Universidad de las Ciencias Informáticas

Trabjo de diploma para optar por el título de Ingeniero en Ciencias Informáticas.

Título: Desarrollo de funcionalidades para la gestión de reglas de derivación y validación de los campos de las hojas CRD en el sistema SIGEC.

Autores: Katerine Mendoza del Toro
Felino Milquides Tito Quintana

Tutor: Ing.Erislán Martinez Jera

Cotutora: Ing. Yamilet Ugarte Céspedes

La Habana, 2014
"Año 56 de la Revolución"

Un gran descubrimiento resuelve un gran problema, pero hay una pizca de descubrimiento en la solución de cualquier problema. Tu problema puede ser modesto, pero si es un reto a tu curiosidad y trae a juego tus facultades inventivas, y si lo resuelves por tus propios métodos, puedes experimentar la tensión y disfrutar del triunfo del descubrimiento.

George Pólya

Declaración de autoría

Declaramos que somos los únicos autores de este trabajo y autorizamos a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

Para que así conste firmamos la presente a los __ días del mes de ___ del año ____.

_____ _____

Katerine Mendoza del Toro — Firma del autor

Felino Milquides Tito Quintana — Firma del autor

_____ _____

Ing. Erislán Martinez Jera — Firma del tutor

Ing.Yamilet Ugarte Céspedes — Firma de Cotutora

Datos de contacto

Ing. Yamilet Ugarte Céspedes

Ing. Yamilet Ugarte Céspedes, graduada de Ingeniera en Ciencias Informáticas en el año 2012 en la Universidad de las Ciencias Informáticas. Recién Graduado en Adiestramiento. Se desempeña como Analista Principal del Sistema de Gestión de Ensayos Clínicos del Departamento Sistemas Especializados en Salud (SES), del Centro de Informática Médica (CESIM). Correo electrónico: yugarte@uci.cu

Ing. Erislán Martinez Jera

Ing. Erislán Martinez Jera, profesor instructor con 4 años de experiencia laboral en la Universidad de las Ciencias Informáticas. Ha impartido las asignaturas: Matemática, Programación, Práctica Profesional y algunos cursos optativos. Pertenece al Dpto. Sistemas de Especializados en Salud donde se desempeña como jefe del proyecto Ensayos Clínicos.

Correo electrónico: ejera@uci.cu

Agradecimientos de Katerine

Quiero agradecer a los profesores que han estado conmigo estos años y me han ayudado a llegar hasta este día, es especial a Luis Mariano, Pedro y Ronal que son grandes profesores. A Zoila y Magui que confiaron en mí momentos difíciles.

Gracias a todos…

A personas con las que eh tenido el privilegio de compartir buenos momentos y malos: Pochet, Asiel, Yanet, Daniellis, Alexis, Hansel, Nelson, Andrés, Rey, JJ, Yunior, Daymer, Gloria, Israel, Daniel, Juan Miguel, Luis Miguel, Julio, Eiler, los Jorges, Gleidis, Miriennis.

A mis tutores que más que eso han sido mis amigos, han estado conmigo en todo momento ayudándome, apoyándome y soportándome mis malcriadeces.

Muchas gracias…

A mi querido compañero de tesis por su paciencia con mis alteraciones y hacerme sonreír en los momentos de estrés muchísimas gracias.

A personas especiales que tuve la oportunidad de conocer en el transcurso de estos años que han estado siempre apoyándome: María Julia, Chuchi, Dayana, Heily, Yamy, la Flaca, Yai, el Fisch, Diani, Bety, Balbinis, Claudia, Joge Enrrique, Yudy, Arian y muy especialmente a tres personas que quiero mucho y siempre han estado conmigo en momentos buenos y malos Azuan, la chiqui y Monika.

Mi amiga de siempre Dianelis que aunque nos hemos pasado varios años distante no ha dejado de animarme, aconsejarme y alentarme en todo momento a luchar y conseguir mí meta.

A mis suegros Yoel y Marlene que son personas que estimo y aprecio queme acogieron en su familia y han sido como mis segundos padres, dándome apoyo.

Mi novio que más que mi novio es mi más grande amigo me ha guiado y ayudado en todos estos años, parte de mi camino hasta aquí se lo debo a él, muchas gracias por estar a mi lado te quiero mucho.

Mi familia que me han apoyado y escuchado mis berrinches, Yudy, Elin, Tafen, Abuelo, mi papá Toni, a mis tío Alain, Amel que ha sido mi hermanito del alma y mi primito que lo quiero mucho Rainer, Rafael, Yudy y Elin.

Agradecer muy en especial a mi mamá, mi abuela y mi niño que me hayan dado fuerzas para seguir adelante en los momentos en que cruzó por mi cabeza la idea de rendirme, gracias por ser mi ejemplo a seguir en todo momento, espero que mi chichí se sienta orgulloso de mi, muchas gracias los quiero mucho.

Agradecimientos de Felino

Dedicatoria de Felino y Katerine

Dedicamos este trabajo a nuestros padres queridos y a todas las personas especiales que han estado ahí apoyándonos.

Resumen

En el Centro de Inmunología Molecular (CIM) que se encuentra en la Habana, Cuba, se realizan investigaciones para tratar y prevenir varias enfermedades, para esto se realizan ensayos clínicos con el fin de probar la eficacia y seguridad de los nuevos medicamentos creados. La información obtenida de estas investigaciones se recoge en el sistema Clínicas 1.0 que se encuentra desplegado en dicho centro, consta de cuatro módulos, uno de ellos es, Gestionar Estudios. Este cuenta con varias funcionalidades dentro de las cuales se encuentra la Gestión de las reglas, que pueden ser de validación o derivación. Estas reglas presentan algunos inconvenientes. Por lo que se desarrolla una segunda versión del sistema llamada Sistema de Gestión de Ensayos Clínicos (SIGEC).

Para el diseño de esta aplicación se usó la metodología de software Proceso Unificado de Desarrollo, la cual se apoyó en el Lenguaje de Modelado Unificado y fue modela con Visual Paradigm. Fue usado el entorno de desarrollo Eclipse, unido a Java como lenguaje de programación, PostgreSQL como sistema gestor de base de datos. Junto a estas herramientas se incorporaron frameworks y librerías. Todos los elementos anteriores giraron alrededor del patrón Modelo-Vista-Controlador. Luego de concluir el desarrollo se obtuvo como resultado un sistema funcional y con atributos de seguridad. Esta aplicación brindará servicios con mayor calidad.

Palabras clave: ensayos clínicos, reglas, validación, derivación, Sistema de Gestión de Ensayos Clínicos.

TABLA DE CONTENIDOS

Tabla de contenido

Introducción 1
Capítulo 1: Fundamentación teórica 5
1.1. Conceptos asociados a Ensayos Clínicos 5
1.2. Centros donde se realizan EC 5
1.3. Clínicas 6
1.4. Estudio de los sistemas existentes 6
1.5. Metodologías, tecnologías y herramientas a considerar 8
1.6. Tecnologías 9
1.7. Herramientas 12
Capítulo 2: Características del sistema 14
2.1 Propuesta de solución 14
2.2. Modelo de dominio 14
2.3. Especificación de los requisitos de software 15
2.4 Modelo de casos de uso del sistema 18
Capítulo 3: Diseño del sistema 25
3.1. Descripción de la arquitectura 25
3.2. Diagramas de clases del diseño 28
Capítulo 4: Implementación 40
4.1. Modelo de datos 40
4.2. Modelo de implementación 47
4.3. Tratamiento de errores 49
4.4. Seguridad 50
4.5. Estrategias de codificación. Estándares y estilos a utilizar 51
Conclusiones 53
Recomendaciones 54
Referencias Bibliográficas 55
Bibliografía 57
Anexos 60
Anexo 1: 60
Anexo 2: 61
Glosario de términos 65

Introducción

El crecimiento de las Tecnologías de la Información y las Comunicaciones (TIC) ha propiciado un aumento en la industria del software que continúa alcanzando logros en diferentes sectores de la sociedad. Uno de los sectores más beneficiados es la salud, para el cual se han desarrollado aplicaciones informáticas en aras de mejorar la calidad de los servicios. Algunos de estos sistemas por ejemplo se encuentran en centros de investigaciones científicas, los cuales facilitan el trabajo de los profesionales de esta rama. Varios de estos centros de investigaciones científicas se dedican a la creación de fármacos para tratar y/o prevenir enfermedades. Para recoger los datos de estas investigaciones en algunos países del mundo se trabaja en sistemas que gestionan la recogida de datos de los Ensayos Clínicos (EC).

Los ensayos clínicos son estudios de investigación que prueban el funcionamiento de los nuevos enfoques clínicos en las personas. Cada estudio responde preguntas científicas e intenta encontrar mejores formas de prevenir, explorar, diagnosticar o tratar una enfermedad. Los ensayos clínicos también pueden comparar un tratamiento nuevo con uno que se encuentra disponible(1).

En el mundo existen varios centros que realizan EC para la creación de fármacos, estos centros no sólo se encuentran en los países con un alto nivel de desarrollo. Cuba también cuenta con sus propios centros de investigación, uno de ellos es el Centro de Inmunología Molecular (CIM), donde se encuentra desplegado el sistema Clínicas en su versión 1.0. Este sistema fue desarrollado en la Universidad de las Ciencias Informáticas (UCI), en el Centro de Informática Médica (CESIM). El mismo se utiliza para la gestión de los ensayos clínicos que se desarrollan en dicho centro.

Clínicas consta de cuatro módulos: Decir lo que significa cada módulo breve una oración(“Administrar Sistema”, “Gestionar Estudios”, “Enviar Datos” y “Extraer Datos”). El módulo “Gestionar estudios” del sistema cuenta con varias funcionalidades entre las cuales se encuentra la gestión de las reglas, que luego van a ser ejecutadas en el módulo “Enviar datos”.

Existen dos tipos de reglas, las reglas de derivación, mediante las cuales el sistema será capaz de asignar un valor a una variable evitando que el usuario introduzca datos incorrectos y las reglas de validación, mediante las que se comprueban que los datos de los campos de las hojas de los Cuadernos de Recogida de Datos (CRD) son correctos. Estas hojas CRD son formularios dinámicos lo cual complica aún más su validación y deben llenarse según una planificación definida en el cronograma del estudio, y a medida que se van llenando se ejecutan las reglas garantizando que los datos se guarden de manera correcta.

Las reglas de validación presentan varios inconvenientes por ejemplo las que se aplican a las variables de tipo fecha no permiten realizar operaciones aritméticas como suma y resta, sólo permite lógicas y relacionales por lo que si el usuario requiere realizar este tipo de operaciones debe hacerlo manualmente, lo cual puede ocasionar la entrada de datos erróneos al sistema, provocando que el estudio arroje resultados falsos.

Por su parte las reglas de derivación presentan el inconveniente de que para registrar los datos el usuario debe indicar guardar dos veces a la hoja CRD, la primera vez, para ejecutar las reglas establecidas y la segunda para guardar los datos. Estos inconvenientes evidencian que el sistema no cuenta con un ambiente sencillo y no es fácil de manejar para todos los usuarios, afectando la usabilidad del mismo.

Otro inconveniente que presenta la gestión de las reglas es que, una vez que el diseño del estudio es aprobado por el investigador promotor y se inicia la conducción de los ensayos, no es posible agregar nuevas reglas o modificar las existentes, por tanto el especialista debería rediseñar el estudio desde cero e introducir los datos nuevamente si quiere realizar estos cambios trayendo como consecuencia pérdidas de tiempo y atraso del EC.

Por lo anteriormente planteado se identifica el siguiente **problema a resolver,** ¿Cómo mejorar la gestión y ejecución de las reglas de validación y derivación de los campos de las hojas del Cuaderno de Recogida de Datos del sistema Clínicas 1.0?

Con vista a dar solución al problema planteado se define como **objeto de estudio** validación de los datos de formularios dinámicos, enmarcado en el **campo de acción** proceso validación de los datos de formularios dinámicos en sistemas de gestión de ensayos clínicos, siendo así el **objetivo general** desarrollar, en el Sistema de Gestión de Ensayos Clínicos, funcionalidades para la gestión y ejecución de reglas de derivación y validación de los campos de las hojas del Cuaderno de Recogida de Datos**.** Para dar cumplimiento al objetivo general, se plantean las siguientes **tareas de la investigación:**

1. Analizar el estado del arte de los sistemas que realizan la validación de formularios dinámicos a nivel internacional y nacional estableciendo similitudes con la investigación en curso.
2. Asimilar las herramientas, tecnologías y metodología, propuestas en el Centro de Informática Médica (CESIM) para el desarrollo de la solución.
3. Elaborar los artefactos correspondientes a los flujos de trabajo, Modelado de negocio, Requisitos, Análisis y Diseño e Implementación, propuestos por la metodología de desarrollo Proceso Unificado de Desarrollo de Software (RUP).

4. Implementar las funcionalidades para gestionar las reglas de validación y derivación de formularios dinámicos en el Módulo Diseño del Sistema SIGEC, aplicando las pautas de diseño definidas en el Centro de Informática Médica (CESIM) y a partir de la especificación de requisitos.

Para dar cumplimento a las tareas propuestas fueron utilizados los siguientes métodos de investigación:

Métodos teóricos

- ✓ Analítico-Sintético: el uso de este método permite hacer un análisis del problema de investigación.
- ✓ Histórico-Lógico: este método se emplea para conocer información acerca de los sistemas que realizan validación de formularios dinámicos.
- ✓ Inductivo-Deductivo: es utilizado luego de haber consultado la bibliografía, para obtener los elementos a emplear en la investigación.
- ✓ Modelación: este se aplicará en la representación, mediante diagramas, del proceso de gestión de las reglas.

Métodos empíricos

- ✓ Análisis documental: se usa para el análisis de la bibliografía consultada.
- ✓ Entrevista: se usa para conocer cómo se realiza el proceso de validación en algunos sistemas y varias de sus características.
- ✓ Observación: se emplea para obtener información sobre el objeto de la investigación en el sistema Clínicas 1.0.

El desarrollo de la solución propuesta permite obtener varios beneficios:

- ✓ Permite una validación más óptima de los datos recogidos de los pacientes, al contar con reglas más complejas que Clínicas 1.0, esto disminuye los errores en los datos clínicos mejorando así la gestión de los estudios.
- ✓ Reduce el tiempo de desarrollo de los EC y evita el atraso de los mismos brindando la posibilidad de modificar y agregar reglas después de iniciada la conducción del ensayo.

El documento se encuentra dividido en cuatro capítulos, estructurados de la siguiente manera:

Capítulo 1: Fundamentación Teórica: Se presentan los principales conceptos para el desarrollo del sistema. Se realiza un estudio de varios sistemas que realizan validación de formularios dinámicos. Se fundamentan las tecnologías, metodologías y herramientas de desarrollo utilizadas.

Capítulo 2: Características del sistema: Se describen las principales características del sistema a través del Modelo de Dominio, conjuntamente con la especificación de los requisitos funcionales y no funcionales y el Modelo de Casos de Usos del Sistema.

Capítulo 3: Diseño del sistema: En este capítulo se describe todo el flujo de trabajo del diseño del sistema, realizando el modelo de diseño y definiendo la arquitectura y patrones a utilizar.

Capítulo 4: Implementación: Se presenta el flujo de trabajo de implementación, partiendo de los resultados obtenidos en el diseño. Se detalla el Modelo de datos, donde se almacena toda la información requerida en el sistema y se exponen aspectos referentes a la seguridad del mismo, así como la forma en que se tratarán los errores.

Capítulo 1: Fundamentación teórica

En este capítulo se relacionan los conceptos asociados a la investigación y se realiza un análisis de sistemas existentes a nivel internacional y nacional que gestionan la validación de formularios dinámicos. Además se describen las características de la metodología, tecnologías y herramientas utilizadas para el desarrollo de las funcionalidades.

1.1. Conceptos asociados a Ensayos Clínicos

Los EC son estudios de investigaciones que se realizan a grupos de pacientes que presentan una misma enfermedad. Su objetivo es valorar la eficacia y seguridad de nuevos fármacos o tratamientos a través de su aplicación a seres humanos. Estos estudios son realizados en centros de salud e investigaciones.

Uno de los elementos esenciales para la ejecución de un EC es la definición del cronograma general, el cual no es más que una lista de detalles que deben ser tomados en cuenta para dar cumplimiento a las actividades necesarias. Dicho cronograma define el tiempo durante el cual los pacientes serán tratados o evaluados en el estudio y las diferentes etapas por las cuales deberá transitar.

Como parte de la definición de un cronograma general se encuentran los momentos de seguimiento, que son los distintos períodos en los cuales se aplicará el tratamiento a los pacientes o se realizará seguimiento durante el desarrollo de un EC, lo que permitirá evaluar la reacción del paciente ante el tratamiento orientado.

Otro aspecto dentro de un cronograma general son las hojas CRD, en estas se recogen los datos de los pacientes durante el estudio. Dichos datos son muy importantes porque a partir de ellos se evaluará si el medicamento está listo para ser aplicado en los pacientes con determinada enfermedad y qué tan eficaz es.

1.2. Centros donde se realizan EC

Existen varios centros donde se gestionan EC, uno de ellos es el Centro de Inmunología Molecular (CIM). El objetivo principal de las investigaciones de este centro es la búsqueda de nuevos productos para el diagnóstico y tratamiento de enfermedades. En este centro cuando se crea un nuevo fármaco se escoge un grupo de personas sobre las cuales realizar el estudio, y la información asociada al EC es gestionada en el sistema Clínicas 1.0.

