de referencia necesario para guiar la construcción de software para un sistema informático. La arquitectura de software establece los fundamentos para que analistas, diseñadores y programadores trabajen en una línea común que permita alcanzar los objetivos y necesidades del sistema informático.

ASCII: (acrónimo inglés de ***A**merican **S**tandard **C**ode for **I**nformation **I**nterchange — Código Estándar Estadounidense para el Intercambio de Información*), es un código de caracteres basado en el alfabeto latino, tal como se usa en inglés moderno y en otras lenguas occidentales.

C

Clase: Declaración o abstracción de un objeto cuando se programa usando el paradigma de orientación a objetos.

Código: Cifras, clave. En informática se utiliza para referirse a un conjunto de instrucciones en un lenguaje de programación.

CRUD: En computación CRUD es el acrónimo de Crear, Obtener, Actualizar y Borrar (del original en inglés: ***C**reate, **R**ead, **U**pdate and **D**elete*). Es usado para referirse a las funciones básicas en bases de datos o la capa de persistencia en un software.

F

***Facelets*:** Es un lenguaje de declaración de páginas, poderoso pero ligero, que es usado para construir vistas de *Java Server Faces* usando plantillas de estilo de XHTML y construyendo arboles de componentes.

***Framework*s:** Estructura predefinida para la creación de aplicaciones. Puede estar formado por un conjunto de librerías y clases o por una arquitectura que facilita el desarrollo de software.

I

IDE: Ambiente integrado de desarrollo (*Integrated development environment*). Es un conjunto de software que permite el desarrollo de aplicaciones.

J

Java Server Faces (**JSF**): Es una tecnología y *framework* para aplicaciones Java basadas en web que simplifica el desarrollo de interfaces de usuario en aplicaciones Java EE.

JEE: *Java Platform, Enterprise Edition* o Java, es una plataforma de programación (parte de la Plataforma Java) para desarrollar y ejecutar software de aplicaciones en el lenguaje de programación Java. Permite utilizar arquitecturas de N capas distribuidas y se apoya ampliamente en componentes de software modulares ejecutándose sobre un servidor de aplicaciones.

M

Metodología de desarrollo: En ingeniería de software es un marco de trabajo usado para estructurar, planificar y controlar el proceso de desarrollo en sistemas de información.

Modelo: Representación abstracta de la realidad. Diagramas que representan la estructura de un sistema dado.

Multiplataforma: Es un término usado para referirse a los programas, sistemas operativos o lenguajes de programación que puedan funcionar en diversas plataformas como Windows o Linux.

P

Patrón: Es una solución a un problema no trivial que es efectiva (ya se resolvió el problema satisfactoriamente en ocasiones anteriores) y reusable (se puede aplicar a diferentes problemas en distintas circunstancias).

***Properties*:** Es una extensión de archivo mayormente utilizada en tecnologías java relacionadas al almacenamiento de parámetros configurables de una aplicación. También pueden ser utilizados para almacenar cadenas de internacionalización y localización, los cuales son conocidos como paquetes de recursos de propiedad.

S

Software: Término genérico que designa al conjunto de programas que posibilitan realizar una tarea específica en un ordenador.

U

UML: Lenguaje Unificado de Modelado (*Unified Modeling Language*). Es un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema de software.

X

XHTML: Siglas del inglés ***eXtensible HypeRText Markup Language***. XHTML es básicamente HTML expresado como XML válido. Es más estricto a nivel técnico, pero esto permite que posteriormente sea más fácil al hacer cambios o buscar errores entre otros.

XML: Sigla en inglés de ***eXtensible Markup Language*** («lenguaje de marcas extensible»), es un metalenguaje extensible de etiquetas desarrollado por el *World Wide Web Consortium* (W3C).