1.3. Clínicas

El sistema Clínicas 1.0 se encuentra desplegado en el CIM con el propósito acelerar y automatizar el diseño y conducción de EC en este centro. Este sistema consta de cuatro módulos: “Administrar Sistema” el cual permite administrar usuarios que intervendrán en los estudios, sus roles y las direcciones IP desde las cuales accederán al sistema; “Gestionar Estudio” permite la gestión de estudios, sus cronogramas, las hojas CRD, establecer las reglas para validar las variables de las hojas CRD y aprobar el diseño de los estudios; “Enviar Datos” posibilita gestionar sujetos, sus cronogramas específicos y la monitorización de los datos recogidos en los momentos de seguimiento; y finalmente en “Extraer Datos” se crean conjuntos de datos a partir de la información obtenida del módulo Enviar datos, que son exportados en diferentes formatos para que el cliente realice los análisis estadísticos que determinarán si el producto es seguro y eficaz en el tratamiento que se analiza.

Las reglas establecidas a cada campo de las Hojas CRD, para su posterior validación, pueden ser de dos tipos: derivación o validación. Las reglas de derivación consisten en asignar un valor a una variable y las de validación en comprobar que los datos de los campos de las hojas CRD son correctos. Estas permiten la detección de los errores que pueden cometerse al introducir los datos de los pacientes recogidos en el ensayo.

Actualmente el sistema no permite realizar operaciones aritméticas a través de las reglas de validación, lo que imposibilita la validación eficiente de los campos, provocando que el estudio arroje resultados incorrectos. Las reglas de derivación presentan el inconveniente de que para registrar los datos en el sistema el usuario debe indicar guardar dos veces a la hoja CRD por lo que la aplicación no es fácil de manejar para los usuarios. Por otra parte, una vez que el diseño del estudio es aprobado por el investigador promotor las reglas que fueron establecidas no pueden ser modificadas, situación que impide que se puedan agregar nuevas reglas o que sean perfeccionadas las existentes, trayendo como consecuencia pérdidas de tiempo y atraso en el estudio.

A continuación se realiza un estudio de sistemas existentes a nivel internacional y nacional que realizan validación de formularios dinámicos.

1.4. Estudio de los sistemas existentes

A nivel internacional y nacional se realiza un estudio de sistemas que puedan resolver los problemas que presentan las validaciones de los formularios dinámicos.

OpenClinica

OpenClinica es un software para la gestión de EC mediante la captura electrónica de datos (EDC), y la gestión clínica de la información, desarrollado en Estados Unidos. Está diseñado para gestionar adecuadamente una investigación de cualquier campo: medicina, biotecnología, alimentación, química, etc. Entre sus funcionalidades se encuentra la configuración de estudios rápidamente, simplificar el manejo de los estudios, facilitar la creación y gestión de hojas CRD. En este sistema se definen las reglas dentro de un fichero XML que es cargado a la aplicación y se encarga de la validación de los campos donde será introducida la información gestionada, en este fichero se especifica el objetivo de cada una de las reglas, dónde y cuándo se ejecutarán.

Sistema para el Control Farmacológico (Synta)

Es una aplicación para el control farmacológico, capaz de identificar en todo momento a quiénes fueron entregadas la recetas médicas en cada unidad de Cuba, controla los medicamentos consumidos a nivel nacional y lleva el control de todos los pacientes inscritos en las unidades de salud, farmacias. Como un módulo integrado a esta aplicación se encuentra el sistema Nomencladores, sistema de configuración y flexible ante posibles cambios de la información poco variable en el tiempo. De forma general el sistema cuenta con expresiones regulares definidas para realizar sus validaciones. Además para validar los nuevos campos que se crean en los formularios dinámicos, se emplea un mecanismo implementado con la ayuda de librerías de ExtJS o sea, al crear un nuevo campo muestra una interfaz para definir las propiedades que tendrá este, si sólo permite valores enteros, cuántos caracteres permitirá, etc.

Sistema de Información Hospitalaria (HIS)

El Sistema de Información Hospitalaria (HIS) desarrollado en la Universidad de las Ciencias Informáticas (UCI), es un sistema de gestión que permite a los hospitales la recolección, almacenamiento, procesamiento, recuperación y comunicación de información de atención al paciente. El mismo está concebido para llevar el control de las actividades de salud orientadas a los pacientes, permitiendo además gestionar y controlar los recursos de cada área de las instituciones hospitalarias.

En este sistema se utilizan varios tipos de validaciones, pueden ser, del lado del cliente se realizan a través de funciones JavaScript que son utilizadas generalmente para validar campos obligatorios, componentes de fecha y dependencia entre ellos. Se utilizan además las que provee JSF para chequear valores de mínimo y máximo de un campo, campos requeridos y tamaño mínimo de un campo. Por otra parte del lado del servidor se realizan validaciones propias del sistema o los procesos

que chequean valores aceptables o dependencias entre dichos valores. En este sistema además se encuentra el módulo Laboratorio, donde existen formularios dinámicos a los cuales se les realiza la validación en las clases controladoras usando también funciones JavaScript.

El estudio de los sistemas que realizan la validación de formularios dinámicos existentes a nivel internacional y nacional permitió conocer cómo realizan las validaciones de cada uno de ellos, sin embargo no pueden ser usados para dar solución al problema planteado pues las funcionalidades que contienen se ajustan a las necesidades de sus clientes y no aportaron información necesaria para la implementación de las nuevas funcionalidades del sistema SIGEC.

1.5. Metodologías, tecnologías y herramientas a considerar

En este epígrafe se enuncian los conceptos fundamentales relacionados con la metodología, tecnologías y herramientas asimiladas para el proceso de desarrollo de funcionalidades en la gestión de las reglas de derivación y validación para el sistema SIGEC.

1.5.1. Metodologías de desarrollo de software

Para desarrollar las funcionalidades en cuestión se utiliza una metodología que guíe el proceso de desarrollo de software. Una metodología define quién debe hacer qué, cuándo y cómo. Para el desarrollo del sistema SIGEC, en el centro CESIM se definió el uso de la metodología Proceso Unificado de Desarrollo (por sus siglas en inglés, RUP) y el uso del Lenguaje Unificado de Modelado (por sus siglas en inglés, UML) como lenguaje de modelado.

1.5.2. Proceso Unificado de Desarrollo

RUP es una forma disciplinada de asignar tareas y responsabilidades en una empresa de desarrollo (quién hace qué, cuándo y cómo). Los procesos de RUP estiman tareas y horario del plan, midiendo la velocidad de iteraciones concerniente a sus estimaciones originales. Busca detectar defectos en las fases iniciales. Intenta reducir el número de cambios tanto como sea posible y realiza el análisis y diseño tan completo como sea posible. Se caracteriza por ser guiado por casos de uso, centrado en la arquitectura, iterativo e incremental.

1.5.3. Lenguaje Unificado de Modelado

El Lenguaje Unificado de Modelado prescribe un conjunto de notaciones y diagramas estándar para modelar sistemas orientados a objetos, describe la semántica esencial de lo que estos diagramas y

símbolos significan. Se utiliza para visualizar, especificar, construir y documentar los artefactos de un sistema que involucra una gran cantidad de software.
Este lenguaje de modelado permite tener mayor rigor en la especificación, realizar la verificación y validación del modelo desarrollado, automatizar determinados procesos, así como generar código a partir de los modelos y viceversa (2).

Una vez expuestas algunas características acerca de la metodología de desarrollo se describen las tecnologías y herramientas que se utilizarán para el desarrollo de las nuevas funcionalidades.

1.6. Tecnologías

A raíz de la necesidad de solucionar los problemas que presenta la gestión de las reglas del módulo “Gestionar Estudio” del sistema Clínicas, se usan las tecnologías y herramientas definidas en el centro CESIM que permiten su uso sin costo de licencia.

1.6.1. Java

Java es un lenguaje de programación multiplataforma y orientado a objetos, fue diseñado para crear software altamente fiable para lo que proporciona numerosas comprobaciones en compilación y en tiempo de ejecución. Es altamente seguro e indiferente a la arquitectura pues está diseñado para soportar aplicaciones que serán ejecutadas en los más variados entornos de red. Además facilita y reduce el coste del desarrollo del software(3). Este lenguaje dispone de varias características dentro de las cuales se encuentran la disponibilidad de un amplio conjunto de bibliotecas, es simple y fácil de aprender y escribir, es interpretado y compilado a la vez además es robusto y dinámico.

La Máquina Virtual Java es el núcleo del lenguaje de programación Java. De hecho, es imposible ejecutar un programa Java sin ejecutar alguna implantación de la misma. Es la clave de muchas de las características principales de Java, como la portabilidad, la eficiencia y la seguridad. Se encarga de interpretar todo el código java y convertirlo al lenguaje nativo del sistema operativo en uso(4).

1.6.2. Java Server Faces (JSF)

JSF es una tecnología potente, flexible, basada en componentes diseñada para simplificar el desarrollo WEB en Java. Cuenta con excelentes herramientas, entornos de desarrollo de alta productividad y ricas bibliotecas. Facilita y agiliza la construcción de interfaces de usuario, pues realiza la programación a través de componentes y es basada en eventos (5). Los componentes JSF facilitan la construcción de interfaces WEB del lado del servidor, se conectan a fuentes de datos y relacionan de

forma transparente eventos del cliente con manejadores en el servidor (6). Esta tecnología permite desarrollar rápidamente aplicaciones de negocio dinámicas en las que toda la lógica del negocio esta implementada con Java. JSF presenta varias ventajas, resuelve validaciones, conversiones, mensajes de error e internacionalización, es extensible por lo que se pueden desarrollar nuevos componentes a la medida y permite introducir JavaScript en la página para acelerar la respuesta de la interfaz en el cliente.

1.6.3. RichFaces

RichFaces es una librería de componentes enriquecidos para JSF y además posee un framework avanzado para la integración sencilla de las funcionalidades Ajax dentro del desarrollo de las aplicaciones del negocio. RichFaces incluye ciclo de vida, validaciones, conversiones y la gestión de recursos estáticos y dinámicos. Los componentes de RichFaces están construidos con soporte Ajax, que puede ser fácilmente incorporado dentro de las aplicaciones JSF(7).

1.6.4. Ajax4jsf

Ajax4jsf es una librería de código abierto que se integra totalmente en la arquitectura de JSF y extiende la funcionalidad de sus etiquetas dotándolas con tecnología Ajax de forma limpia y sin añadir código JavaScript. Mediante este framework se puede variar el ciclo de vida de una petición JSF, recargar determinados componentes de la página sin necesidad de recargarla por completo, realizar peticiones automáticas al servidor, controlar cualquier evento de usuario, entre otras funciones. En definitiva Ajax4jsf permite dotar a la aplicación JSF de contenido mucho más profesional con muy poco esfuerzo.

1.6.5. Facelets

Java Server Facelets es un framework para plantillas. Es de código abierto, distribuido bajo la licencia Apache y constituye una tecnología alternativa de controlador de vista de JSF. No depende de un contenedor WEB. Soporta todos los componentes de interfaz de usuario JSF y construye su propio árbol de componentes (8).

1.6.6. XHTML

Lenguaje de Marcado de Hipertexto Extensible (XHTML, por sus siglas en inglés), es una versión más estricta y limpia del Lenguaje de Marcado de Hipertexto (HTML, por sus siglas en inglés), que nace

precisamente con el objetivo de remplazar a HTML ante su limitación de uso con las herramientas basadas en el Lenguaje de Marcado Extensible (XML). Su objetivo es avanzar en el proyecto del World Wide WEB Consortium (W3C) de lograr una WEB semántica, donde la información y la forma de presentarla estén claramente separadas (9).

1.6.7. Seam UI

Seam UI constituye una serie de controles JSF altamente integrables con JBoss Seam que añaden mejoras a JSF, desde validación, expresiones Extended EL, hasta la integración de la navegación en la interfaz de usuario basada en flujo de páginas o procesos del negocio.

1.6.8. Seam

Seam es una potente plataforma de desarrollo de código abierto utilizada para construir aplicaciones de Internet en Java. Integra tecnologías como Ajax, JSF, Java Persistence API (JPA), Enterprise Java Beans (EJB 3.0) y Business Process Management (BPM) en un sistema unificado con sofisticadas herramientas.

1.6.9. Hibernate

Hibernate es una herramienta de mapeo objeto-relacional para la plataforma Java que facilita el mapeo de atributos entre una base de datos relacional tradicional y el modelo de objetos de una aplicación, mediante archivos declarativos (XML) que permiten establecer estas relaciones. Es una tecnología de software libre distribuida bajo los términos de la licencia GNU Lesser General Public License (LGPL). Esta herramienta ofrece también un lenguaje de consulta de datos llamado Hibernate Query Language (HQL), al mismo tiempo que una Interfaz de Programación de Aplicaciones (API, por sus siglas en inglés) para construir las consultas automáticamente(10).

1.6.10. Enterprise Java Beans (EJB3)

La tecnología Enterprise JavaBeans (EJB) es un componente de arquitectura del lado del servidor para la plataforma Java Enterprise Edition (Java EE). El estándar EJB3 es desarrollado por Java Community Process (JCP). Encapsula la lógica del negocio de una aplicación y permite construir aplicaciones de negocio portables, reutilizables y escalables usando el lenguaje de programación Java. EJB3 puede residir en diferentes servidores y puede ser invocado por un cliente remoto. La lógica del negocio reside en los Enterprise Beans y no en el lado del cliente, permitiendo que el desarrollo del lado del cliente esté desacoplado de la lógica del negocio (11).

1.6.11. Java Persistence API (JPA)

Java Persistence API (JPA), es la API de persistencia desarrollada para la plataforma Java EE y está incluida en el estándar EJB3. Esta API busca unificar la manera en que funcionan las utilidades que proveen un mapeo objeto-relacional. El objetivo que persigue su diseño es no perder las ventajas de la orientación a objetos al interactuar con una base de datos, como sucedía con EJB2, y permitir usar objetos regulares conocidos como POJO (Plain Old Java Object)(12).

1.6.12. Java Enterprise Edition 5 (JEE 5)

Es la plataforma que provee Sun Microsystem para brindar soporte y desarrollar software para las empresas. JEE está enfocado en la creación de aplicaciones empresariales, las cuales se caracterizan por resolver problemas corporativos, lo que supone almacenamiento seguro, concurrencia, seguridad, escalabilidad y procesamiento distribuido. Dichos sistemas están construidos sobre una infraestructura base: los servidores de aplicaciones. Se puede definir a JEE como una arquitectura distribuida y multicapa, con especificaciones para empaquetar componentes redistribuibles para despliegue (6).

1.6.13. JBoss Application Server

JBoss Application Server es un servidor multiplataforma de aplicaciones J2EE, de código abierto, implementado en Java puro y orientado a la arquitectura de servicios. Por ser una plataforma certificada J2EE, soporta todas las funcionalidades de J2EE 1.4 e incluye servicios adicionales como persistencia. JBoss es ideal para aplicaciones Java y aplicaciones WEB. También soporta EJB 3.0, lo que hace el desarrollo de las aplicaciones mucho más simple. Con su utilización se pueden implementar sistemas con facilidad, únicamente es necesario colocar los archivos en el directorio /server/default/deploy (13).

1.7. Herramientas

1.7.1. Eclipse

Eclipse es un Entorno de Desarrollo Integrado (IDE, por sus siglas en inglés) de código abierto (su diseño permite fácil extensión por parte de terceros) y multiplataforma. Proporciona herramientas para gestionar espacios de trabajo, construir, lanzar y depurar aplicaciones y compartir objetos con el equipo de desarrollo. Está construido sobre un mecanismo para descubrir, integrar y ejecutar módulos (en inglés plug-ins) (14). Esto lo diferencia de otros entornos monolíticos donde las funcionalidades están todas incluidas, las necesite el usuario o no. Este mecanismo de módulos permite a Eclipse extenderse, usando otros lenguajes de programación como C/C++, Python y Java.

1.7.2. Sistemas de Gestión de Bases de Datos PostgreSQL

Es un Sistema Gestor de Base de Datos (SGBD) relacional de código abierto que puede ser ejecutado sobre la mayoría de los sistemas operativos que existen en la actualidad. Posee características sofisticadas como fiabilidad, Control de Concurrencia Multi-Versión (MVCC, por sus siglas en inglés), replicación asíncrona, transacciones anidadas, realización de respaldo de datos en línea, optimizador o planificador de consultas y soporta internacionalización. Es altamente escalable en cuanto a la cantidad de información que puede manejar y al número de usuarios concurrentes que puede alojar. Presenta características que no tienen otros SGBD como son los tipos de datos definidos por el usuario, la herencia y el uso de normas (15).

1.7.3. Visual Paradigm

Es una herramienta de Ingeniería de Software Asistida por Computadora (CASE, por sus siglas en inglés), que utiliza como lenguaje de modelado UML y que soporta el ciclo de vida completo del desarrollo de software: análisis y diseño orientados a objetos, construcción, pruebas y despliegue. Permite dibujar todos los tipos de diagramas de clases, código inverso, generar código desde diagramas y generar documentación. Se integra con varios IDE como Eclipse, NetBeans, VisualStudio, entre otros. Provee un generador de mapeo de objetos relacionales para los lenguajes de programación Java, PHP y algunos lenguajes del ambiente .Net (16).

Conclusiones

Se realizó un estudio a nivel internacional y nacional de sistemas que realizan validación de formularios dinámicos el cual demostró que no pueden ser utilizados para solucionar el problema planteado pues la forma en que realizan las validaciones de los formularios dinámicos no es la que se requiere realizar en SIGEC. Fueron descritas las características de la metodología de desarrollo, tecnologías y herramientas especificadas para el desarrollo de funcionalidades en el sistema SIGEC las cuales permitirán el desarrollo de la solución.

Capítulo 2: Características del sistema

El presente capítulo muestra el Modelo de Dominio, el cual describe el entendimiento común alcanzado por los involucrados respecto a los conceptos del dominio. Por otra parte, se exponen los requisitos funcionales y no funcionales, los cuales se transforman en casos de uso, con el objetivo de estructurar el Diagrama de Casos de Uso y a partir de los cuales se implementará el sistema.

2.1 Propuesta de solución

Debido a las deficiencias tecnológicas encontradas en el sistema Clínicas resulta engorroso para los desarrolladores incorporar nuevas funcionalidades, debido a esto el equipo de desarrollo decide implementar una nueva versión, llamada Sistema de Gestión de Ensayos Clínicos (SIGEC), que además de gestionar la información basado en Clínicas 1.0, incorporará nuevas funcionalidades en el proceso de gestión de las reglas en el módulo Diseño, y en otros módulos.

2.2. Modelo de dominio

El Modelo de Dominio es una representación visual de los conceptos u objetos que se manejan en el dominio del sistema. Los objetos o conceptos incluidos en el Modelo de Dominio no describen clases u objetos del software; sino entidades o conceptos del mundo real asociadas al problema en cuestión. Dicho modelo podrá ser utilizado como una base de las abstracciones relevantes en el proceso de construcción del sistema.

2.2.1. Conceptos fundamentales del dominio

Reglas: conjunto de normas que se establecen para que ciertos datos sean válidos.

Hoja CRD: formulario que contiene los datos de los estudios realizados a los pacientes.

MS: conjunto de hojas CRD en las cuales se recogen los datos del mismo día en que se realiza el estudio.

MS Programado: acontecimientos que se espera que ocurran. Se pueden planificar.

MS No Programado: acontecimientos que ocurren inesperadamente. No se pueden planificar.

Variable: campos que se encuentran en el formulario.

Dato: información que se recoge en los campos.

2.2.2. Diagrama de Modelo de dominio

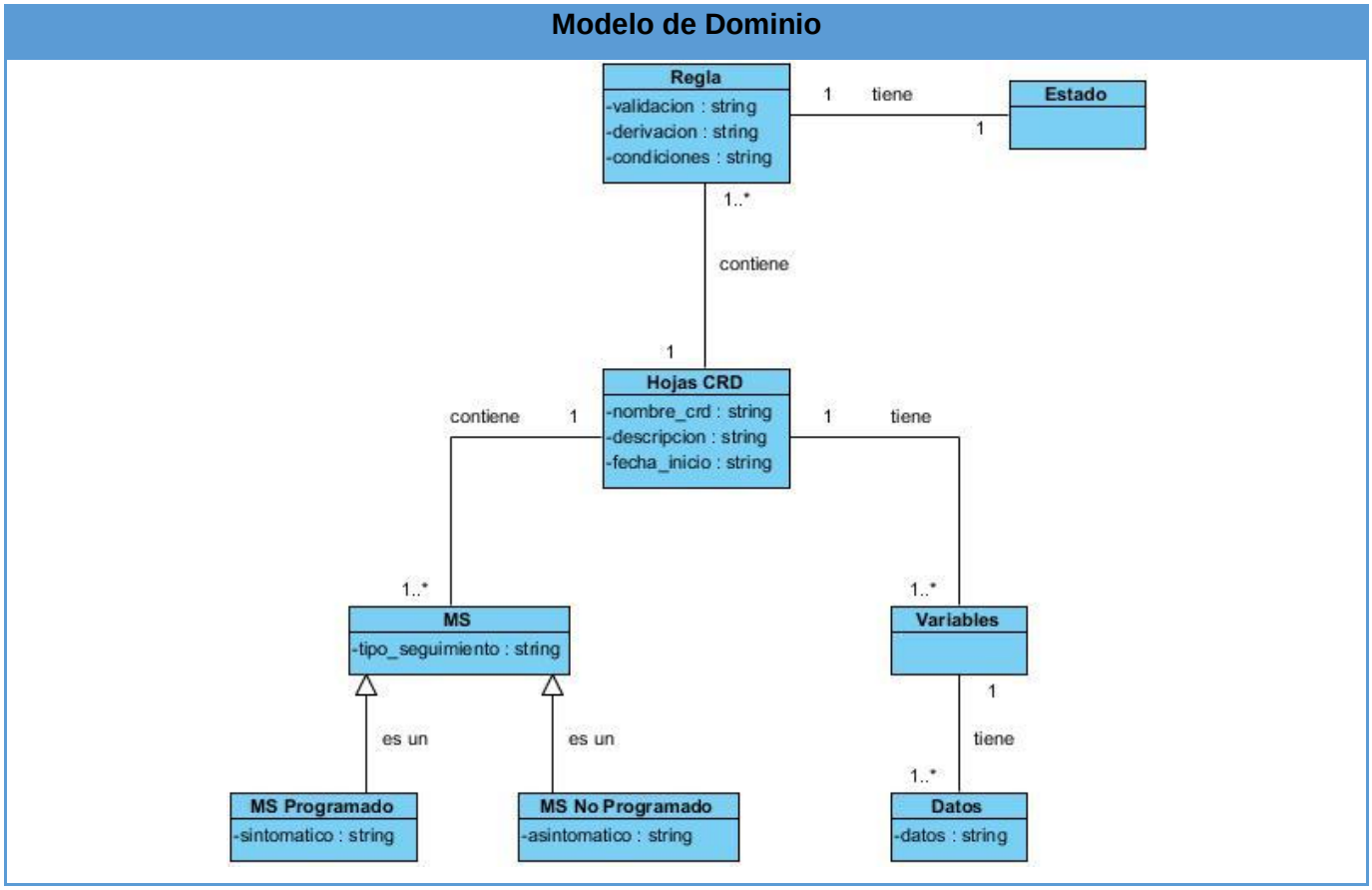

Figura.1 Modelo de dominio

En el diagrama de la figura 1, se observa que en el sistema existen varios conceptos asociados a la gestión de las reglas: por ejemplo los momentos de seguimiento que pueden ser, programados y no programados en estos se encuentran las hojas CRD que contienen varias variables y en estas van a ser registrados los datos del estudio, además encontramos las reglas, de validación y derivación, estas son aplicadas a los datos introducidos en las variables, tienen asociado un estado que puede ser: iniciado, no iniciado aprobado y completado.

2.3. Especificación de los requisitos de software

2.3.1. Requisitos funcionales del sistema

Los requisitos funcionales son capacidades o condiciones que el sistema debe cumplir y que definen el comportamiento interno del software. Estos describen los servicios que se espera que el sistema cumpla para satisfacer las necesidades del usuario.

El sistema debe permitir:

RF 1 Visualizar MS por visitas o generales

RF 2 Visualizar hoja CRD por MS

RF 3 Visualizar variables de una hoja CRD

RF 4 Adicionar variable de dependencia

RF 5 Eliminar variable de dependencia

RF 6 Visualizar variable de dependencia

RF 7 Adicionar casos

RF 8 Eliminar casos

RF 9 Establecer condición de una variable de dependencia

RF 10 Visualizar condición de una variable de dependencia

RF 11 Modificar condición de una variable de dependencia

RF 12 Establecer regla de derivación de una variable

RF 13 Visualizar reglas de derivación en dependencia del tipo de variable

RF 14 Modificar reglas de derivación en dependencia del tipo de variable

RF 15 Establecer regla de validación para una variable

RF 16 Visualizar reglas de validación en dependencia del tipo de variable

RF 17 Modificar reglas de validación en dependencia del tipo de variable

RF 18 Visualizar la validación de una variable

RF 19 Ejecutar las reglas de validación

RF 20 Visualizar el estado de la gestión de las reglas

RF 21 Visualizar el número de variables que faltan por completar

RF 22 Mostrar el nombre de las variables que faltan por completar

RF 23 Aprobar reglas

RF 24 Definir reglas iguales para todas las variables de una hoja

2.3.2. Requisitos no funcionales del sistema

Los requisitos no funcionales son propiedades o cualidades que el producto debe tener. Son características que hacen al producto atractivo, usable, rápido o confiable. Estos requerimientos se agrupan en varias categorías:

2.3.2.1 Usabilidad

- ✓ La aplicación web tendrá un ambiente sencillo y fácil de manejar para todos los usuarios incluso para aquellos inexpertos en el trabajo con aplicaciones informáticas.
- ✓ Para su correcto funcionamiento el sistema requiere un Servidor de Base de Datos con las siguientes características: Procesador Intel Dual Core, Sistema Operativo Windows 7 o superior y Linux, Memoria RAM 4GB, 500 GB de Disco Duro.
- ✓ Para su correcto funcionamiento el sistema requiere un Servidor de Aplicaciones con las siguientes características: Procesador Intel Dual Core, Sistema Operativo Windows 7 o superior y Linux, Memoria RAM 4GB y 120 GB de Disco Duro. Además, como servidor web JBoss4.2.2.
- ✓ Las PC Clientes deberán contar con las siguientes características: Procesador Intel Pentium IV o superior, Sistema Operativo Windows 7 o superior o Sistema Operativo Linux, memoria RAM 1GB y 80 GB de disco duro.
- ✓ El sistema podrá ser consultado desde los siguientes navegadores: Mozilla Firefox 3.6 o superior y Google Chrome 1.4 o superior.
- ✓ Debe estar instalada la máquina virtual de Java: java-7-openjdk para GNU Linux, o la jdk-6u3 o superior para Windows.

2.3.2.2 Confiabilidad

- ✓ La aplicación tendrá un sistema de trazas que registran el flujo constante de los datos y los responsables de sus cambios.
- ✓ Se mantendrá la seguridad y el control a nivel de usuario, garantizando el acceso de los mismos sólo a los niveles establecidos de acuerdo a la función que realizan.
- ✓ Ninguna información que se haya ingresado en el sistema será eliminada físicamente de la base de datos.

2.3.2.3 Restricciones de diseño

- ✓ Se usa como herramienta CASE Visual Paradigm para el modelado de los productos de trabajo generados en cada fase del ciclo de vida.
- ✓ Se usa como lenguaje de programación Java.
- ✓ Se usa como Gestor de Base de Datos PostgreSQL.

2.3.2.4 Interfaz

- ✓ Las páginas principales tendrán información que servirá de guía al usuario.
- ✓ Las páginas no están cargadas de imágenes.
- ✓ Cada rol tiene acceso a la interfaz que se corresponda con los permisos asignados.
- ✓ Se hará uso de simbología mediante íconos para indicar el estado de los elementos utilizados en el diseño. Además, los íconos contendrán funcionalidades específicas.

2.3.2.5 Estándares aplicables

- ✓ Contará con las pautas de diseño definidas para aplicaciones web del CESIM, permitiendo al usuario obtener de manera uniforme y organizada la información del sistema.

2.3.2.6 Seguridad

- ✓ El acceso a cualquier manipulación del sistema, tanto entrada como análisis de datos será sometido a un proceso de autenticación del usuario.
- ✓ Las contraseñas deberán tener más de siete caracteres de longitud y tener una fortaleza media. Los usuarios estarán obligados a cambiar la contraseña cada 90 días como máximo.
- ✓ Cada usuario tendrá asignado uno o varios roles en el sistema. Cada rol definido tendrá niveles de acceso a las funcionalidades del sistema.
- ✓ Cada modificación realizada en el sistema debe ser atribuible a un usuario particular según su autenticación.

2.4 Modelo de casos de uso del sistema

2.3.1. Actor del sistema

Actor	Descripción
Usuario	Es el encargado de realizar la supervisión de las reglas y de la validación y derivación de las variables de acuerdo a los permisos que le sean asignados.

Tabla 2.1 Definición de actores del sistema

Diagrama de casos de uso

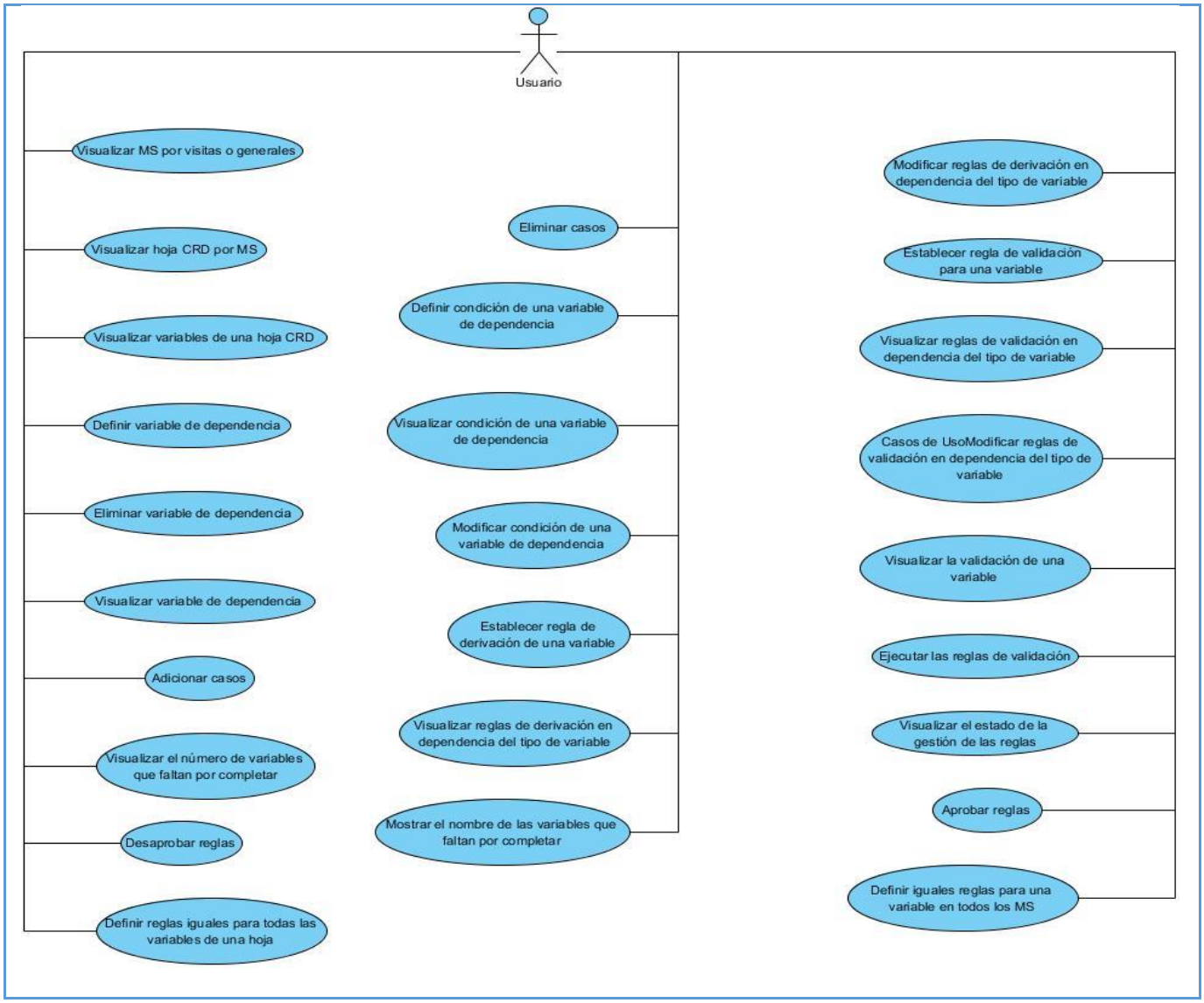

Fig.2 Diagrama de casos de uso

2.4.2.1 Descripción de los casos de uso

A continuación se describen algunos de los casos de uso que forman parte del diagrama anterior. La descripción del resto se encuentra en el artefacto SES - SIGEC - 010114a_ECU_Diseño_v1.1, que se encuentra en el expediente de proyecto de Ensayos Clínicos del departamento Sistemas para la Salud.

Caso de uso	
CU	Adicionar variable de dependencia
Propósito	Permitir al usuario adicionar una variable de dependencia.
Actores: Usuario	

Resumen: El sistema muestra la lista de variables de dependencia dándole la posibilidad al usuario de adicionar una variable.	
Referencias	
Acción del actor	**Respuesta del sistema**
Selecciona módulo Diseño	
	Muestra una interfaz para seleccionar en que parte del módulo diseño se desea trabajar.
Selecciona Gestión de reglas.	
	Muestra una interfaz para seleccionar “Gestionar reglas por días” o “Gestionar reglas iguales para todos los días”.
Sección Gestionar reglas por días	
Actor	**Sistema**
	Muestra un formulario para seleccionar el momento de seguimiento.
El usuario selecciona la opción “Seguimiento”	
	Muestra un formulario para seleccionar la hoja CRD en la que se va a trabajar.
Selecciona la hoja CRD	
	Muestra un formulario con el listado de variables de dependencia.
Selecciona la opción “Adicionar”	
	Muestra los datos a seleccionar que va a tener la variable de dependencia que se va añadir.
Indica adicionar	
	Guarda la información y se adiciona una nueva variable de dependencia.

Flujos alternos de la sección	
Acción del actor	**Respuesta del sistema**
Selecciona la opción para cancelar.	
	Finaliza el caso de uso.

Sección “Gestionar reglas iguales para todos los días”	
Actor	**Sistema**
Selecciona “Gestionar reglas iguales para todos los días”	
	Muestra un formulario para seleccionar el momento de seguimiento.
El usuario selecciona la opción “Seguimiento”	
	Muestra un formulario para seleccionar la hoja CRD en la que se va a trabajar.
Selecciona la hoja CRD	
	Muestra un formulario con el listado de variables de dependencia.
Selecciona la opción “Adicionar”	
	Muestra los datos a seleccionar que va a tener la variable de dependencia que se va añadir.
Indica adicionar	
	Guarda la información y se adiciona una nueva variable de dependencia, finalizando el caso de uso.

Flujos alternos de la sección	
Actor	**Sistema**
Selecciona la opción para cancelar.	
	Finaliza el caso de uso.

Puntos de extensión.
(11)

Tabla 2.2 Descripción del caso de uso: Adicionar variable de dependencia

Caso de uso		
CU	Aprobar cronograma	
Propósito	Permitir al usuario aprobar el cronograma de ejecución de un estudio.	
Actores: Usuario		
Resumen: El caso de uso inicia cuando el usuario decide aprobar el cronograma de ejecución de un estudio. El sistema cambia el estado del cronograma a “Aprobado”.		
Referencias	(5)	
Acción del actor		**Respuesta del sistema**
Selecciona la opción “Aprobar reglas” en la interfaz que permite la gestión de las reglas.		
		Muestra un mensaje de alerta al usuario indicando que una vez aprobado el cronograma, no se podrán realizar más cambios en el mismo. Pregunta al usuario si desea confirmar o cancelar la acción.
Selecciona la opción para confirmar.		
		Establece el estado “Aprobado” al cronograma, no permitiendo realizar más cambios en este. Brinda la opción de desaprobar el cronograma. Finaliza el caso de uso.
Flujo alternativo		
Acción del actor		**Respuesta del sistema**
Selecciona la opción para cancelar.		
		Finaliza el caso de uso.
Puntos de extensión.		

(11)

Tabla 2.3 Descripción del caso de uso: Aprobar cronograma

Caso de uso		
CU	Adicionar caso	
Propósito	Permitir al usuario adicionar casos.	
Actores: Usuario		
Resumen: El sistema muestra la lista de casos, dándole la posibilidad al usuario de que añada un caso.		
Referencias	(5)	
Acción del actor		**Respuesta del sistema**
Selecciona gestionar reglas.		
		Muestra una interfaz para seleccionar “Gestionar reglas por días” o “Gestionar reglas iguales para todos los días”.
Selecciona “Gestionar reglas por días” o “Gestionar reglas iguales para todos los días”		
		Muestra un formulario para seleccionar el momento de seguimiento.
El usuario selecciona la opción “Seguimiento”		
		Muestra un formulario para seleccionar la hoja CRD en la que se va a trabajar.
Selecciona la hoja CRD		
		Muestra un formulario con el listado de casos.
Selecciona adicionar		
		Verifica que existan variables de dependencia. Si existen variables de dependencia, el sistema actualiza el listado de casos que se está visualizando, agregando una fila para establecer

	condiciones y derivaciones para la variable (un caso); ir al paso 1 del flujo normal de eventos.
Flujo alternativo	
Acción del actor	**Respuesta del sistema**
	Si no existen variables de dependencia, el sistema actualiza el listado de casos que se está visualizando, agregando una fila para establecer condiciones y derivaciones para la variable (un caso), permitiéndole al actor adicionar solamente un caso con condición: *por defecto*, finalizando el caso de uso.
Puntos de extensión.	
(11)	

Tabla 2.4 Descripción del caso de uso: Adicionar casos

Conclusiones

La realización del modelo del dominio, la descripción de los requisitos tanto funcionales como no funcionales y las especificaciones de los casos de uso del sistema, permitió lograr un mejor entendimiento del negocio para el posterior diseño e implementación de las funcionalidades.

Capítulo 3: Diseño del sistema

Con el objetivo de obtener una solución óptima surgen elementos importantes a considerar como los que serán descritos en este capítulo. Se realiza una descripción detallada de la arquitectura que presenta el sistema. Se presenta el patrón arquitectónico usado y los principales patrones de diseño puestos en práctica. Son representados además los diagramas de paquetes y clases del diseño que forman parte del Modelo de Diseño.

3.1. Descripción de la arquitectura

La arquitectura de software, es un conjunto de patrones y abstracciones coherentes que proporcionan un marco de referencia necesario para guiar la construcción de un software dentro de un sistema informático. Permite a los programadores, diseñadores, ingenieros y analistas trabajar bajo una línea común que posibilite la compatibilidad necesaria para lograr el objetivo deseado.

De acuerdo con la IEEE la arquitectura es:

- ✓ El nivel conceptual más alto de un sistema en su ambiente.
- ✓ La organización fundamental de un sistema descrita en:
    - Sus componentes.
    - Relación entre ellos y con el ambiente.
    - Principios que guían su diseño y evolución.

3.1.1 Arquitectura cliente-servidor

Esta arquitectura intenta proveer usabilidad, flexibilidad, interoperabilidad y escalabilidad en las comunicaciones. Permite a los usuarios finales obtener acceso a la información de forma transparente, desde diferentes lugares y aún en entornos multiplataforma. Este modelo consiste básicamente en que el cliente envía un mensaje solicitando un determinado servicio a un servidor (realiza una petición), que envía uno o varios mensajes con la respuesta (provee el servicio solicitado) (17).

Esta arquitectura presenta varias ventajas: recursos centralizados (se administran a nivel de servidor los recursos comunes a todos los usuarios, por ejemplo, una base de datos centralizada), seguridad mejorada y red escalable (es posible quitar o agregar clientes sin afectar el funcionamiento de la red y sin la necesidad de realizar mayores modificaciones) (18).

3.1.2. Patrón arquitectónico: Modelo Vista Controlador (MVC)

Un patrón de arquitectura de software describe un problema particular y recurrente del diseño, que surge en un contexto específico, y presenta un esquema genérico y probado de su solución.

El patrón MVC está diseñado para reducir el esfuerzo de programación necesario en la implementación de sistemas múltiples y sincronizados de los mismos datos. Hace a los sistemas más flexibles y adaptables. Su característica principal es que el Modelo, las Vistas y los Controladores se tratan como entidades separadas, lo que permite su desarrollo independiente, garantizando así la actualización y mantenimiento del software de forma sencilla y en un reducido espacio de tiempo. Además presenta la ventaja de que la conexión entre el Modelo y sus Vistas es dinámica, o sea, se produce en tiempo de ejecución, no en tiempo de compilación (19).

Modelo: El Modelo es el objeto que representa los datos del programa. Maneja los datos y controla todas sus transformaciones. El Modelo no tiene conocimiento específico de los Controladores o de las Vistas, ni siquiera contiene referencias a ellos. Es el propio sistema el que tiene encomendada la responsabilidad de mantener enlaces entre el Modelo y sus Vistas, y notificar a las Vistas cuando cambia el Modelo (19).

Vista: La Vista es el objeto que maneja la presentación visual de los datos representados por el Modelo. Genera una representación visual del Modelo y muestra los datos al usuario. Interactúa preferentemente con el Controlador, pero es posible que trate directamente con el Modelo a través de una referencia al propio Modelo (19).

Controlador: El Controlador es el objeto que proporciona significado a las órdenes del usuario, actuando sobre los datos representados por el Modelo, centra toda la interacción entre la Vista y el Modelo. Cuando se realiza algún cambio, entra en acción, bien sea por cambios en la información del Modelo o por alteraciones de la Vista. Interactúa con el Modelo a través de una referencia al propio Modelo (19).

Entre sus ventajas se puede señalar que el MVC aporta una construcción de software fácil de mantener, en la que se pueden localizar de forma ágil los errores. Supone un diseño modular, y muy poco acoplado, favoreciendo la reutilización. El uso de este patrón trae asociado directamente un grupo de patrones de diseño.

3.1.3. Patrones de diseño

Los patrones pretenden ser la solución al problema de la comunicación de experiencias entre profesionales del software. Cada patrón describe un problema concurrente en el entorno, para describir después el camino a la solución a ese problema, de tal forma que pueda ser reutilizado en distintos

proyectos**.** Un patrón de diseño proporciona un esquema para refinar los subsistemas o componentes de un software y las relaciones entre ellos. Describe estructuras recurrentes de comunicar componentes que resuelven un problema de diseño en un contexto particular. Son patrones de un nivel de abstracción menor que los patrones de arquitectura. Están por lo tanto más próximos al código fuente final. Su uso no se refleja en la estructura global del sistema.

Los patrones utilizados para el diseño del sistema son los Patrones de Software para la Asignación General de Responsabilidad (GRASP, por sus siglas en inglés) que describen los principios fundamentales de diseño de objetos para la asignación de responsabilidades. A las clases se asignaron las responsabilidades que podían realizar según la información contenida.

Dentro de los patrones GRASP se encuentran cuatro patrones muy utilizados: Experto, Creador, Alta Cohesión y Bajo Acoplamiento.

- ✓ Experto: es el principio básico de asignación de responsabilidades. Indica que la responsabilidad de la creación de un objeto debe recaer sobre la clase que conoce toda la información necesaria para crearlo.
- ✓ Creador: el patrón creador ayuda a identificar quién debe ser el responsable de la creación (o instanciación) 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, usa directamente las instancias creadas del objeto, almacena o maneja varias instancias de la clase.
- ✓ Alta cohesión: expresa que la información que almacena una clase debe ser coherente y se encuentra en la mayor medida de lo posible relacionada con la clase.
- ✓ Bajo acoplamiento: es la idea de tener las clases lo menos ligadas entre sí posible. De tal forma que en caso de producirse una modificación en alguna de ellas, se tenga la mínima repercusión posible en el resto de clases, potenciando la reutilización, y disminuyendo la dependencia entre las clases.
- ✓ Controlador: Asigna la responsabilidad de controlar el flujo de eventos del sistema, a clases específicas. Esto facilita la centralización de actividades (validaciones, seguridad). El controlador no realiza estas actividades, las delega en otras clases con las que mantiene un modelo de alta cohesión. Se recomienda dividir los eventos del sistema en el mayor número de controladores para poder aumentar la cohesión y disminuir el acoplamiento.

3.1.4. Diagrama de paquetes

Diagrama de paquetes

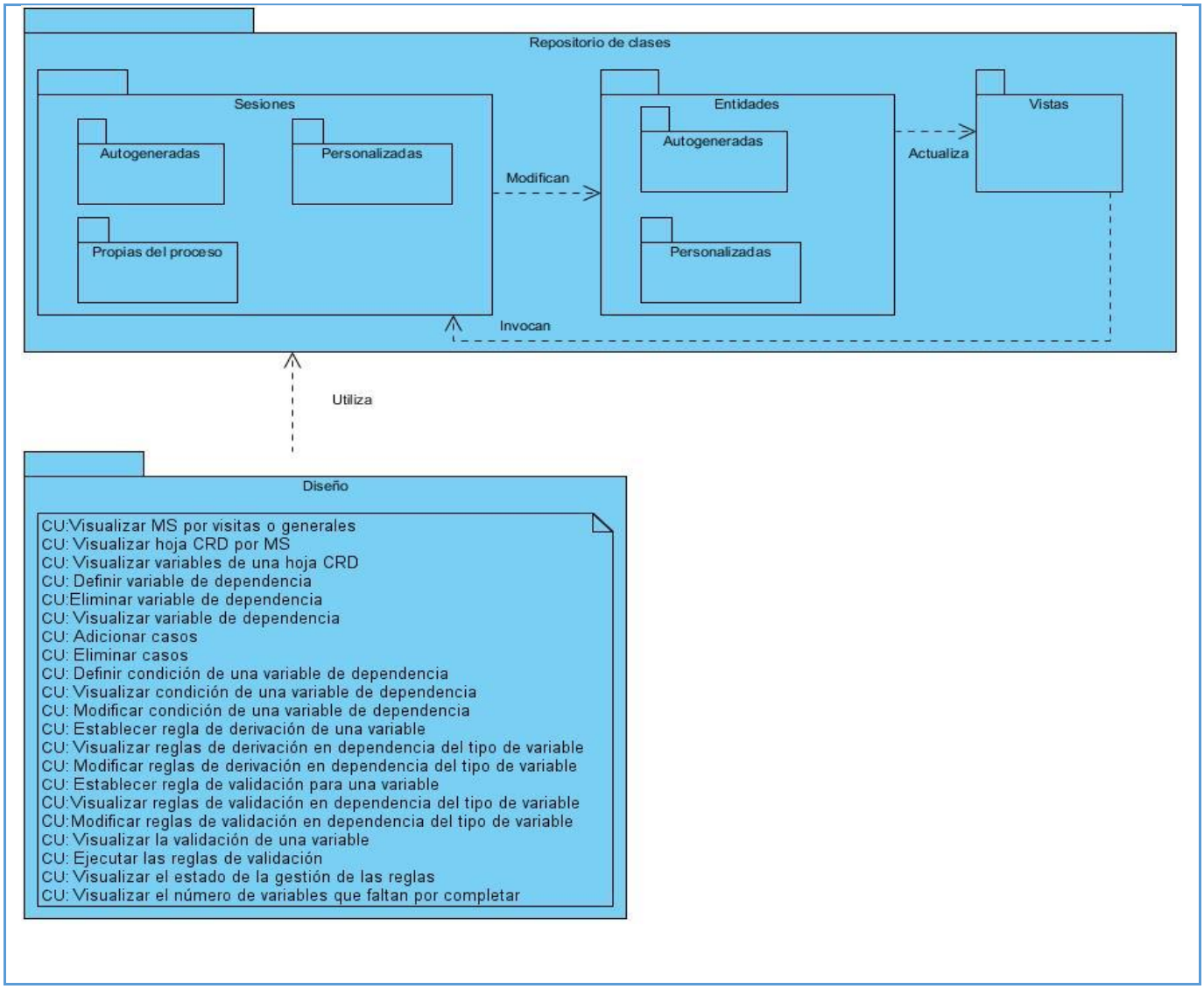

Fig.3 Diagrama de paquetes

3.2. Diagramas de clases del diseño

Un diagrama de clases es un tipo de diagrama estático que describe la estructura de un sistema, muestra sus clases, atributos y las relaciones entre ellos. Los diagramas de clases son utilizados durante el proceso de análisis y diseño de los sistemas, donde se crea el diseño conceptual de la información que se manejará en el sistema, y los componentes que se encargarán del funcionamiento y la relación entre uno y otro.

3.2.1. Descripción de elementos del diagrama de clases del diseño

Nombre: PC_momentos_seguimiento

	
Descripción	Son las páginas WEB, las cuales se encuentran del lado del cliente y proveen la interacción directa con el usuario y se ejecutan sobre un navegador.

Tabla 3.1 Descripción de elementos del diseño: Página cliente

Nombre: Form_momentos_seguimiento	
	
Descripción	Contienen un conjunto de elementos de entrada que están contenidos en las páginas clientes. Su función es enviar directamente la información a las páginas servidoras.

Tabla 3.2 Descripción de elementos del diseño: Formulario

Nombre: SP_momentos_seguimiento	
	
Descripción	Se ejecutan del lado del servidor. Su objetivo es proveer una respuesta a las peticiones realizadas desde la vista. Gestionan una acción específica.

Tabla 3.3 Descripción de elementos del diseño

La descripción de los otros diagramas se encuentra en el artefacto SES - SIGEC - 010215_MDI_DISEÑO_v1.0 en el departamento de Sistemas para la Salud, en el proyecto Ensayos Clínicos.

Estos diagramas de clases del diseño estarán compuestos por páginas servidoras que construyen páginas clientes que a su vez contienen formularios que capturaran y mostraran la información enviándolas a las páginas servidoras. Estas páginas servidoras invocan métodos o responsabilidades en la clase controladora que según la acción solicitada puede modificar o consultar las entidades.

3.2.2. Diagrama de clases del diseño: Adicionar variable de dependencia

Diagrama de clases del diseño

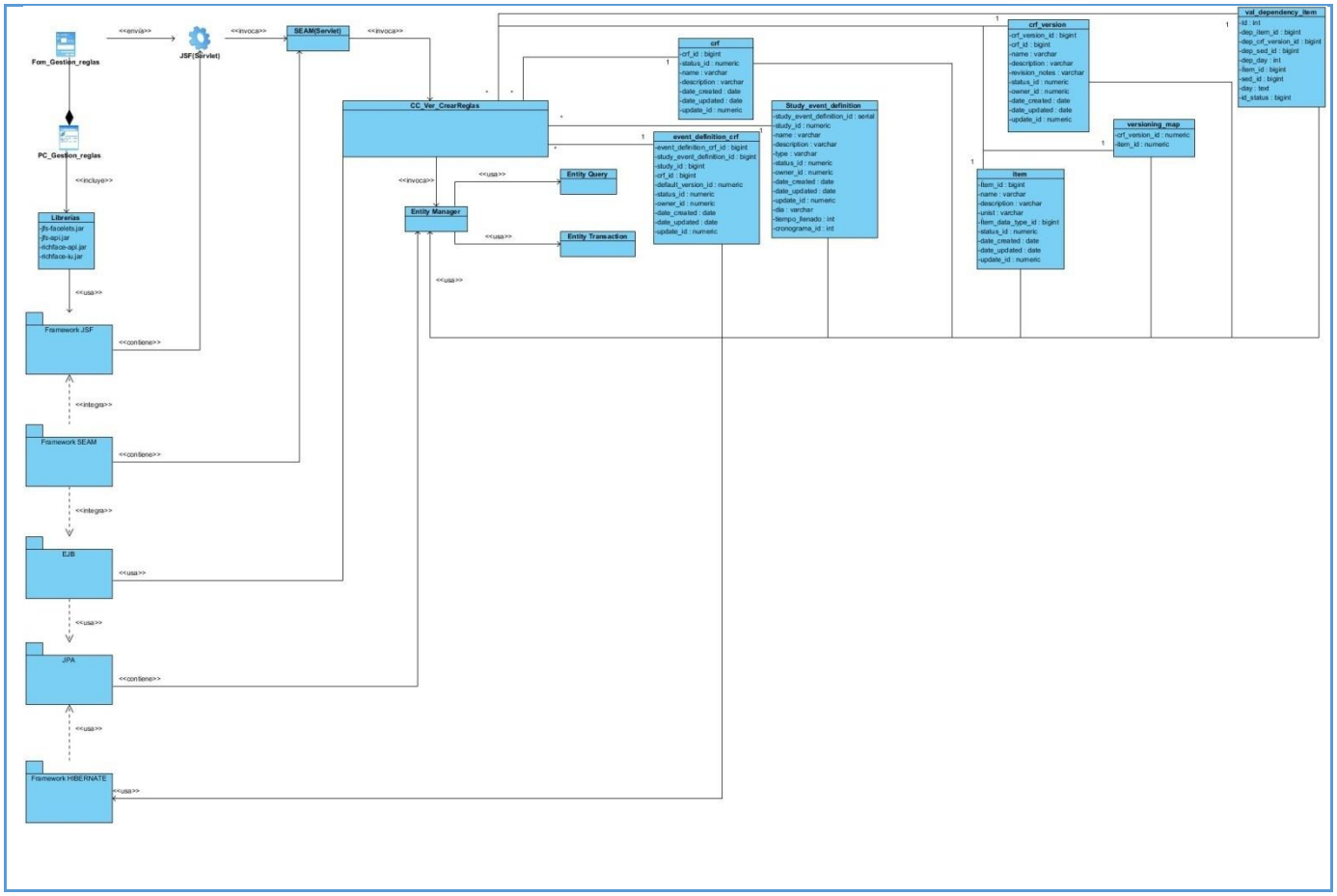

Fig.4 Diagrama de clases del modelo de diseño

3.2.3. Diagrama de clases del diseño: Adicionar casos

Diagrama de clases del diseño

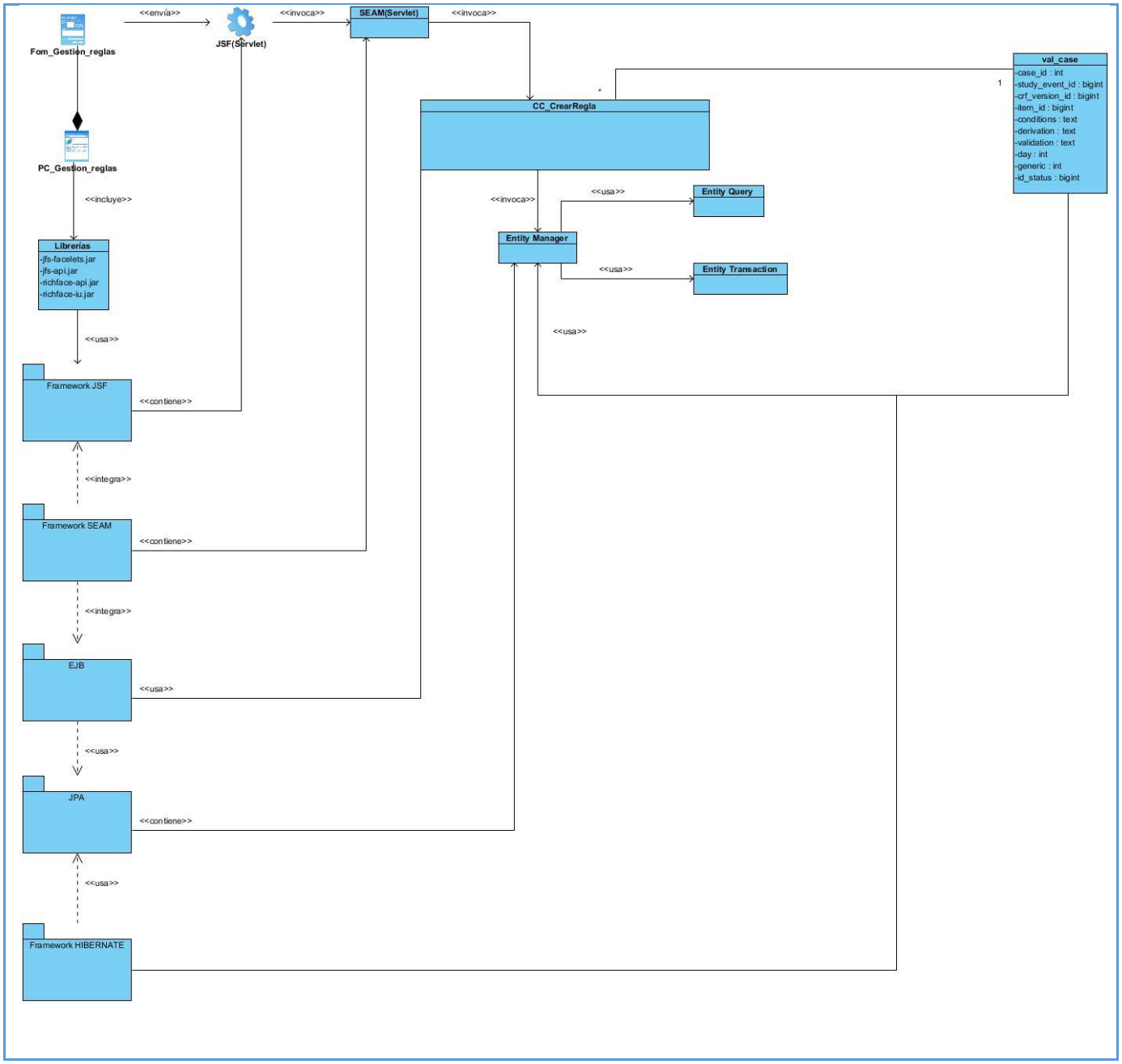

Fig.5 Diagrama de clases del modelo de diseño

3.2.4. Diagrama de clases del diseño: Aprobar reglas

Diagrama de clases del diseño

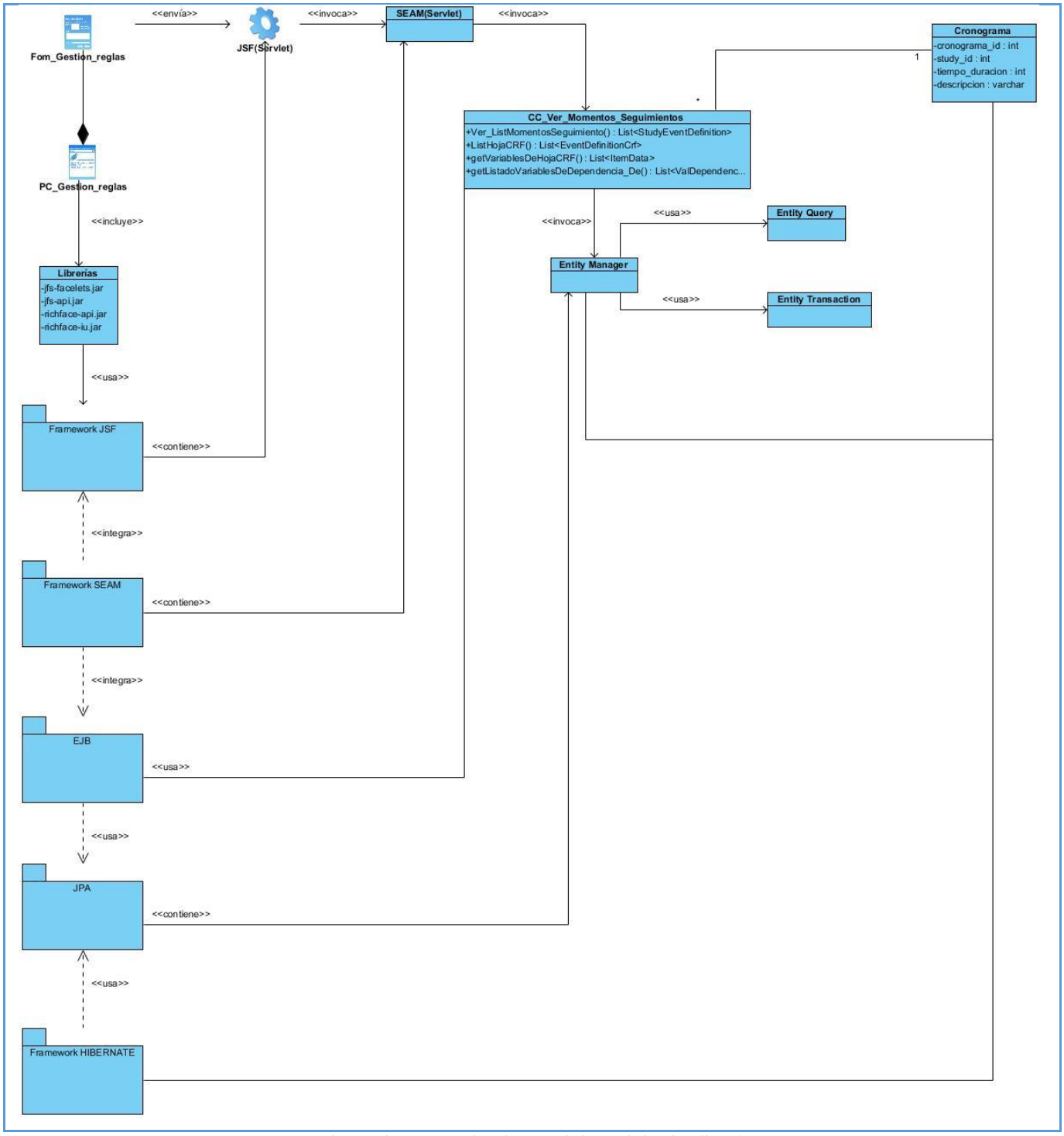

Fig.6 Diagrama de clases del modelo de diseño

A continuación se realiza una descripción de las clases que han sido identificadas en el diseño para su futura implementación, con el objetivo de lograr una mejor comprensión del sistema.

Nombre: CC_Ver_Momentos_Seguimiento

Controladora	
Atributo	**Tipo**
studioActivo	Study
Cronograma	Cronograma
momentosSeguimientos	List<StudyEventDefinition>
momentosSeguimientosProgramados	List<StudyEventDefinition>
momentosSeguimientosNoProgramados	List<StudyEventDefinition>
etapasCronograma	List<Etapa>
Variables	List<Item>
Para cada responsabilidad:	
Nombre:	Ver_ListMomentosSeguimiento()
Descripción:	Permite obtener el listado de todos los momentos de seguimiento.
Nombre:	ListHojaCRF()
Descripción:	Permite obtener el listado de todas las hojas CRF.
Nombre:	getVariablesDeHojaCRF()
Descripción	Permite obtener la variable de la hoja indicada.
Nombre:	getListadoVariablesDeDependencia_De()
Descripción:	Permite obtener un listado de variables de dependencia.

Tabla 3.4 Descripción de las clases del diseño: Controladora

Nombre: CC_CrearRegla	
Controladora	
Descripción:	Permite gestionar todo lo relacionado con las reglas del sistema.

Nombre: CE_Item	
Entidad	
Atributo	**Tipo**
item_id	Bigint
Name	Varchar
description	Varchar
Units	Varchar
phi_status	Boolean
item_data_type_id	Bigint
item_reference_type_id	Numeric
owner_id	Numeric
date_created	Date
date_updated	Date
update_id	Numeric
oc_oid	Varchar
Para cada responsabilidad:	
Descripción	Es donde se almacenan todas las variables del estudio.

Tabla 3.5 Descripción de las clases del diseño: Entidad

Nombre: CE_crf	
Entidad	
Atributo	**Tipo**
crf_id	Bigint
status_id	Bigint

Name	Varchar
description	Varchar
date_created	Date
date_updated	Date
update_id	Numeric
Para cada responsabilidad:	
Descripción	Almacena todas las hojas CRD del sistema.

Tabla 3.6 Descripción de las clases del diseño: Entidad

Nombre: crf_version	
Entidad	
Atributo	**Tipo**
crf_version_id	bigint
crf_id	bigint
Name	varchar
Description	varchar
revision_notes	varchar
status_id	numeric
owner_id	numeric
date_created	date
date_updated	date
update_id	numeric
Para cada responsabilidad:	

Descripción	Almacena las versiones de las hojas CRD.

Tabla 3.7 Descripción de las clases del diseño: Entidad

Nombre: CE_event_definition_crf	
Entidad	
Atributo	**Tipo**
event_definition_crf_id	Bigint
study_event_definition_id	Bigint
study_id	Bigint
crf_id	Bigint
default_version_id	Numeric
status_id	Numeric
owner_id	Numeric
date_created	Date
date_updated	Date
update_id	Numeric
Para cada responsabilidad:	
Descripción	Contiene las relaciones de los momentos de seguimiento y las hojas CRD.

Tabla 3.8 Descripción de las clases del diseño: Entidad

Nombre:study_ event_definition	
Entidad	
Atributo	**Tipo**

study_event_definition_id	Bigint
study_id	Bigint
Name	Varchar
description	Varchar
Type	Varchar
status_id	Numeric
owner_id	Numeric
date_created	Date
date_updated	Date
update_id	Numeric
Dia	Varchar
tiempo_llenado	Integer
cronograma_id	Bigint
Para cada responsabilidad:	
Descripción	Guarda los momentos de seguimiento.

Tabla 3.9 Descripción de las clases del diseño: Entidad

Nombre:val_dependency_item	
Entidad	
Atributo	**Tipo**
Id	integer
dep_item_id	bigint

dep_crf_version_id	bigint
dep_sed_id	bigint
dep_day	integer
Ítem_id	bigint
sed_id	bigint
Day	text
id_status	bigint
Para cada responsabilidad:	
Descripción	Guarda las variables del sistema con sus respectivas variables de dependencia.

Tabla 3.10 Descripción de las clases del diseño: Entidad

Nombre:versioning_map	
Entidad	
Atributo	**Tipo**
crf_version_id	numeric
item_id	numeric
Para cada responsabilidad:	
Descripción	Contiene la relación de las tablas item y crf_version, almacena los identificadores

Tabla 3.11 Descripción de las clases del diseño: Entidad

Conclusiones

El uso del patrón arquitectónico MVC permitió la distribución óptima de funcionalidades para el desarrollo del sistema. El empleo de los patrones de diseño posibilitó que las clases involucradas en el

desarrollo de las funcionalidades realizaran sus funciones específicas. Los diagramas de clases del diseño permitieron estructurar la aplicación, sus clases y atributos, definiendo el diseño conceptual de la información y los componentes que se encargarán del funcionamiento.

Capítulo 4: Implementación

La implementación es la realización de una especificación técnica o algoritmos como un programa o componente de software. Las especificaciones del diseño físico son convertidas a código fuente. Se hace alusión a los elementos físicos, que son necesarios para el despliegue del sistema desarrollado. Además se hace referencia a los posibles errores que pueda tener el sistema y cómo se garantizará la seguridad.

4.1. Modelo de datos

El Modelo de datos es la traducción del análisis de requisitos al esquema conceptual, mediante una representación gráfica de las entidades y sus relaciones. Es usado para definir el mapeo entre las clases del diseño y las estructuras de datos. Está compuesto por entidades, atributos y sus relaciones.

- ✓ Las entidades son objetos de los que el sistema necesita guardar información.
- ✓ Los atributos son las características asociadas a una entidad. Estos pueden ser clasificados en obligatorios, opcionales, claves foráneas y claves primarias (estas se dividen en simples y compuestas).
- ✓ Las relaciones, por su parte, muestran la forma en que dos entidades se asocian. Se representan mediante una línea que une a las dos entidades implicadas y manifiestan dos características principales: la cardinalidad y la obligatoriedad.
- ✓ La cardinalidad se refiere al número de ocurrencias de una entidad con respecto a la otra. La entidad de donde parte la relación tendrá tantas ocurrencias como indique el número asociado a la entidad a donde llega la relación señalando con una flecha el sentido de entrada, de no mostrarse el número de la cardinalidad se asume que la ocurrencia es de solamente una vez.
- ✓ La obligatoriedad determina que ante la existencia de una entidad pueden existir ocurrencias de otras relacionadas con esta. Si la ocurrencia es obligatoria se representa mediante una línea continua, en caso contrario, se realiza a través de una línea discontinua.

En la figura 6 se muestra el modelo de datos utilizado para el desarrollo de la gestión de las reglas.

Modelo de datos

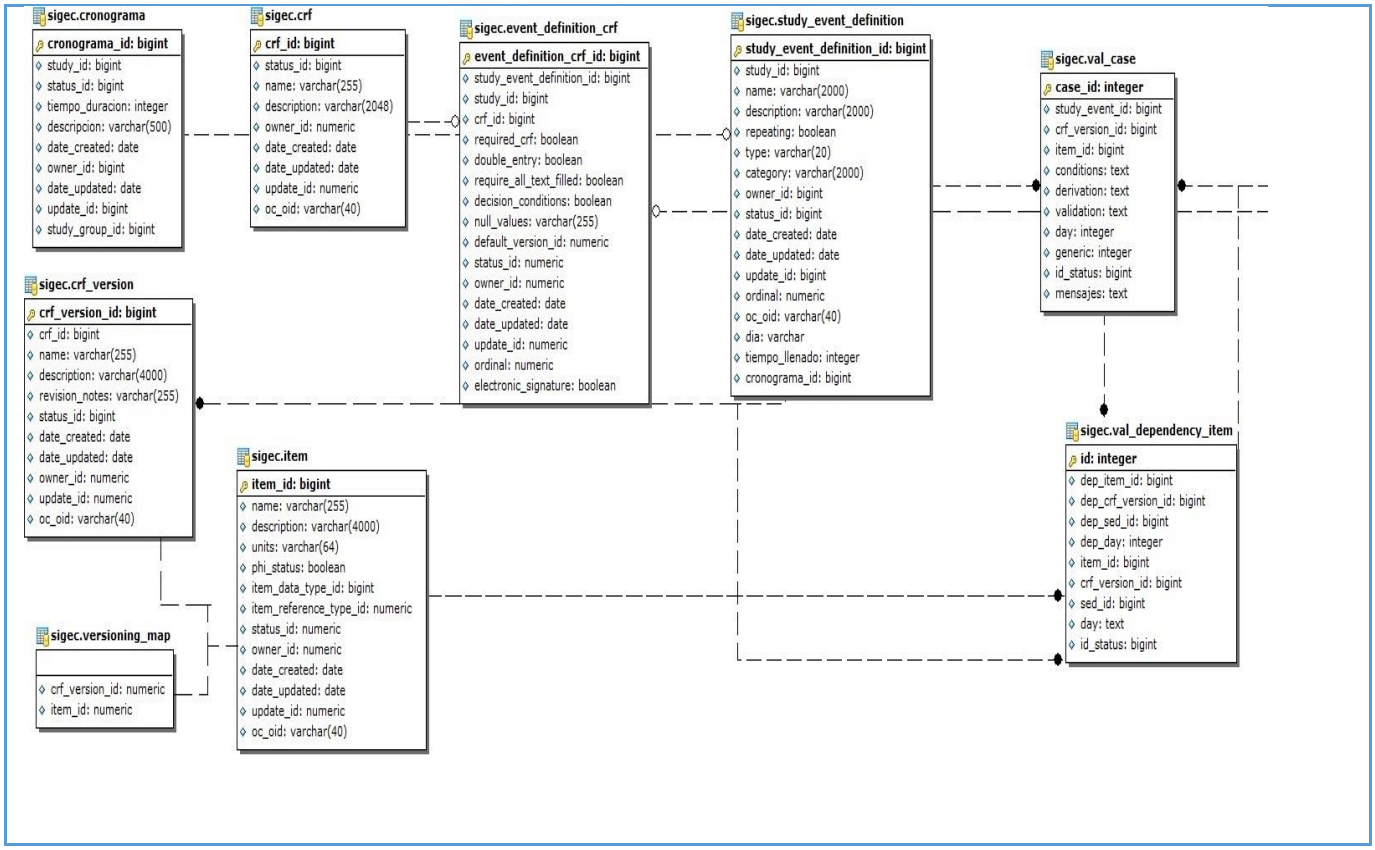

Fig.6 Diagrama modelo de datos

A continuación se describen las entidades que conforman el modelo anterior.

4.1.1. Descripción de las tablas de la base de datos

Nombre: Item		
Descripción: Almacena todas las variables del estudio.		
Atributo	**Tipo**	**Descripción**
Ítem_id	bigint	Identificador de la variable.
Name	varchar	Nombre de la variable.
Description	varchar	Guarda la descripción de la variable.
Unist	varchar	Unidad de la variable.
Ítem_data_type_id	bigint	Identificador del tipo de dato de la variable.
status_id	numeric	Indica en el estado en que se encuentra la

		variable.
date_created	date	Guarda la fecha de creación de la variable.
date_updated	date	Guarda la fecha en que fue modificada la variable.
update_id	numeric	Identificador para cada vez que se modifique una variable.

Tabla 4.1 Descripción de la entidad: item

Nombre: Cronograma		
Descripción: Guarda los cronogramas y se seleccionan los asociados al estudio.		
Atributo	**Tipo**	**Descripción**
cronograma_id	Bigint	Identificador del cronograma.
study_id	Bigint	Indica el estudio activo en el que se realizó el cronograma.
status_id	Bigint	Indica el estado del cronograma.
tiempo_duracion	Integer	Tiempo esperado de duración del cronograma.
descripcion	varchar	Describe el cronograma.
date_created	date	Fecha de creación del cronograma.
owner_id	bigint	Usuario que insertó o modificó el cronograma.
date_updated	date	Fecha en que se actualizó el cronograma.
update_id	bigint	Identificador para cada vez que se modifique un cronograma determinado.
study_group_id	bigint	Grupo de sujetos al cual pertenece el cronograma.

Tabla 4.2 Descripción de la entidad: cronograma

Nombre: crf		
Descripción: Almacena todas las hojas CRD del sistema.		
Atributo	**Tipo**	**Descripción**
crf_id	bigint	Identificador de la hoja CRD.
status_id	bigint	Identificador del estado en que se encuentra la hoja CRD.
Name	varchar	Guarda el nombre de la hoja.
description	varchar	Descripción de la hoja.
date_created	date	Guarda la fecha de creación de la hoja.
date_updated	date	Guarda la fecha en que fue actualizada la hoja.
update_id	numeric	Identificador para cada vez que se modifique una hoja.

Tabla 4.3 Descripción de la entidad: crf

Nombre: crf_version		
Descripción: Almacena las versiones de las hojas CRD.		
Atributo	**Tipo**	**Descripción**
crf_version_id	bigint	Identificador de la versión de la hoja CRD.
crf_id	bigint	Identificador de la hoja.
Name	varchar	Nombre de la versión.
Description	varchar	Descripción de la versión.
revision_notes	varchar	Notas de la revisión de la versión.
status_id	numeric	Estado de la versión.
owner_id	numeric	Usuario que modificó o insertó la versión.

date_created	date	Fecha en que se creó la versión.
date_updated	date	Fecha en que se actualizó la versión.
update_id	numeric	Identificador para cada vez que se modifique una versión.

Tabla 4.4 Descripción de la entidad: crf_version

Nombre: event_definition_crf		
Descripción: Contiene las relaciones de los momentos de seguimiento y las hojas CRD.		
Atributo	**Tipo**	**Descripción**
event_definition_crf_id	bigint	Identificador de la hoja CRD asociada al momento de seguimiento.
study_event_definition_id	bigint	Identificador del momento de seguimiento.
study_id	bigint	Estudio activo en el que se introdujo la hoja en el momento de seguimiento.
crf_id	bigint	Identificador de la hoja CRD asociada.
default_version_id	numeric	Versión de la hoja CRD.
status_id	numeric	Estado de la hoja CRD asociada al momento de seguimiento.
owner_id	numeric	Usuario activo.
date_created	date	Fecha de creación.
date_updated	date	Fecha de actualización.
update_id	numeric	Identificador para cada vez que se modifique una hoja asociada al momento de seguimiento.

Tabla 4.5 Descripción de la entidad: event_definition_crf

Nombre: study_ event_definition

Descripción: Guarda los momentos de seguimiento.		
Atributo	**Tipo**	**Descripción**
study_event_definition_id	bigint	Identificador del momento de seguimiento.
study_id	bigint	Estudio activo en el que se realizó el momento de seguimiento.
Name	varchar	Es el nombre del momento de seguimiento.
description	varchar	Descripción del momento de seguimiento.
Type	varchar	Tipo de momento de seguimiento.
status_id	numeric	Estado del momento de seguimiento.
owner_id	numeric	Usuario que insertó o modificó el momento de seguimiento.
date_created	date	Fecha de creación del momento de seguimiento.
date_updated	date	Fecha en que se actualizó el momento de seguimiento.
update_id	numeric	Identificador para cada vez que se modifique un momento de seguimiento.
Dia	varchar	Almacena los días en que ocurre el momento de seguimiento.
tiempo_llenado	integer	Cantidad de días en que se debe llenar un momento de seguimiento.
cronograma_id	bigint	Identificador del cronograma en que se encuentra ese momento de seguimiento.

Tabla 4.6 Descripción de la entidad: study_ event_definition

Nombre: val_case
Descripción: Almacena las reglas que se ejecutarán para cada variable, dígase condición,

derivación y validación.		
Atributo	**Tipo**	**Descripción**
case_id	integer	Identificador de cada uno de los casos.
study_event_id	bigint	Identificador del momento de seguimiento al que pertenece esa regla.
crf_version_id	bigint	Identificador de la versión de la hoja CRD a la que pertenece esa variable.
item_id	bigint	Identificador de la variable.
conditions	text	Almacena la condición que se tiene que cumplir para esa variable.
derivation	text	Almacena la derivación que se tiene que cumplir una variable.
validation	text	Guarda la validación que se debe cumplir para esa variable.
Day	integer	Almacena el día en que se ejecutará una regla.
Generic	integer	Forma de identificar que los casos son únicos.
id_status	bigint	Identificador del estado en que se encuentra esa regla.

Tabla 4.7 Descripción de la entidad: val_case

Nombre: val_dependency_item		
Descripción: Guarda las variables del sistema con sus respectivas variables de dependencia.		
Atributo	**Tipo**	**Descripción**
Id	integer	Identificador de cada dependencia.

dep_item_id	bigint	Identificador de la variable de dependencia.
dep_crf_version_id	bigint	Identificador de la versión de la hoja CRD a la que pertenece la variable de dependencia.
dep_sed_id	bigint	Almacena el identificador de la variable de dependencia.
dep_day	integer	Día perteneciente a la variable de dependencia.
Ítem_id	bigint	Identificador de la variable en cuestión.
sed_id	bigint	Almacena el identificador de la variable en cuestión.
Day	text	Día en que la variable en cuestión dependerá de la variable de dependencia.
id_status	bigint	Identificador del estado en que se encuentra.

Tabla 4.8 Descripción de la entidad: val_dependency_item

Nombre: versioning_map		
Descripción: Contiene la relación de las tablas item y crf_version, almacena los identificadores de ambas.		
Atributo	**Tipo**	**Descripción**
crf_version_id	numeric	Identificador de la versión de una hoja CRD.
item_id	numeric	Identificador de la variable relacionada con esa hoja.

Tabla 4.9 Descripción de la entidad: versioning_map

4.2. Modelo de implementación

Está conformado por los Diagramas de Componentes y de Despliegue, describiendo cómo los elementos del Modelo de Diseño se implementan en términos de componentes, ficheros de código fuente y ejecutables. El Modelo de Implementación describe además cómo se organizan los

componentes de acuerdo con los mecanismos de estructuración, disponibles en el entorno de implementación y en el lenguaje de programación utilizado, y cómo dependen los componentes unos de otros (20).

4.2.1. Diagrama de despliegue

Es un modelo de objetos que describe la distribución física del sistema en términos de cómo se distribuye la funcionalidad entre los nodos de cómputo. Por esta razón, tal distribución tiene una influencia principal para las actividades de implementación. Cada nodo representa un recurso de cómputo.

Para el despliegue del sistema se contará con dos servidores, uno de aplicaciones y otro de base de datos, conectados entre sí mediante el Protocolo de Control de Transmisión/ Protocolo de Internet (TCP/IP, por sus siglas en inglés). Además contará con las estaciones clientes que se comunicarán con el servidor WEB mediante el Protocolo de Transferencia de Híper Texto (HTTP, por sus siglas en inglés) o HTTPS, para que la información viaje de manera segura. Las estaciones clientes contarán con una impresora y se comunicarán mediante los puertos USB o LPT. A continuación se muestra el Diagrama de despliegue en la figura 7.

Diagrama de despliegue

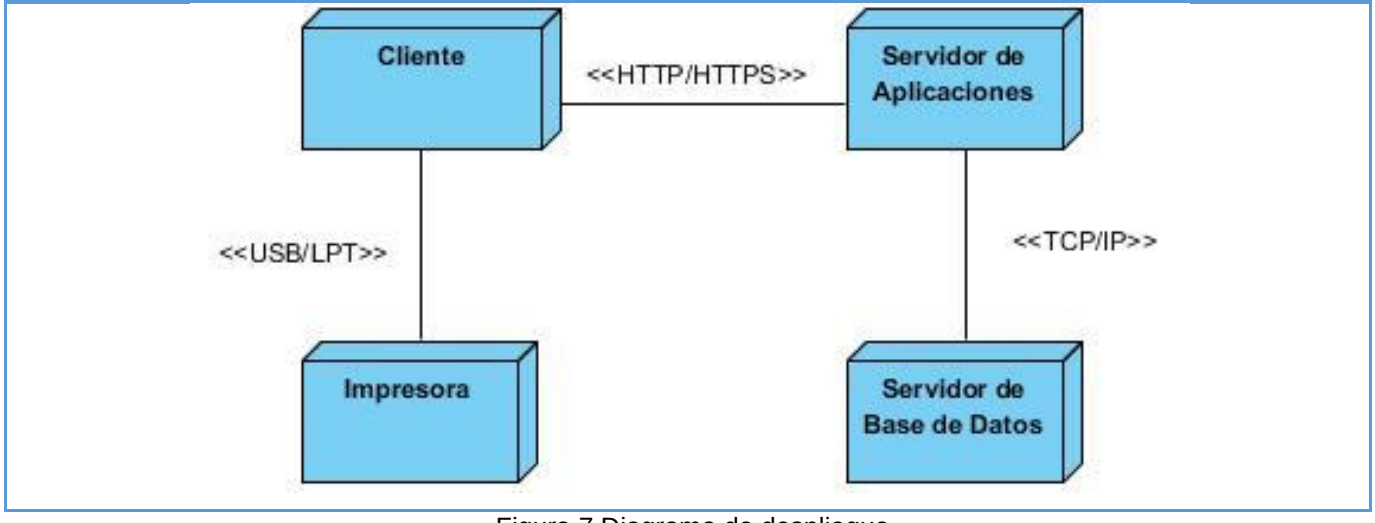

Figura.7 Diagrama de despliegue

4.2.2. Diagrama de componentes

Es una representación gráfica que muestra un conjunto de elementos del modelo tales como componentes, subsistemas o paquetes de implementación y sus relaciones. Permite modelar la vista estática del sistema mostrando las dependencias lógicas entre un conjunto de componentes de

software. Los componentes pueden ser de código fuente, librerías, binarios o ejecutables y tienen relaciones de traza con los elementos del modelo que implementan (21).

Los elementos del modelado se agrupan en paquetes, evidenciándose el patrón arquitectónico MVC. Los componentes encargados de la visualización de la información estarán contenidos en el paquete Vistas, los de la lógica de negocio en el paquete Controlador y los de acceso a datos en el Modelo, como se muestra en la Figura 8.

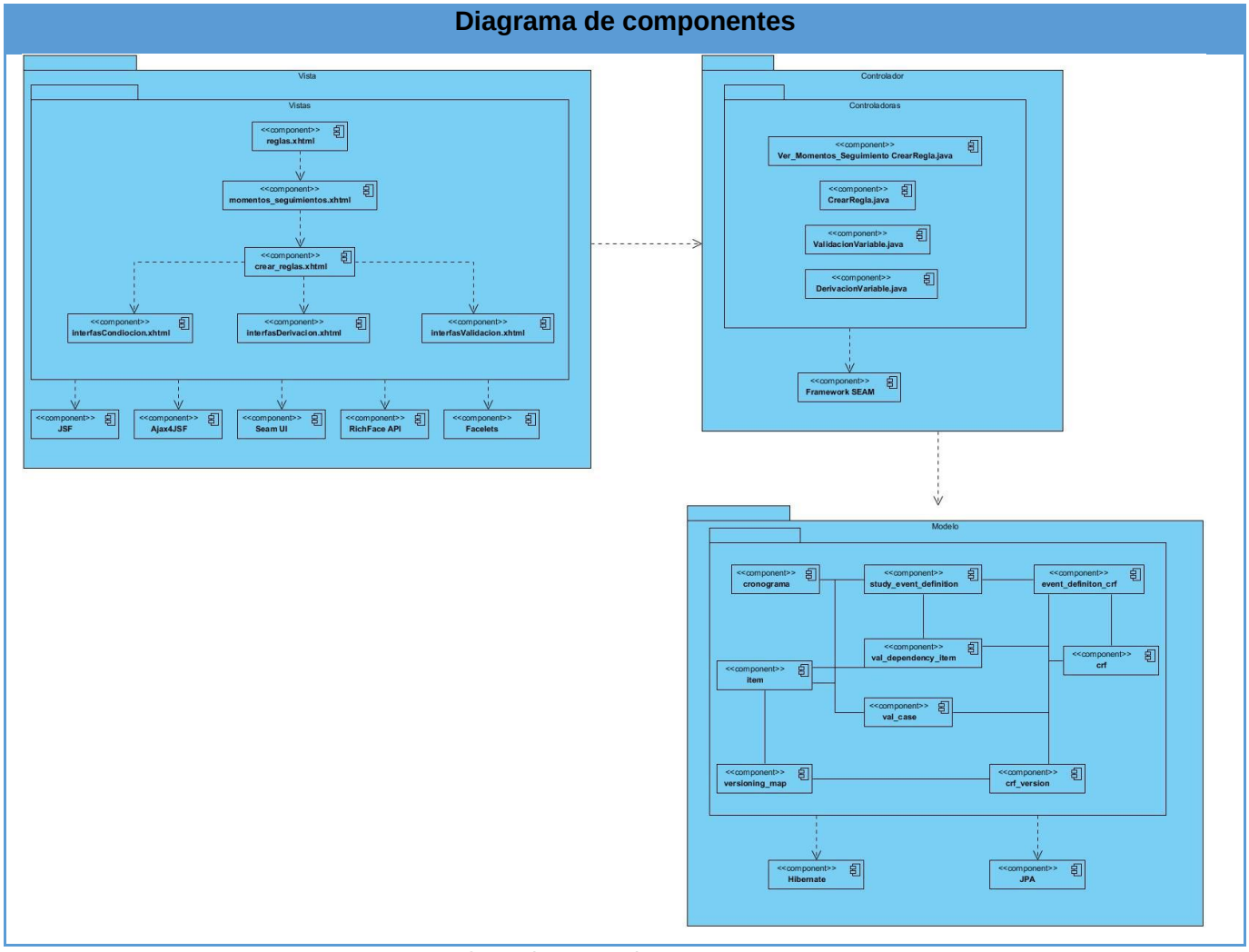

Fig.8 Diagrama de componentes

4.3. Tratamiento de errores

El tratamiento de errores es uno de los aspectos más importantes a tener en cuenta durante el desarrollo del sistema. La validación de la información garantiza la corrección y precisión de todos los valores introducidos en la aplicación, además de lograr elevar la calidad de la misma.

Manejo de excepciones

En el sistema propuesto el tratamiento de excepciones se realiza en las porciones de código propensas a situaciones inesperadas, especialmente donde se realicen inserciones o modificaciones a la base de datos y en las validaciones de los datos insertados por los usuarios en la interfaz. El tratamiento de estos errores se realiza usando la siguiente sentencia:

```
Try
{
//declaración de código que causa la excepción
}
Catch (NombredeExcepcionobj)
{
//código para tratar el error
}
```

Para utilizar las capacidades de internacionalización que brinda JSF, los mensajes de respuesta a cada una de las excepciones son definidos en archivos o ficheros para cada uno de los idiomas que soportará el sistema, lo cual permite una mejor comunicación con el usuario, sin tener que crear versiones de la aplicación para cada idioma.

4.4. Seguridad

La seguridad en el desarrollo de software es un elemento de vital importancia. Proteger los datos que manejan las aplicaciones contra uso indebido constituye una necesidad. Para que cualquier sistema informático sea calificado como seguro debe contar con un correcto equilibrio entre las siguientes características:

Integridad: La información sólo puede ser modificada por quien está autorizado y de manera controlada.

Confidencialidad: La información sólo debe ser accesible para los autorizados.

Disponibilidad: La información debe estar disponible cuando se necesita.

Irrefutabilidad (No repudio): El uso y/o modificación de la información por parte de un usuario debe ser irrefutable, es decir, que el usuario no puede negar dicha acción.

La seguridad del sistema desarrollado la debe garantizar el módulo Configuración del Sistema de Gestión de Ensayos Clínicos, el cual permitirá definir con antelación los roles que interactúan con el sistema, asignándole a cada uno solamente las funcionalidades que le corresponden.

4.5. Estrategias de codificación. Estándares y estilos a utilizar

Un estándar de codificación comprende todos los aspectos de la generación de código, de tal manera que sea práctico y entendible para todos los programadores. Por lo general, incluye pautas sobre cómo nombrar variables y constantes, dónde ubicar comentarios, cómo colocar entre paréntesis, entre otras. No detecta los errores existentes, más bien evita la ocurrencia de estos, lo que permite obtener un código de alta calidad.

Comentarios, separadores, líneas, espacios en blanco y márgenes

Comentarios: Se deben usar los comentarios para realizar descripciones de código y facilitar información adicional que no es legible en el código mismo. Los comentarios deben contener sólo información relevante para la lectura y entendimiento del programa. No deben encerrarse en grandes cuadrados dibujados y no deben incluir caracteres especiales.

Líneas en blanco: Se debe dejar una línea en blanco antes y después de la declaración de una clase o de una estructura y de la implementación de una función.

Espacios en blanco: Se recomienda usar espacios en blanco entre operadores lógicos y aritméticos para lograr una mayor legibilidad del código. No se debe usar espacio en blanco después del corchete abierto y antes del cerrado de un arreglo, luego del paréntesis abierto y antes del cerrado o antes de un punto y coma.

Variables y contantes

El nombre de estas, debe permitir que con sólo leer se conozca el propósito de las mismas.

Clases y objetos

Los nombres de las clases deben comenzar con la primera letra en mayúscula y el resto en minúscula, en caso de que sea un nombre compuesto se empleará notación Pascal Ejemplo: EstaClase ().

El nombre de los atributos de las clases debe comenzar con la primera letra en minúscula y estará en correspondencia al tipo de dato al que se refiere, en caso de que sea un nombre compuesto, la segunda palabra comenzará con mayúscula.

Para nombrar las funciones se deben utilizar verbos que denoten la acción que hacen las mismas. Ejemplo: CrearReglas ()

Identación

Debe ser de dos espacios por bloque de código. No se debe usar el tabulador; pues este puede variar según la computadora o la configuración de dicha tecla. Los inicios ({) y cierre (}) de ámbito deben estar alineados debajo de la declaración a la que pertenecen y deben evitarse si hay sólo una instrucción.

Para el inicio y fin de bloque se recomienda dejar dos espacios en blanco desde la instrucción anterior para el inicio y fin de bloque {}. Lo mismo sucede para el caso de las instrucciones: if, else, for, while, do while, switch, foreach.

Conclusiones

Se desarrollaron los requisitos funcionales especificados anteriormente. Para ello se realizó el Modelo de datos del sistema, el Diagrama de despliegue, el Diagrama de componentes y se realizó una descripción de las tablas de la base de datos. Durante el proceso de codificación se cumplió con los estándares y estilos definidos, lo que permitió obtener un diseño que facilitó el trabajo de todos los programadores y fácil de mantener en el transcurso del tiempo.

Conclusiones

Con el desarrollo de funcionalidades para la gestión de las reglas del Sistema de Gestión de Ensayos Clínicos se ha cumplido con el objetivo general y las tareas de la investigación, por lo que se obtienen las siguientes conclusiones:

- ✓ Fueron analizados los principales conceptos relacionados con el proceso de validación de formularios dinámicos, lo que permitió la comprensión de los mismos y esclarecer el flujo de la investigación.
- ✓ El estudio de los sistemas existentes a nivel internacional y nacional que realizan la validación de formularios dinámicos, demostró que estos no proveen información para dar solución al problema planteado.
- ✓ Para el desarrolla de las funcionalidades se asimiló, la metodología de desarrollo, tecnologías y herramientas que permitieron estructurar de forma entendible la aplicación.
- ✓ Se obtuvieron los artefactos correspondientes a los flujos de trabajo, Modelado de negocio, Requisitos, Diseño y Análisis e Implementación propuestos por la metodología de desarrollo Proceso Unificado de Desarrollo de Software (RUP), que permitieron una mejor comprensión de las funcionalidades desarrolladas.

Recomendaciones

La solución propuesta puede ser enriquecida, por lo que se recomienda:

- ✓ Validar las reglas durante su gestión con el objetivo de comprobar si son correctas semánticamente.

Referencias Bibliográficas

1. **Pike, Rockville.** Informacion de salud para usted. *Ensayos clinicos.* [En línea] 2013. [Citado el: 12 de diciembre de 2014.] http://www.nlm.nih.gov/medlineplus/spanish/clinicaltrials.html..

2. **Systems, Popkin Software and.***Modelado de Sistemas con UML.*

3. **Roldan, Hospital Bouquet.** Turnos por mensaje de texto, una buena experiencia que se extenderá a otros centros de salud y a nuestro vecino país de Chile. [En línea] 2010. [Citado el: 26 de enero de 2014.] http://hospitalbouquetroldan.blogspot.com/2011_04_01_archive.html.

4. **Latina, Osmosis.** JVM ("Java Virtual Machine"). [En línea] 2000. [Citado el: 4 de febrero de 2014.] http://www.osmosislatina.com/java/basico.htm.

5. **Ferguson, Jhon.** [En línea] 2007. [Citado el: 4 de febrero de 2014.] http://www.wakaleo.com/public_resources/jsf-jumpstarter.pdf.

6. **Ghia, Dustin.** Programación Práctica. *JEE5-Fundamentos.* [En línea] 31 de marzo de 2011. [Citado el: 4 de febrero de 2014.] http://programmabilis.blogspot.com/2011/03/i1-fundamentos-de-jee5.html..

7. **otros, Mark Newton y.** WildFly.RichFaces. [En línea] 2008. [Citado el: 4 de febrero de 2014.] http://www.jboss.org/richfaces.

8. *9 Tecnologías, pág 23, PDF.*

9. LIBROSWEB. *HTML y XHTML.* [En línea] 2010. [Citado el: 6 de febrero de 2014.] http://www.librosweb.es/xhtml/capitulo_1/html_y_xhtml.html..

10. **Community, JBoss.** HIBERNATE Tools. *Hibernate Tools for Eclipse and Ant.* [En línea] [Citado el: 6 de febrero de 2014.] http://www.hibernate.org/subprojects/tools.html.

11. **Andes, Universidad de los.** Departamento de Sistemas. *Enterprise Java Bean 3.* [En línea] [Citado el: 6 de febrero de 2014.] http://sistemas.uniandes.edu.co/~isis3702/dokuwiki/lib/exe/fetch.php?media=principal:isis3702-ejb3.pdf.

12. **Grados, Luis Rondon.** JAVA J2EE. *JPA -Java Persistence API.* [En línea] 28 de agosto de 2009. [Citado el: 10 de febrero de 2014.] http://luchorondon.blogspot.com/2009/04/jpa-java-persistence-api.html.

13. **M, Wilmer Jaramillo.** Fedora People. *JBoss Application Server.* [En línea] 2006. [Citado el: 10 de febrero de 2014.] http://wilmer.fedorapeople.org/files/presentations/JBoss.pdf.

14. **Foundation, Eclipse.** Eclipsepedia. *FAQ What is Eclipse?* [En línea] 2006. [Citado el: 10 de febrero de 2014.] http://wiki.eclipse.org/FAQ_What_is_Eclipse%3f.

15. PostgreSQL. *What is PostgreSQL?* [En línea] 20 de mayo de 2009. [Citado el: 15 de febrero de 2014.] http://wiki.postgresql.org/wiki/FAQ#What_is_PostgreSQL.3F_How_is_it_pronounced.3F_What_is_Postgres.3F.

16. Free Download Manager. *Visual Paradigm para UML.* [En línea] marzo de 2007. [Citado el: 15 de febrero de 2014.]

http://www.freedownloadmanager.org/es/downloads/Paradigma_Visual_para_UML_%28M%C3%8D%29_14720_p/.

17. *Capítulo 5.Cliente - Servidor. pág. 9, PDF.*

18. **Kioskea.net.** Entorno cliente/servidor. [En línea] [Citado el: 26 de marzo de 2014.] http://es.kioskea.net/contents/148-entorno-cliente-servidor..

19. **Romero, Yanette Díaz González y Yenisleidy Fernández.** Revista Telem@tica. *Patrón Modelo-Vista-Controlador.* [En línea] 2012. [Citado el: 26 de marzo de 2014.] http://revistatelematica.cujae.edu.cu/index.php/tele/article/view/15..

20. [aut. libro] McGraw-Hill. *Ingeniería de Software, un enfoque práctico.* 2001.

21. Departamento de Sistemas Informáticos. [En línea] http://www.info-ab.uclm.es/asignaturas/42530/pdf/M2tema12.pdf.

Bibliografía

Andes, Universidad de los. Departamento de Sistemas. *Enterprise Java Bean 3.* [En línea] [Citado el: 6 de febrero de 2014.] http://sistemas.uniandes.edu.co/~isis3702/dokuwiki/lib/exe/fetch.php?media=principal:isis3702-ejb3.pdf.

Arquitectura Java. Introducción a EJB 3.1 (I). [En línea] 2 de agosto de 2013. [Citado el: 20 de enero de 2014.] http://www.arquitecturajava.com/introduccion-a-ejb-3-1-i/.

Antwan. Metodología RUP y Metodología UML. [En línea] 24 de junio de 2009. [Citado el: 18 de diciembre de 2013.] http://antwan03.blogspot.com/2009/06/metodologia-rup-y-metodologia-uml.html.

Álvarez, Alex. JBOSS en Centos. [En línea] 28 de junio de 2013. [Citado el: 18 de diciembre de 2013.] http://alexalvarez0310.wordpress.com/2013/06/28/jboss-en-centos/.

Álvarez, Jorge Cortés. Metodologías de desarrollo de software RUP–Proceso Racional Unificado. [En línea] 17 de febrero de 2013. [Citado el: 12 de febrero de 2014.] http://www.slideshare.net/cortesalvarez/metodologa-rup 17-2-2013.

Álvarez Cardona, Mabel, y otros. Desarrollo de un modelo estándar de reporte de eventos adversos en el software ALASCLÍNICAS. [En línea] 3 de junio de 2013. [Citado el: 23 de enero de 2014.] https://compumat.uci.cu/sites/default/files/public/p1330-ponencia-3227_0.pdf.

Baumann, Benjamin. OpenClinica Reference Guide. [En línea] [Citado el: 15 de 03 de 2014.] https://docs.openclinica.com/3.1/rules.

Capítulo 5.Cliente - Servidor. pág. 9, PDF.

Community, JBoss. HIBERNATE Tools. *Hibernate Tools for Eclipse and Ant.* [En línea] [Citado el: 6 de febrero de 2014.] http://www.hibernate.org/subprojects/tools.html.

Desarrolloweb.com. Introducción a JSF (Java Server Faces). Primer artículo de un pequeño manual sobre esta tecnología. [En línea] 21 de febrero de 2006. [Citado el: 20 de enero de 2014.] http://www.desarrolloweb.com/articulos/2380.php.

Departamento de Sistemas Informáticos. [En línea] http://www.info-ab.uclm.es/asignaturas/42530/pdf/M2tema12.pdf.

Definicion.de. Definición de Java. [En línea] [Citado el: 18 de diciembre de 2013.] http://definicion.de/java/.

Eclipse. About the Eclipse Foundation. [En línea] [Citado el: 18 de diciembre de 2013.] https://www.eclipse.org/org/.

Eclipse. About the Eclipse Foundation. [En línea] 2013. [Citado el: 20 de enero de 2014.] https://www.jboss.org/developer/about.html.

Ferguson, Jhon. [En línea] 2007. [Citado el: 4 de febrero de 2014.] http://www.wakaleo.com/public_resources/jsf-jumpstarter.pdf.

Foundation, Eclipse. Eclipsepedia. *FAQ What is Eclipse?* [En línea] 2006. [Citado el: 10 de febrero de 2014.] http://wiki.eclipse.org/FAQ_What_is_Eclipse%3f.

Free Download Manager. *Visual Paradigm para UML.* [En línea] marzo de 2007. [Citado el: 15 de febrero de 2014.] http://www.freedownloadmanager.org/es/downloads/Paradigma_Visual_para_UML_%28M%C3%8D%29_14720_p/.

Ghia, Dustin. Programación Práctica. *JEE5-Fundamentos.* [En línea] 31 de marzo de 2011. [Citado el: 4 de febrero de 2014.] http://programmabilis.blogspot.com/2011/03/i1-fundamentos-de-jee5.html..

Grados, Luis Rondon. JAVA J2EE. *JPA -Java Persistence API.* [En línea] 28 de agosto de 2009. [Citado el: 10 de febrero de 2014.] http://luchorondon.blogspot.com/2009/04/jpa-java-persistence-api.html.

Introducción a Ajax4jsf. [En línea] [Citado el: 12 de 04 de 2014.] http://www.adictosaltrabajo.com/tutoriales/tutoriales.php?pagina=Ajax4Jsf.

Junta de Andalucía. JavaServer Faces(JSF). [En línea] [Citado el: 20 de febrero de 2014.] http://www.juntadeandalucia.es/servicios/madeja/contenido/recurso/101.

Kioskea.net. Entorno cliente/servidor. [En línea] [Citado el: 26 de marzo de 2014.] http://es.kioskea.net/contents/148-entorno-cliente-servidor..

Larman, Craig.*UML y Patrones. Introducción al análisis y diseño orientado a objeto.* [Documento PDF] Uruguay : s.n., 2008.

Latina, Osmosis. JVM ("Java Virtual Machine"). [En línea] 2000. [Citado el: 4 de febrero de 2014.] http://www.osmosislatina.com/java/basico.htm.

LIBROSWEB. *HTML y XHTML.* [En línea] 2010. [Citado el: 6 de febrero de 2014.] http://www.librosweb.es/xhtml/capitulo_1/html_y_xhtml.html..

Limachi, Bernado. Metología RUP. [En línea] 14 de Septiembre de 2012. [Citado el: 18 de diciembre de 2013.] http://www.slideshare.net/bernardolimachi/metodologia-rup-14288208.

M, Wilmer Jaramillo. Fedora People. *JBoss Application Server.* [En línea] 2006. [Citado el: 10 de febrero de 2014.] http://wilmer.fedorapeople.org/files/presentations/JBoss.pdf.

Martínez, Rafael. Sobre PostgreSQL. [En línea] 2 de octubre de 2010. [Citado el: 18 de diciembre de 2013.] www.postgresql.org.es.htm.

Orallo, Enrique Hernández. El Lenguaje Unificado de Modelado (UML). [En línea] [Citado el: 14 de diciembre de 2013.] http://www.disca.upv.es/enheror/pdf/ActaUML.PDF.

otros, Mark Newton y. WildFly.RichFaces. [En línea] 2008. [Citado el: 4 de febrero de 2014.] http://www.jboss.org/richfaces.

Patrones y Antipatrones: una Introducción - Parte I . [En línea] [Citado el: 16 de marzo de 2014.] http://msdn.microsoft.com/es-es/library/bb972242.aspx.

Pautas_Diseño_AplicacionesWEB_CESIM . [Documento PDF] La Habana : Universidad de las Ciencias Informáticas, 2010.

Peluxhebalderas. Entornos de desarrollo integrados (IDE´s). [En línea] 25 de enero de 2013. [Citado el: 20 de enero de 2014.] http://peluxhebalderas.wordpress.com/2013/01/25/entornos-de-desarrollo-integrados-ides-2/.

pgAdmin PostgreSQL Tool. Introduction. [En línea] 10 de octubre de 2013. [Citado el: 20 de febrero de 2014.] http://www.pgadmin.org.

Pike, Rockville. Informacion de salud para usted. *Ensayos clinicos.* [En línea] 2013. [Citado el: 12 de diciembre de 2014.] http://www.nlm.nih.gov/medlineplus/spanish/clinicaltrials.html..

PostgreSQL. *What is PostgreSQL?* [En línea] 20 de mayo de 2009. [Citado el: 15 de febrero de 2014.] http://wiki.postgresql.org/wiki/FAQ#What_is_PostgreSQL.3F_How_is_it_pronounced.3F_What_is_Postgres.3F.

Puebla, Iván García. Slimming Básico de JBoss. [En línea] 7 de febrero de 2008. [Citado el: 18 de diciembre de 2013.] http://www.adictosaltrabajo.com/tutoriales/tutoriales.php?pagina=slimmingBasicoJBoss

Quispe Carita, Vilma, Huamantuco Solorzano, Dante Harry y Vargas Yupanqui, José Luis. METODOLOGIA RUP (RATIONAL UNIFIED PROCESS). [En línea] 2011. [Citado el: 17 de febrero de 2014.] http://www.monografias.com/trabajos-pdf4/metodologia-rup-una-puno/metodologia-rup-una-puno.shtml.

Ramos, Juan Alonso. Introducción a Ajax4jsf. [En línea] [Citado el: 20 de febrero de 2014.] http://www.adictosaltrabajo.com/tutoriales/tutoriales.php?pagina=Ajax4Jsf.

Roldan, Hospital Bouquet. Turnos por mensaje de texto, una buena experiencia que se extenderá a otros centros de salud y a nuestro vecino país de Chile. [En línea] 2010. [Citado el: 26 de enero de 2014.] http://hospitalbouquetroldan.blogspot.com/2011_04_01_archive.html.

Romero, Yanette Díaz González y Yenisleidy Fernández. Revista Telem@tica. *Patrón Modelo-Vista-Controlador.* [En línea] 2012. [Citado el: 26 de marzo de 2014.] http://revistatelematica.cujae.edu.cu/index.php/tele/article/view/15..

Software.com.ar. Visual Paradigm para UML . [En línea] [Citado el: 23 de enero de 2014.] http://www.software.com.ar/visual-paradigm-para-uml.html.

Suárez, José Manuel Sánchez. Introducción a RichFaces. [En línea] 1 de febrero de 2010. [Citado el: 20 de febrero de 2014.] http://www.adictosaltrabajo.com/tutoriales/tutoriales.php?pagina=richFacesJsfIntro.

Systems, Popkin Software and.*Modelado de Sistemas con UML.*

Valencia, Freddy. JavaServer Facelets. [En línea] [Citado el: 20 de febrero de 2014.] http://es.scribd.com/doc/112815385/JavaServer-Facelets.

Welicki, León. Patrones y Antipatrones: una Introducción - Parte I. [En línea] [Citado el: 16 de marzo de 2014.] http://msdn.microsoft.com/es-es/library/bb972242.aspx.

9 Tecnologías, pág 23, PDF.

[aut. libro] McGraw-Hill. *Ingeniería de Software, un enfoque práctico.* 2001.

Anexos

Anexo 1: Entrevista para el sistema Synta

Entrevista con el ingeniero Yunior Pacheco

Pregunta: ¿Qué es Synta?

Respuesta: Es un sistema para el control farmacológico.

P: ¿Para qué se usa?

R: Es usado para identificar a quienes fueron entregadas las recetas médicas en cada unidad de Cuba, además controla los medicamentos consumidos a nivel nacional y la cantidad de pacientes inscritos en cada unidad de salud, farmacia.

P: ¿Qué es el sistema Nomencladores?

R: Es un sistema de configuración y flexible ante posibles cambios de la información poco variable en el tiempo.

P: ¿Cómo se realiza la validación de los formularios dinámicos en este sistema?

R: Cuando se crean nuevos campos se validan usando un mecanismo implementado con la ayuda de librerías de ExtJS, o sea cuando este es creado el sistema muestra una interfaz para definir las propiedades que va tener este, si solo permite entero, si la longitud del campo va a ser 12, etc.

Anexo 1.1: Entrevista para el Sistema de Información Hospitalaria (HIS)

Pregunta: ¿Qué es el HIS?

Respuesta: Es un sistema de gestión de información.

P: ¿Para qué se usa?

R: Este sistema permite la recolección, almacenamiento, procesamiento, recuperación y comunicación de los datos de atención al paciente, permite además gestionar y controlar los recursos de cada una de las áreas de las instituciones hospitalarias.

P: ¿Cómo se realiza la validación en el sistema?

R: En este sistema se utilizan varios tipos de validaciones, pueden ser, del lado del cliente que se realizan a través de funciones JavaScript que son utilizadas generalmente para validar campos obligatorios, los componentes de fecha y dependencia entre ellos. Se utilizan además los que provee JSF para chequear valores de mínimo y máximo de un campo, campos requeridos y tamaño mínimo de un campo. Por su parte del lado del servidor se realizan validaciones propias del sistema o los procesos que chequean valores aceptables o dependencias entre dichos valores. En este sistema además se encuentra el módulo Laboratorio, donde existen formularios dinámicos a los cuales se les realiza la validación en las clases controladoras usando también funciones JavaScript.

Anexo 2: Interfaces del sistema adicionar variable de dependencia

Figura A2: Adicionar variable de dependencia

Figura A2: Adicionar variable de dependencia

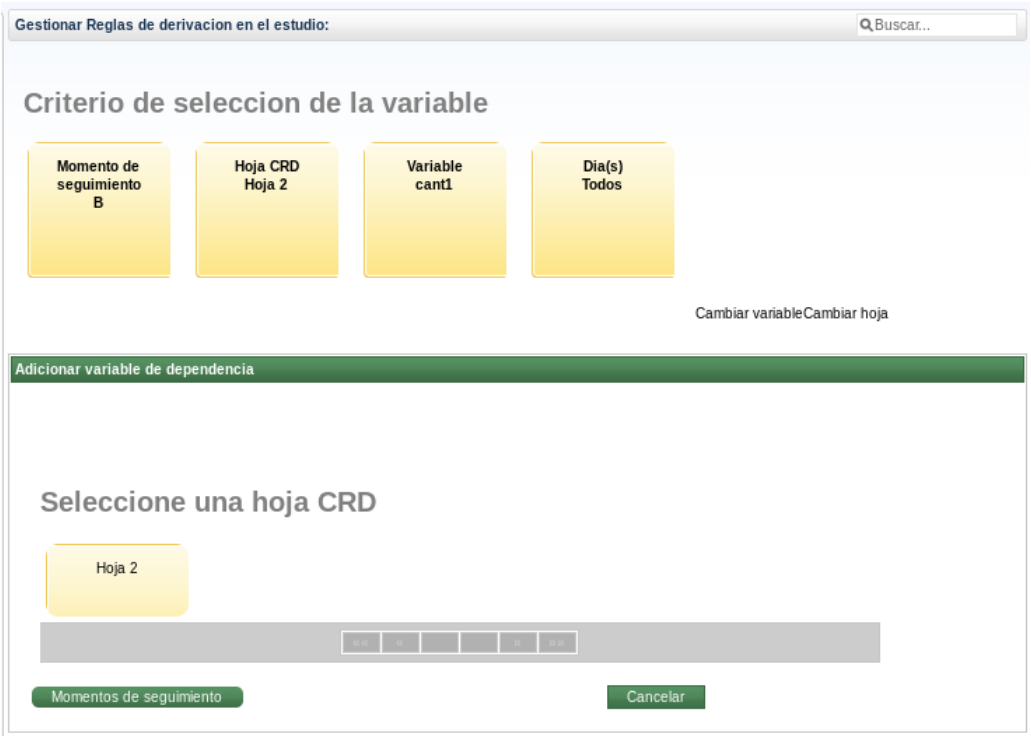

Figura A2: Adicionar variable de dependencia

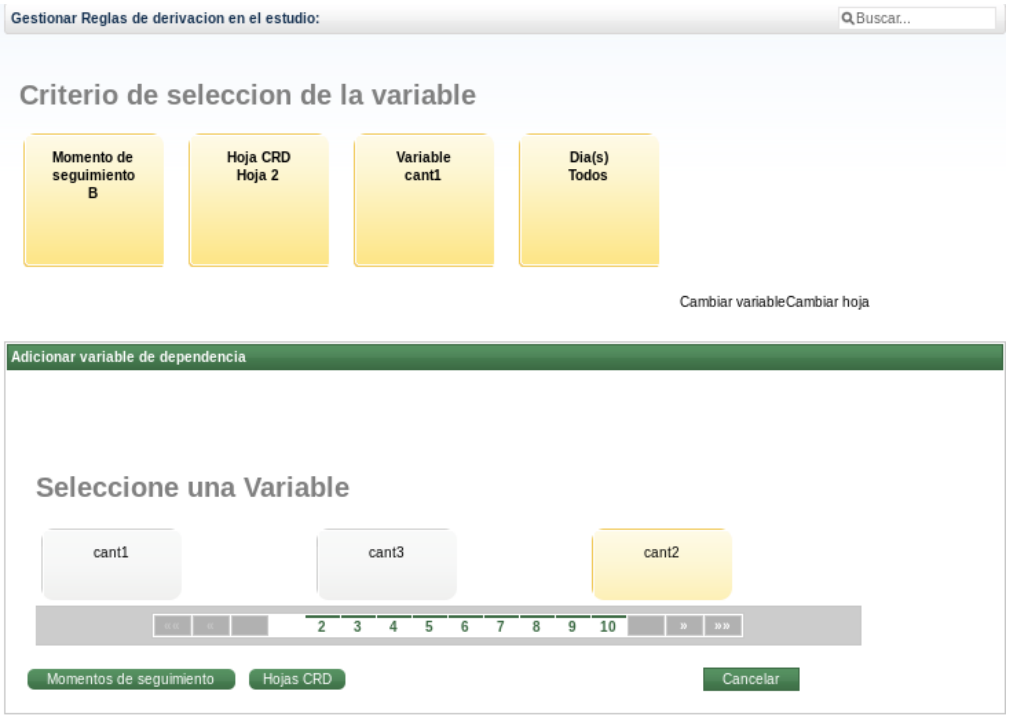

Figura A2: Adicionar variable de dependencia

Figura A2: Adicionar variable de dependencia

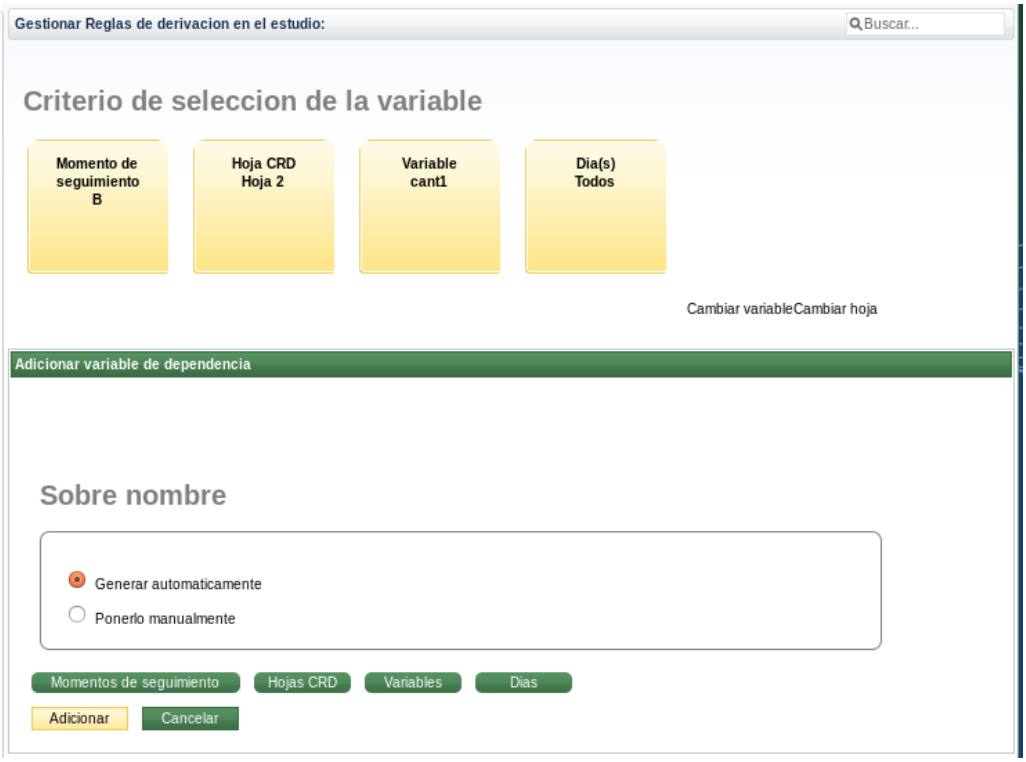

Figura A2: Adicionar variable de dependencia

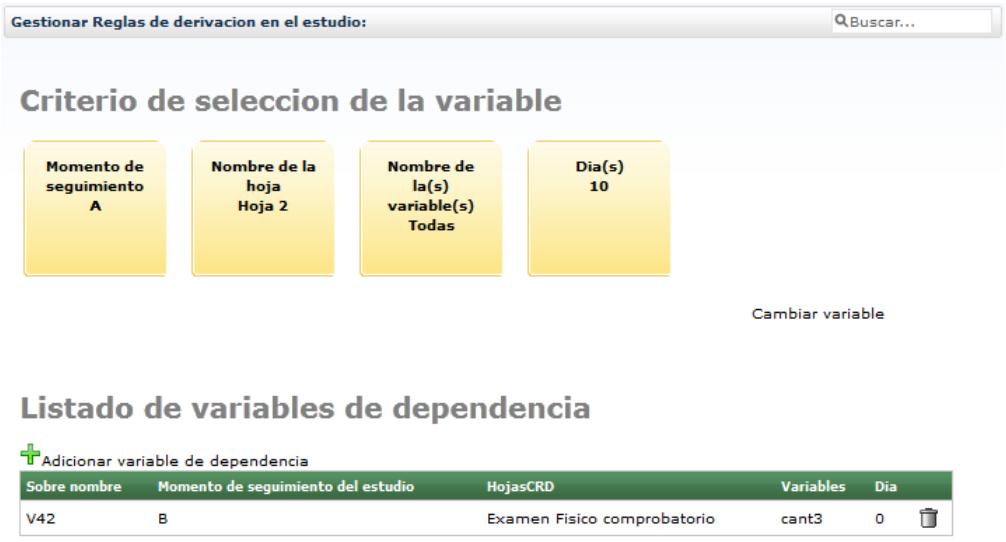

Figura A2: Adicionar variable de dependencia

Glosario de términos

Cronograma: Representa la planificación de un conjunto de Momentos de Seguimiento a los cuales estarán asociados uno o un conjunto de Hojas CRD, el cual será aplicado a un cierto número de sujetos. Los datos que se recogen en esta clase son: la fecha en que se iniciará el estudio, la fecha de fin, entre otros.

Etapa: La clase Etapa representa los períodos en los cuales se realizará el estudio. Estos períodos serán definidos en dependencia del tipo de estudio que se esté realizando, los mismos pueden ser:

- ✓ Evaluación.
- ✓ Tratamiento.
- ✓ Seguimiento.

MS: Describe un tipo de seguimiento que sucede durante la realización del estudio, estos pueden ser programados o no programados.

MS Programado: Describe aquel MS que se espera que suceda para cada sujeto, como parte del progreso normal del estudio.

MS no Programado: Describe un MS que no se espera que suceda, pero que puede suceder si las circunstancias lo requieren.

Reglas: Conjunto de normas que se establecen para que ciertos datos sean válidos.

Variables: Son los campos que componen el formulario.

Datos: Es la información que se recoge en los campos.

Centro: Recoge información relacionada a los sitios u hospitales tales como: nombre del centro, provincia, país, código postal, teléfono de contacto y correo electrónico.

Hojas CRD: Describe, cómo se recoge toda la información relacionada a un sujeto con un determinado estudio. Esta recoge varios datos como el nombre del CRD, la fecha de inicio, descripción, entre otros.

Cronograma Específico: Representa la asociación de la planificación general de momentos de seguimiento del estudio a un paciente específico. Contiene la fecha para la cual se planificó cada momento de seguimiento, la etapa a la que pertenece cada uno y el estado.

Investigador Principal: Es el médico que se encuentra al frente de los pacientes que serán sometidos a determinado EC.

Investigador Promotor: Es el encargado de orientar la recogida de datos de un paciente en la hoja CRD. Tiene acceso a visualizar, imprimir y salvar cualquier información relacionada con el ensayo. Es el encargado de firmar las reglas de validación y el cronograma completado por el Gerente de Datos, aprobando y desaprobando el diseño.